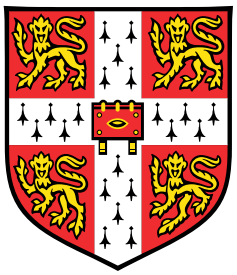
End-to-end Contextual Speech Recognition and Understanding



Guangzhi Sun

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Trinity College June 2023

Declaration

I hereby declare that this dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the The University of Cambridge or any other University of similar institution except as declared in the Preface and specified in the text. Some of the material has been presented at or submitted to international conferences and journals (Sun et al., 2021a,c, 2022a,b, 2023a,b). This dissertation contains fewer than 65,000 words including appendices, footnotes, tables and equations, but excluding the bibliography, and has fewer than 150 figures.

Guangzhi Sun
June 2023

Acknowledgements

I would like to first devote my deepest gratitude to my PhD supervisor, Prof. Phil Woodland. I am immensely grateful for the opportunity Phil offered me to pursue research in speech and language technology in the summer of 2018 where my journey of research began. I would like to extend my heartfelt thanks to Phil for believing in my potential and providing me with the opportunity to undertake this challenging and rewarding PhD research. The countless hours dedicated to refining research ideas, research directions and papers have been constantly motivating, supporting and encouraging me to move forward throughout my entire PhD study. In addition, Phil with his academic integrity and professionalism has been an exceptional role model that has, and will continue to profoundly influence my research.

I would like to express my profound gratitude to Prof. Chao Zhang for his invaluable contributions to my research during my PhD studies. Chao demonstrated an impressive level of engagement in the research discussions, offering numerous insightful suggestions that greatly enhanced the quality of my work. I would also like to devote my sincere gratitude to Dr Qiujia Li and Dr Florian Kreyssig. They have been immensely supportive throughout my PhD research, offering timely suggestions for my research and cheering me up during the difficult pandemic time.

I would like to thank Prof. Steve Young who offered inspirational suggestions and introduced valuable connections that significantly helped my research. I deeply appreciate the precious discussions at the early stages of my research with Prof. Mark Gales who has served as my PhD advisor, and the indispensable support from Dr Kate Knill. I would also like to thank my college tutor, Dr Nicolas Bell, and Trinity College fellows Prof. Joan Lasenby, Prof. Per Ola Kristensson, Prof. Hugh Hunt, Prof. Stuart Haigh and Prof. Matthew Juniper who have been supportive in all aspects of my PhD life. I also want to thank my colleagues at PolyAI Ltd, Dr Ivan Vulić and Dr Paweł Budzianowski for their thought-provoking discussions and their efforts in real customer data collection. I also sincerely appreciate the effort of my collaborators from the Machine Intelligence Lab, Xianrui Zheng, Evonne Lee and Danyi Liu, and the warm support from my lab mates Dongcheng Jiang, Wen Wu, Xiaodong Wu, Keqi Deng, Potsawee Manakul, Dr Yu Wang, Dr Qingyun Dou, Dr Linlin Wang and Dr Mengjie Qian who have helped me in various ways. I must also thank my best

friends, Junzhe Zhao, Yuchen Zhou, Yufeng Zhao and Anran Jin, who always cheered me up when I was frustrated. I would also extend my thank to Cambridge Trust for their financial support which made my PhD study possible.

Finally, I would like to devote my most profound gratitude to my wife, Xiao Zhan, who always unconditionally supported me, encouraged me and accompanied me, especially during difficult times. And to my parents who have been unwaveringly standing by my side, offering their unreserved support in every conceivable manner. I would like to dedicate this thesis to them.

Table of contents

List of figures

List of tables

Nomenclature

List of Abbreviations

Adam Adaptive Momentum

AED Attention-based Encoder Decoder

AM Acoustic Model

AMI Augmented Multi-party Interaction

ANN Artificial Neural Networks

ASR Automatic Speech Recognition

BERT Bi-directional Encoder Representations from Transformer

BLMD Biasing-word-driven Language Model Discounting

BPE Byte Pair Encoding

BPTT Back-propagation Through Time

CE Cross Entropy

CER Character Error Rate

CLM Class Language Model

CNN Convolutional Neural Networks

CTC Connectionist Temporal Classification

DB Deep Biasing

DNN Deep Neural Networks

DR Density Ratio

DST Dialogue State Tracking

DSTC Dialogue State Tracking Challenge

FBANK Mel-scale Filter Bank

FC Fully-connected

GCN Graph Convolutional Network

GD Gradient Descent

GMM Gaussian Mixture Model

GNN Graph Neural Network

GPT Generative Pre-trained Transformer

GPU Graphics Processing Unit

GRU Gated Recurrent Unit

HMM Hidden Markov Model

KA2G Knowledge-Aware Audio Grounded

KB Knowledge Base

LLM Large Language Model

LM Language Model

LSTM Long Short-term Memory

MAP *Maximum a Posteriori*

MBR Minimum Bayes Risk

MBWE Minimum Biasing Word Error

MFCC Mel-Frequency Cepstral Coefficient

MHA Multi-Head Attention

ML Maximum Likelihood

MLE Maximum Likelihood Estimation

MLP Multi-layer Perceptron

MMI Maximum Mutual Information

MWE Minimum word error

NLU Natural Language Understanding

OCR Optical Character Recognition

OOV Out-Of-Vocabulary

OSC One-Step Constrained

PE Positional Encoding

PLM Pre-trained Language Model

PLP Perceptual Linear Prediction

PPL Perplexity

QA Question Answering

R_s-WER Rare Slides Word Error Rates

R-WER Rare Word Error Rate

ReLU Rectified Linear Unit

RNN Recurrent Neural Networks

RNN-T Recurrent Neural Network Transducer

RPE relative positional encoding

SCM Source-Channel Model

SF Shallow Fusion

SGD Stochastic Gradient Descent

SLU Spoken Language Understanding

SLURP Spoken Language Understanding Resource Package

SPB Slot Probability Biasing

SS Slot shortlist

SSL Self-supervised Learning

SVG Slot-Value Generator

T5 Text-To-Text Transfer Transformer

TCPGen Tree-Constrained Pointer Generator

ToD Task-oriented Dialogue

Tree-RNN Tree Recursive Neural Network

WER Word Error Rate

WFST Weighted Finite State Transducer

List of Symbols

</s> End of sequence token

<s> Begin of sequence token

η Learning rate

κ Penalty coefficient

$\mathbf{A}$ Graph affinity matrix

$\mathbf{a}$ Attention weight vector

$\mathbf{b}^l$ Bias vector for layer l

$\mathbf{c}$ Context Vector

$\mathbf{D}$ Graph diagonal degree matrix

$\mathbf{I}$ Identity matrix

$\mathbf{k}$ Attention key vector

$\mathbf{L}$ Graph Laplacian

$\mathbf{o}$ Observation vector

$\mathbf{q}$	Attention query vector
$\mathbf{r}$	Relative positional encoding
$\mathbf{v}$	Attention value vector
$\mathbf{W}$	Word sequence
$\mathbf{W}^l$	Weight matrix for layer l
$\mathbf{x}$	Input feature vector
$\mathbf{y}$	Output vector
∇	Gradient operator *del*
$\odot$	Element-wise product
$\mathcal{D}$	Dataset
$\mathcal{H}$	Set of possible alignments in RNN-T
$\mathcal{L}(\cdot)$	Loss function
$\mathcal{N}(\cdot)$	Normal distribution
$\mathcal{W}$	Word error rate measure
$\mathcal{Y}$	Set of word pieces
$\sigma(\cdot)$	Sigmoid activation function
θ	Parameters of the neural network
$\varnothing$	Blank symbol for RNN-T
H	Alignment in RNN-T
$Swish(\cdot)$	Swish activation function
y	Output label
[CLS]	BERT next sentence prediction token

Chapter 1

Introduction

Conversational artificial intelligence (AI) systems, such as voice assistants and meeting transcription systems, have become increasingly prominent in various aspects of daily life, particularly with the advancement of deep learning techniques. Among the many tasks involved in conversational AI systems, automatic speech recognition (ASR) and spoken language understanding (SLU) are two essential components. ASR refers to the transcription of a speech signal sequence into a linguistic token sequence such as a sequence of words, while SLU involves the extraction of relevant text information from the speech, such as user intent or the name of a restaurant that has been mentioned. The performance of both ASR and SLU has benefited from the implementation of deep neural networks (DNN), which enable the development of end-to-end trainable systems that can tackle both tasks as sequence-to-sequence problems.

End-to-end trainable neural-based systems for ASR and SLU are extensively used in applications with dynamically changing contexts, which require them to handle important content words that may be infrequent or absent in the training data. However, such systems often exhibit suboptimal performance on such words, since they are trained with static parameters on specific datasets. As a result, it becomes both crucial and challenging for these end-to-end trainable systems to incorporate dynamic contextual knowledge to enhance the performance on these rare and infrequent words and phrases. One promising approach to address this challenge is contextual biasing.

Contextual biasing aims at integrating contextual knowledge into end-to-end trainable ASR and SLU systems and has a broad application in a range of domains. Such contextual knowledge is typically represented in the form of a *biasing list*, which contains words or phrases (referred to as *biasing words*) that are likely to occur in a given context. Sources of biasing lists may include user contact books, recently visited websites, information from presentation slides or the ontology of a dialogue system. Biasing words are often rare content

words, such as nouns or proper nouns, that are critical to downstream tasks and therefore carry significant value. Incorporating them in the biasing list can improve the system performance on those words, which also leads to better controllability over these end-to-end systems.

Dedicated approaches have been proposed to achieve contextual biasing based either on shallow fusion (SF) or deep context methods. SF-based biasing involves interpolating the model output with a special weighted finite-state transducer (WFST) or language model (LM) adapted to incorporate contextual knowledge. Meanwhile, deep context methods usually rely on an attention mechanism to extract information from the biasing list into a compact vector representation which is used as an auxiliary input to end-to-end systems.

This thesis focuses on the tree-constrained pointer generator (TCPGen) as a novel neural component for contextual biasing that has been successfully applied to both end-to-end ASR and SLU systems. As a trainable neural component, TCPGen builds a neural shortcut to directly bias the output distribution with a dynamic interpolation factor to improve system flexibility. TCPGen structures the biasing list into a prefix tree which enables it to handle large biasing lists containing thousands of words. As a result, TCPGen combines the advantages of both SF and deep context methods. This thesis also includes dedicated training and decoding methods for TCPGen, graph neural networks (GNN) prefix-tree encodings, and the application of TCPGen to SLU tasks.

In this chapter, end-to-end ASR and SLU are introduced. Then the thesis outline is presented, followed by highlighting the main contributions.

1.1 ASR and SLU

This thesis covers the topic of contextual knowledge integration in end-to-end neural-based ASR and SLU systems, where both ASR and SLU problems are introduced below.

1.1.1 End-to-end ASR

ASR is a task that involves the transcription of speech signals into linguistic tokens. The development of high-performing ASR systems has been a topic of active research for several decades. Traditionally, ASR has been achieved using the noise source-channel model (SCM), which treats the observed speech signal as the output of a noisy channel with the intended text as its input. Typically, hidden Markov models (HMMs) are employed as the backbone modelling approach, with deep neural networks (DNNs) used to model the likelihood of the observation sequence based on a hidden state sequence. The SCM adopts a modular framework that decomposes the ASR problem into smaller sub-tasks, such as

acoustic modelling and language modelling. A multi-stage pipeline, which includes front-end processing, feature normalisation, sequence training, adaptive training, and decoding, is used to combine the various components and produce the desired output. It is easier to incorporate structured contextual knowledge in this modular framework, such as a lexicon or WFSTs, which allows the recogniser to perform robustly, even with limited resources.

In contrast to the modular design in SCM, recently, the rapid advancement of learning algorithms and computing hardware has led to the design of single end-to-end trainable models. These models have achieved comparable or even superior performance compared to SCM in ASR. This thesis specifically concentrates on two such models, namely the attention-based encoder-decoder (AED) model and the recurrent neural transducer (RNN-T) model, both of which have been integrated with TCPGen.

The architecture of AED comprises three main components, namely the encoder, attention mechanism, and decoder. In the context of ASR, the encoder transforms input acoustic features into a sequence of hidden representations, while the decoder uses an attention mechanism to generate one output token at a time based on all previous outputs. Unlike SCM-based systems, AED models jointly optimise the acoustic and language models. Attention in AED requires seeing the entire audio sequence before generating transcriptions, making it challenging to recognise in streaming mode.

RNN-T is an alternative end-to-end system that has been widely used for ASR, which is easy to work in streaming mode. RNN-T contains an encoder, a predictor and a joint network. As AED, the encoder encodes the input acoustic features into a sequence of hidden vectors. The predictor in RNN-T is analogous to the LM in an SCM, which encodes the previously decoded tokens into a compact vector for next-token prediction. The joint network combines the acoustic and language information from the encoder and predictor respectively. RNN-T makes a prediction of zero or more tokens followed by a null symbol, $\varnothing$, at each encoder step, and the recognition result is obtained by removing $\varnothing$ symbols.

As end-to-end trainable neural systems, both AED and RNN-T integrate all knowledge into a single static system and often exhibit degraded performance on content words that are rare or unseen in the training set, and contextual knowledge serves as a promising remedy to improve the performance of those words. However, compared to the SCM approach which integrates contextual knowledge relatively easily by customising certain modules e.g. the lexicon, it is more challenging for end-to-end systems. Therefore, it is imperative to investigate both efficient and effective dynamic contextual knowledge integration methods for end-to-end systems.

1.1.2 End-to-End SLU

The goal of SLU is to accurately comprehend the intent and context of spoken language and to extract relevant information from it. This typically involves slot filling and intent classification. Intent classification is the task to classify the user intent in an utterance into one of the predefined intent types. Slot filling aims at filling the correct value for predefined slots (e.g. restaurant and hotel names). The development of accurate SLU systems is crucial for enabling effective human-machine interactions in many conversational AI systems such as task-oriented spoken dialogue systems.

SLU has traditionally been achieved using rule-based and statistical methods, but with recent advances in deep learning techniques, neural-based models have become increasingly popular. Earlier approaches adopted a two-stage pipeline where an ASR model is first employed to obtain the transcription, and a natural language understanding (NLU) component, which is trained with text-only data, is then run on the transcription to get intent and slot-filling information. While the intent is determined by performing classification based on the compact vector extracted using a DNN representing the entire utterance, slot filling is often achieved by sequence tagging approaches, where tags indicating whether a token belongs to a specific slot are predicted for all tokens in the sequence. Alternative approaches also re-formulate slot filling as a sequence-to-sequence natural language generation (NLG) task, where the value of a slot is generated by providing text-based queries to the system (e.g. what is the name of the restaurant). More recently, both tagging and generation systems for NLU significantly benefit from fine-tuning a pre-trained large LM (LLM) with intent or slot-filling output layers.

1.2 Thesis Outline

This thesis starts with relevant background knowledge about DNNs, ASR and SLU. Then, the contributions in contextual biasing for ASR and SLU using TCPGen are provided. In this section, the overview of each chapter is given, including references to the corresponding papers.

- Chapter 2 covers the fundamental elements of DNNs, including their basic building blocks such as types of neural network layers and activation functions. It then introduces training algorithms and regularisation methods that improve training for DNNs. It also covers the fundamentals of Transformer and foundation models.

- Chapter 3 first provides a brief review of SCM, and then introduces end-to-end trainable ASR and SLU systems. For ASR, AED and RNN-T are introduced respectively,

including various network structures of the systems, the training and the inference algorithms. For SLU, tagging and sequence-generation-based systems are introduced, together with the use of pre-trained LMs in SLU.

- Chapter 4 proposes the TCPGen component for contextual biasing in end-to-end ASR systems (Sun et al., 2021c). This chapter introduces the design considerations of TCPGen as well as how it can be applied to both AED and RNN-T systems. Moreover, a dedicated training algorithm that directly optimises the error rate of biasing words is also proposed in this chapter, together with an internal LM discounting method for TCPGen to reduce the effect of certain language patterns that the model memorises from the training data (Sun et al., 2022a).

- Chapter 5 proposes the use of GNNs to encode the prefix-tree structure in TCPGen (Sun et al., 2022b), which achieves the lookahead functionality that boosts TCPGen performance. This chapter also introduces an audio-visual contextual ASR pipeline where contextual knowledge is extracted from the visual modality via images of presentation slides. This demonstrates the potential benefits of contextual biasing and TCPGen in many other academic and educational applications.

- Chapter 6 extends the application of TCPGen to SLU tasks by proposing a slot shortlist prediction method to obtain a more focused biasing list from structured KBs, together with a TCPGen-style neural shortcut to bias the slot filling output (Sun et al., 2023b). Furthermore, a knowledge-aware audio-grounded (KA2G) slot-filling framework based on TCPGen (Sun et al., 2023a) is proposed by formulating slot-filling as a sequence generation task taking speech inputs.

- The key conclusions and contributions are summarised in Chapter 7. Based on the current findings with TCPGen, Chapter 7 also suggests several possible directions for future work to further explore contextual biasing in conversational AI systems.

1.3 Contributions

The key contributions of this thesis are as follows.

- An effective contextual biasing component, TCPGen, is proposed for end-to-end trainable ASR and SLU. This approach establishes a dynamic neural shortcut to the model output that is both flexible and effective. TCPGen also constructs a prefix tree which allows biasing lists containing thousands of words or phrases to be handled efficiently.

- The minimum biasing word error (MBWE) training algorithm is proposed for contextual biasing with TCPGen, which can be applied to both AED and RNN-T. A biasing-word-driven LM discounting (BLMD) algorithm is also proposed for decoding end-to-end systems with TCPGen.
- A GNN-based prefix-tree encoding that enables a lookahead functionality is proposed for TCPGen. In particular, the tree recursive neural network (tree-RNN), graph convolutional neural networks (GCN) and GraphSAGE are investigated for tree encodings.
- An audio-visual contextual ASR pipeline is proposed as a multi-modal setup for ASR. This pipeline proposes to use contextual knowledge from the visual modality to bias the ASR system, which showcases the potential advantages of incorporating contextual biasing and TCPGen in various academic and educational domains in the future.
- TCPGen is also integrated into end-to-end SLU systems, where a slot shortlist can be used to obtain focused biasing lists from the KB. Shortcuts are also built from the TCPGen component in ASR to the slot-filling output, in both the tagging-based system and the proposed sequence generation-based KA2G framework.
- The proposed TCPGen component for ASR and SLU has been thoroughly evaluated across a wide range of English datasets, including LibriSpeech audiobook corpus, augmented multiparty interaction (AMI) meeting data, SLU resource package (SLURP) data, dialogue state tracking challenge (DSTC) data and the CONCIERGE multi-turn dialogue data from industry.
- TCPGen with GNN encodings in ASR achieved over 60% relative WER reduction on rare words on LibriSpeech, and over 30% on AMI slides rare words using the audio-visual contextual ASR pipeline. The KA2G framework achieved further 1.4% absolute increase in SLU-F1, with an 11.2% absolute SLU-F1 increase on unseen entities compared to a strong pipeline approach.

Chapter 2

Deep Neural Networks

Artificial Neural Networks (ANN) are mathematical models that draw inspiration from biological neural networks. These models can be viewed as a type of function that maps input features or representations to the desired output space using non-linear transformations (Bishop, 1995). ANNs can be represented as a directed and weighted graph composed of interconnected groups of nodes. Each connection has an associated weight (the model parameters), and each node is associated with a non-linear activation function. Deep Neural Networks (DNNs) are a major subclass of ANNs, where multiple layers of nodes are present between the input and output layers. DNNs have the ability to approximate complex functions and are universal functional approximators. Increasing the depth and width of DNNs allows more complex functions to be approximated (Goodfellow et al., 2016).

The deep learning paradigm has provided a robust and versatile framework for the supervised learning of DNNs, which can be applied to classification or regression tasks. By leveraging an adequate amount of input-output training data, the parameters of the model can be learned to optimise a given objective criterion through the error backpropagation mechanism (Rumelhart et al., 1986). DNNs have demonstrated superior performance on various challenging tasks in computer vision, natural language processing, and speech processing domains. The advancement in model architectures, training techniques, access to large-scale datasets, and high-performance computing resources has led to the very widespread use of DNNs in the field. In particular, in the area of speech processing, the use of DNNs has revolutionised modelling approaches and performance.

This chapter presents an overview of DNNs and their learning procedures, which are foundational concepts that will be used frequently throughout the thesis. It begins by introducing the fundamental components of DNNs and subsequently explains diverse optimisation and regularisation techniques that can be employed to improve the training of DNNs. Finally, it introduces the Transformer (Vaswani et al., 2017) which has become the dominant technique

in sequence-to-sequence modelling tasks. Foundation models (Devlin et al., 2019) that are pre-trained with self-supervised learning (SSL) on large-scale datasets that can then be fine-tuned to benefit downstream tasks will also be introduced alongside the Transformer.

2.1 DNN Building Blocks

2.1.1 Multilayer Perceptron

The multi-layer perceptron (MLP) (Bishop, 1995, Rosenblatt, 1962) is a fundamental type of DNN. The aim of an MLP is to learn a set of parameters θ to achieve the desired mapping function $f(\mathbf{x};\theta)$ which maps the input vector $\mathbf{x}$ to the output vector $\mathbf{y}$, often via a series of fully-connected (FC) layers. The form of an FC layer is defined in Eqn. 2.1.

$$\mathbf{h}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}), \tag{2.1}$$

where $\mathbf{h}^{(l-1)}$ is the input to the l-th layer which is either the output of the previous layer $l-1$ or the input vector $\mathbf{x}$ if $l = 1$. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weight matrix and the bias vector containing trainable model parameters for this layer and f is a non-linear scalar mapping operating on each element of a vector, known as the activation function which will be discussed later in this section. In an MLP, the output from the previous layer is connected to the input of the next layer to form a chain, and the information flow is uni-directional from the input to the output, as shown in Fig. 2.1. It is therefore a feed-forward model. Since intermediate layers do not have corresponding desired outputs in the training data, these are called *hidden layers*. The depth of a DNN is measured by the number of hidden layers.

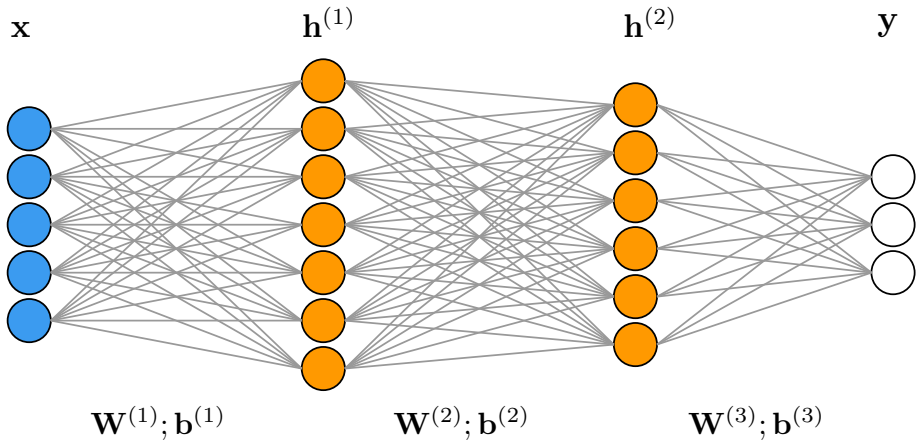


Fig. 2.1 The architecture of a DNN with 2 hidden layers and one output layer.

2.1.2 Activation Functions

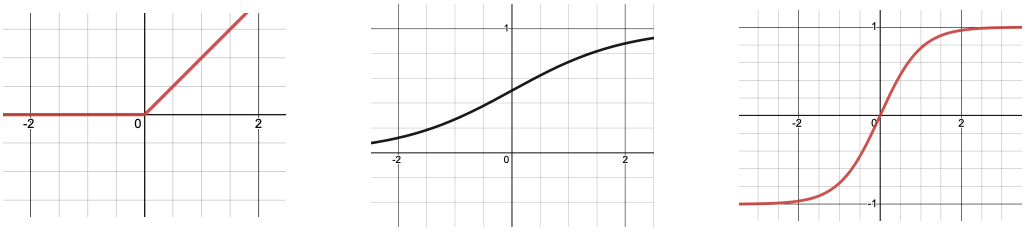
The activation function, $f(\cdot)$, is a crucial component of MLPs, enabling deep neural networks to perform effective non-linear modelling. Activation functions typically possess several desirable properties, such as non-linearity, continuity, monotonicity, and computational efficiency for derivatives. This section introduces four of the most widely used activation functions, namely the rectified linear unit (*ReLU*), *Sigmoid* $\sigma(\cdot)$, *Tanh* and *Softmax*. The equations for the first three are shown in Eqn. 2.2, Eqn. 2.3 and Eqn. 2.4 respectively.

$$\text{ReLU}(x) = \max(0, x), \tag{2.2}$$

$$\sigma(x) = (1 + e^{-x})^{-1}, \tag{2.3}$$

$$\text{Tanh}(x) = (e^{x} - e^{-x})(e^{x} + e^{-x})^{-1}, \tag{2.4}$$

where $\max(\cdot, \cdot)$ chooses the maximum value between the two arguments. Plots for each of the three activation functions are shown in Fig. 2.2 together with the range of values each can take.



ReLU: [0, +∞) Sigmoid: (0, 1) Tanh: (-1, 1)

Fig. 2.2 Plots and ranges of activation functions: ReLU, Sigmoid and Tanh.

These activation functions are usually used as the final computation step of a hidden layer in MLP and are applied to each scalar element in the output vector. On the other hand, the Softmax activation function as shown in Eqn. 2.5 is usually used in the final output layer for classification tasks, or whenever categorical distributions are needed such as in the attention mechanism since it forms a probability mass function over the classes.

$$\text{Softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}}. \tag{2.5}$$

Equation 2.5 shows the i-th element in the output vector of dimension M, which is derived from the input $\mathbf{z}$ of the same dimension. Elements of $\mathbf{z}$ are often referred to as *logits*. Each element in $\text{Softmax}(\mathbf{z})$ is between 0 and 1, and the elements sum to 1.

2.1.3 Convolutional Neural Networks

The convolutional neural network (CNN) (Lecun et al., 1989) is one widely used variation of MLP which was first developed and applied to 2D image-related tasks in computer vision (Krizhevsky et al., 2012). CNNs are very successful when applied to data where the temporal and/or spatial correlations between input variables are crucial. The CNN is inspired by the biological process in which cortical neurons respond to stimuli only in a restricted region. As for MLPs, CNNs contain multiple hidden layers, and by grouping repeated computational patterns together, CNN can be divided into a series of convolutional blocks. A typical convolutional block consecutively performs convolution, pooling and non-linear activation, and may also include normalisation and FC layers. The final output of the block sequence will be arranged into a vector and passed through an FC output layer, as shown in Fig. 2.3.

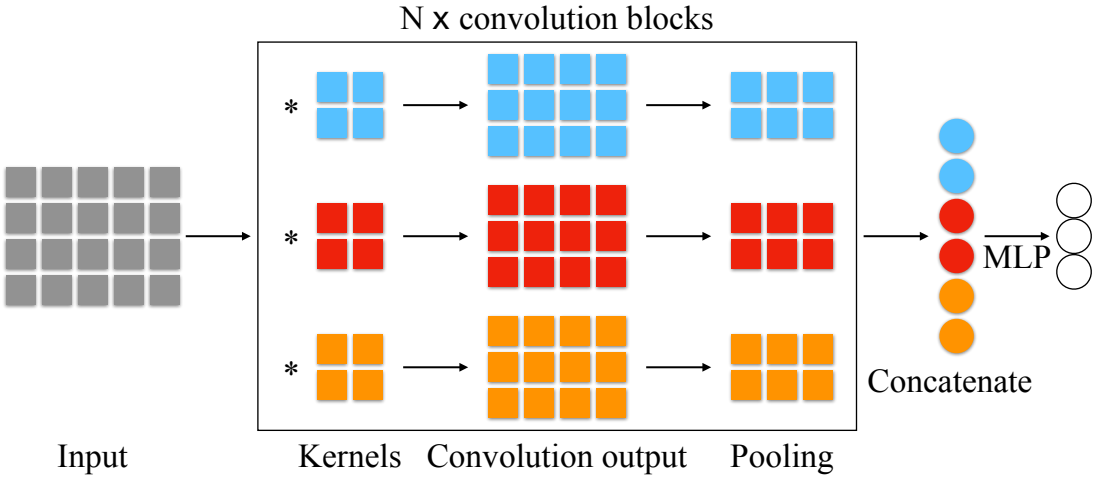


Fig. 2.3 The architecture of a CNN with N hidden convolutional blocks.

- The convolution layer is the core of a CNN whose parameters are in the form of trainable filters (i.e. kernels) usually having a small receptive field (e.g. 3×3). Given the input to each layer, each filter is convolved across the width and height of the input, computing the dot product between the entries of the filter and the input, which results in a 2-dimensional map of that filter. Through training, the network learns filters that activate when it detects a specific type of feature at some spatial positions in the input.
- A pooling layer is a form of non-linear down-sampling. Max pooling is the most widely used algorithm which partitions the input image into non-overlapping rectangles and for each rectangle, it outputs the element with the maximum value.

In CNNs, the convolution kernel is normally much smaller than the input features, which allows it to perform feature extraction of a high-dimensional input using a relatively small

number of parameters, which otherwise will result in extremely wide layers in MLP. The small convolution kernels also enable CNNs to capture local shift-invariant features, which is particularly beneficial for image data where a specific object should have similar representations no matter where it appears. In addition to computer vision, this characteristic of CNNs has also been applied to speech and language fields such as speech recognition (Sainath et al., 2013) and sentence classification (Kim, 2014).

2.1.4 Recurrent Neural Networks

The information flow in DNNs mentioned so far is uni-directional from the input to the output (i.e. feed-forward), whereas the recurrent neural network (RNN) which deals with sequential inputs, allows information feedback from the network output using a recurrent structure. A uni-directional Elman RNN is shown in Fig. 2.4.

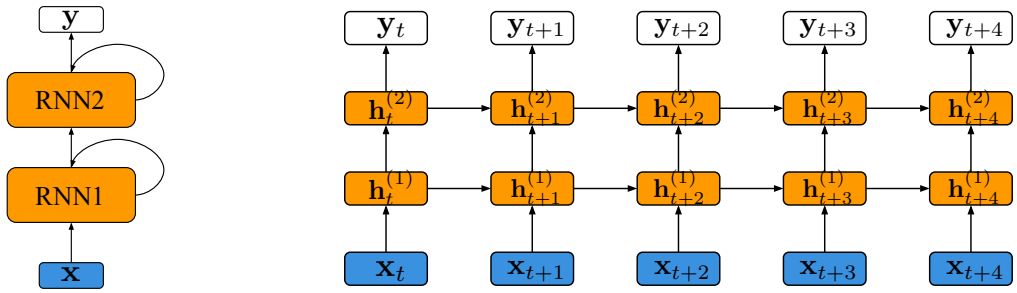


Fig. 2.4 The architecture of an RNN. Left: the folded RNN where the recurrence is shown by the self-loop in the graph. Right: the unfolded RNN showing information flow at each step.

At each position or time step in the sequence, the RNN layer takes in both the input vector, $\mathbf{x}_t$ and the RNN layer output from the previous time, $\mathbf{h}_{t-1}$, also called the *RNN hidden state*. The output of the RNN layer will again be used as the input to the next time step. By modifying Eqn. 2.1, the computation of each RNN layer can be written as Eqn. 2.6.

$$\mathbf{h}_t = f(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h), \tag{2.6}$$

where $\mathbf{W}_h$ and $\mathbf{U}_h$ are both parameter weight matrices of the RNN layer. In addition to the unidirectional RNN where hidden states are only carried from previous to future positions, bi-directional RNNs allow hidden states to be carried from future positions to the previous ones at the same time.

The RNN is trained using the back-propagation through time (BPTT) algorithm which will be covered later. The model above exhibits a *vanishing gradient* problem (Bengio et al., 1994) due to gradients of some activation functions such as the Sigmoid function[1] which

[1] Note that ReLU RNN does not have vanishing gradient problem

is often used for RNNs, being smaller than 1/4, such that long-term dependencies in the sequence are poorly captured. Therefore, the following variant of RNN based on the gating mechanism is introduced which yields better performance and is widely used nowadays.

2.1.4.1 Long Short-Term Memory

The long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) network introduces a specialised memory component, known as the *memory cell* which serves as a storage of long-term information. The graphical illustration of the LSTM computation at time t is shown in Fig. 2.5 below.

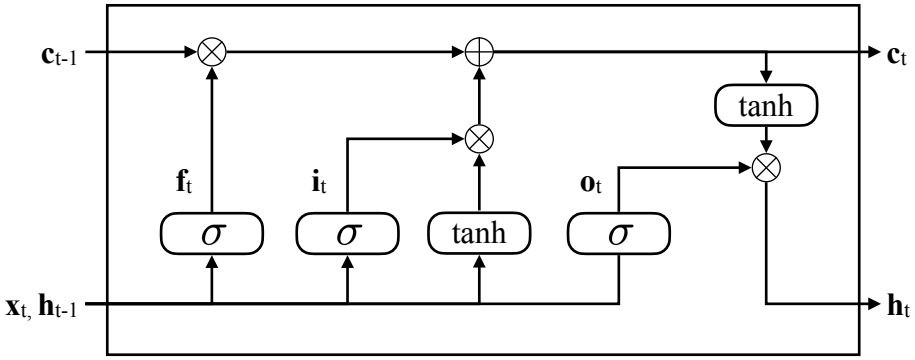


Fig. 2.5 Diagram of an LSTM cell. At the time step t, $\mathbf{x}_t$ is the input vector, $\mathbf{h}_t$ is the hidden state and $\mathbf{c}_t$ is the cell state.

The memory cell, usually represented by $\mathbf{c}$, will be continuously visited and updated using the gating mechanisms shown in Eqn. 2.7-2.9.

$$\mathbf{f}_t = \sigma(\mathbf{W}_f^f \mathbf{x}_t + \mathbf{W}_f^r \mathbf{h}_{t-1} + \mathbf{W}_f^m \mathbf{c}_{t-1} + \mathbf{b}_f), \tag{2.7}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i^f \mathbf{x}_t + \mathbf{W}_i^r \mathbf{h}_{t-1} + \mathbf{W}_i^m \mathbf{c}_{t-1} + \mathbf{b}_i), \tag{2.8}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o^f \mathbf{x}_t + \mathbf{W}_o^r \mathbf{h}_{t-1} + \mathbf{W}_o^m \mathbf{c}_t + \mathbf{b}_o), \tag{2.9}$$

where $\mathbf{f}_t$ is the forget gate, $\mathbf{i_t}$ is the input gate and $\mathbf{o}_t$ is the output gate. Then the update equations for the hidden states are shown in Eq. 2.10 and Eq. 2.11.

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot f(\mathbf{W}_c^f \mathbf{x}_t + \mathbf{W}_c^r \mathbf{h}_{t-1}), \tag{2.10}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot f(\mathbf{c}_t). \tag{2.11}$$

By inspecting the form of the memory cell which contains a gated connection to the memory cell of the last step, and another gated connection to the information of the current step, i.e. $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$, longer-term dependencies can be established via those gated connections.

2.1.5 Attention Mechanisms

The attention mechanism is one of the most powerful components in sequence modelling tasks which has significantly improved neural machine translation (Bahdanau et al., 2014) when first introduced, and has been widely applied to various DNN structures in more recent research (Wang et al., 2017, Zhang et al., 2019). The idea behind attention is to model the inter-dependency between the input and the output by dynamically weighting the importance of each part of the input using DNN-based scaling factors. While there are multiple variants of attention mechanisms, this section will focus on additive attention and dot-product attention. These methods are illustrated in 2.1.5.1 and two successful DNN models that are constructed based on attention mechanisms are introduced in Chapter 3.

2.1.5.1 Additive and dot-product attention

Attention combines the input vector sequence by a weighted average where attention weights are dynamically determined based on the input and/or output. Given a sequence of input vectors $\mathbf{x}_{1:T}$ to the attention layer, the output is formulated in Eqn. 2.12.

$$\mathbf{c} = \sum_{t=0}^{T} \alpha_t \mathbf{x}_t, \tag{2.12}$$

where $\alpha_{1:T}$ is a set of attention weights which are obtained from a score function. The calculation of the score function is similar to the process of querying a database, where a *query* vector $\mathbf{q}$ is used to query a set of *key* vectors $\mathbf{k}_t$ derived from the input sequence. Following this naming convention, the vectors to be summed up, $\mathbf{x}_t$, are often referred to as *value* vectors. Depending on how the score function is achieved, the attention mechanism can be divided into additive and dot-product types. For additive attention, the score is defined in Eqn. 2.13 and for dot-product attention, the score is defined in Eqn. 2.14.

$$s_t = \mathbf{v}^T \tanh(\mathbf{W}[\mathbf{q};\mathbf{k}_t]). \tag{2.13}$$

$$s_t = \mathbf{q}^T \mathbf{k}_t. \tag{2.14}$$

where $\mathbf{v}$ and $\mathbf{W}$ are both parameters to be trained, and $[\cdot;\cdot]$ denotes vector concatenation operation. The weights are thus determined by applying the Softmax activation function to the score vector $\mathbf{s} = [s_1, s_2, ..., s_T]$

$$\alpha = \text{Softmax}(\mathbf{s}). \tag{2.15}$$

When the query vector $\mathbf{q}$ is also derived from the same input sequence, the specific attention mechanism is called *self-attention*. Furthermore, when multiple sets of attention weights are obtained by deriving multiple sets of query-key pairs for the same input sequence, it is referred to as the *multi-head attention*.

The attention mechanism has gained prominence in sequence-to-sequence tasks as it overcomes the challenge of capturing a single hidden representation for increasingly longer sequences. In contrast to compressing the information from a sequence into a fixed-length vector, the attention mechanism allows the system to focus dynamically on different parts of the input sequence in the transformed space. This technique effectively generates the desired output in each position in the corresponding output sequence. The attention mechanism has proved beneficial for machine translation tasks where input-to-output alignment is complex, and the decoder needs to attend to various features linked to words in the input sequence at different output steps. The attention mechanism has also inspired the development of novel sequence models such as Transformers introduced later.

2.2 Optimisation

After introducing important neural network building blocks, this section will focus on the optimisation of DNNs. The network optimisation, or network training, is the task to find the set of model parameters, θ that minimises the expectation of the given loss function $\mathcal{L}$ across the true data distribution P_{data}.

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{\mathbf{x},\mathbf{y}\sim P_{\text{data}}}[\mathcal{L}(f(\mathbf{x};\theta),\mathbf{y})]. \tag{2.16}$$

Optimisation starts with designing a suitable DNN structure and choosing an appropriate loss function for a given task. Then, DNN parameters are initialised (see Section 2.2.4.1) to achieve a good starting point for training. After initialisation, network training then commences. Network training is an iterative procedure and the parameters are updated in each iteration which usually comprises three steps. First, the loss function $\mathcal{L}$ is designed and calculated for the intended task by performing the computation procedures of the DNN from input to output, denoted as the *forward pass*. Then, the gradients of the loss with respect to the model parameters are calculated from the final layer down to the first layer, denoted as the *backward pass*. Finally, the model parameters are updated according to their gradients using selected algorithms that work well with the task, such as stochastic gradient descent. Furthermore, for better convergence and better generalisation to unseen data, normalisation and regularisation techniques are usually applied.

2.2.1 Training Criteria and Cross-entropy Loss

A suitable training criterion should be selected that directly reflects the task the DNN is expected to achieve, which is also one of the most important parts of deep learning. Most DNNs adopt the maximum likelihood (ML) criterion, which estimates values of model parameters that minimise the difference between the empirical data distribution defined by the training data, and the distribution predicted by the model. However, as the true data distribution is usually intractable, it is necessary to approximate the expected loss using data samples, i.e. the training data. If the training data is $\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, the model predicts the distribution of $\mathbf{y}_i$ given $\mathbf{x}_i$. Assume the data distribution is P_{data} and the model distribution is $P(\mathbf{y}|\mathbf{x}, \theta)$ where θ represents model parameters, then the ML training criterion to obtain the best parameter θ^* is shown in Eqn. 2.17.

$$\theta^* = \arg\max_{\theta} \prod_{i=1}^{N} P(\mathbf{y}_i|\mathbf{x}_i; \theta) = \arg\max_{\theta} \sum_{i=1}^{N} \log P(\mathbf{y}_i|\mathbf{x}_i; \theta), \tag{2.17}$$

where the product rule for probabilities applies here to compute the joint probability, assuming data samples are independent, which can be converted into the sum of log probabilities. The closed form of this loss depends on the form of the model distribution. Since most systems considered in this report have Softmax output layers which produce a probability over all possible values of scalar targets y_i, the ML criterion can be written in Eqn. 2.18.

$$\theta^* = \arg\min_{\theta} \mathcal{L} = \arg\min_{\theta} -\sum_{i=1}^{N} \log P(y_i|\mathbf{x}_i; \theta), \tag{2.18}$$

where the arg max is replaced by arg min with an additional minus sign inserted to make it a loss function that is to be minimised. This loss is often referred to as the cross-entropy loss which is widely used in the training due to its simplicity and a close match to most task objectives.

2.2.2 Error Back-propagation

Error back-propagation is the algorithm used to calculate the gradient of the loss function w.r.t model parameters using the chain rule of partial differentiation. Taking the network in Fig. 2.1 as an example, given loss function $\mathcal{L}$, the gradient w.r.t. $\mathbf{W}^{(3)}$ can be calculated in Eqn. 2.19.

$$\frac{\partial \mathcal{L}}{\partial W_{i,j}^{(3)}} = \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial W_{i,j}^{(3)}} = \frac{\partial \mathcal{L}}{\partial y_i} \cdot h_j^{(2)}. \tag{2.19}$$

If the elements in $\mathbf{y}$ are the logits before passing through the Softmax layer and the loss function is cross-entropy, the gradient of $\mathbf{y}$ has a simple form as shown in Eqn. 2.20.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \mathbf{p} - \mathbf{y}, \tag{2.20}$$

where $\mathbf{p}$ is the vector representing the predicted label distribution. To calculate gradients for the previous layer, the chain rule is applied which requires the gradient w.r.t. $\mathbf{h}^{(2)}$ to be first calculated.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(2)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{h}^{(2)}} \cdot \frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W}^{(2)}}, \tag{2.21}$$

This procedure can be repeatedly applied to the previous layers until all layers have been traversed in the DNN. As the chain rule expansion is similar to propagating the gradients along the network, it is termed gradient or *error back-propagation*. Practically, for RNNs, the same algorithm can be applied by first unfolding the RNN by a fixed number, T, steps, known as truncated BPTT. Then the gradient of the loss at a certain step is computed and back-propagated to the preceding steps using the chain rule, which is known as the *BPTT* algorithm. Specifically, for the loss computed at the final step T, the gradient w.r.t. the weight $\mathbf{W}_h$ in Eqn. 2.6 can be expanded as shown in Eqn. 2.22.

$$\begin{aligned} \frac{\partial \mathcal{L}_T}{\partial \mathbf{W}_h} &= \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{W}_h} + \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_{T-1}} \cdot \frac{\partial \mathbf{h}_{T-1}}{\partial \mathbf{W}_h} + \ldots \\ &= \sum_{t=1}^{T} \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \cdot \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \cdot \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h}. \end{aligned} \tag{2.22}$$

According to Eqn. 2.6, intermediate derivatives $\partial \mathbf{h}_t / \partial \mathbf{h}_{t-1}$ can be further expanded to the previous step as shown in Eqn. 2.23.

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \frac{\partial f(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h)}{\partial \mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h} \cdot \frac{\partial \mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h \mathbf{h}_{t-1} + \mathbf{b}_h}{\partial \mathbf{h}_{t-1}}. \tag{2.23}$$

When $f(\cdot)$ is the sigmoid function, $df(x)/dx = f(x)(1 - f(x))$. Because $f(x) \in (0, 1)$, the maximum value of this derivative is 0.25. Therefore, when it is repeatedly multiplied in the above expansion, the value of the derivatives will be scaled by 0.25 for every BPTT step, resulting in the gradient gradually vanishing. This is the *vanishing gradient* problem which can be mitigated by unfolding fewer steps or adding gated connections such as in the LSTM.

2.2.3 Parameter update

2.2.3.1 Stochastic gradient descent

After obtaining gradients associated with weights and biases, the final step of the training iteration is to update the weight values. One simple but powerful update algorithm is the gradient descent (GD) which takes the form shown in Eqn. 2.24.

$$\theta \leftarrow \theta - \eta \frac{1}{M} \sum_{i=1}^{M} \nabla_\theta \mathcal{L}(f(\mathbf{x}_i;\theta), \mathbf{y}_i), \tag{2.24}$$

where η is the *learning rate* and M is the number of data points in the training set. Since the gradient term is an expectation, the expectation can be approximately estimated using a small set of samples. Therefore, when the training set is large so that a single update step can not include all data samples, the set is usually split into small subgroups known as *minibatches*. The gradient term in Eqn. 2.24 is estimated for each minibatch as shown in Eqn. 2.25

$$\theta \leftarrow \theta - \eta \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta \mathcal{L}(f(\mathbf{x}_i;\theta), \mathbf{y}_i), \tag{2.25}$$

where m' is the size of the minibatch. For each update, a minibatch is sampled from the training set to estimate the gradient at current θ. This procedure is referred to as the *stochastic gradient descent* (SGD). SGD significantly reduces the computation required for each gradient update, yet offers a much noisier gradient estimation which results in a lower convergence speed in terms of the number of updates.

For faster convergence, *momentum* is used in SGD. It introduces an additional variable $\mathbf{v}$ which describes the direction and speed, i.e. velocity, at which the parameters move in the space. The use of momentum in SGD mimics the motion of a unit mass and the negative gradient term in Eqn. 2.25 is the force acting on the mass. Formally, the velocity update and the parameter update are shown in Eqn. 2.26 and Eqn. 2.27 respectively.

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta \mathcal{L}(f(\mathbf{x}_i;\theta), \mathbf{y}_i), \tag{2.26}$$

$$\theta \leftarrow \theta + \mathbf{v}, \tag{2.27}$$

With momentum, the size of each step depends not only on the norm of gradient and the learning rate but also on how well the past gradient sequence is “aligned". If the sequence of gradients is well-aligned which resembles a particle descending a steep part of a surface,

the velocity as well as the updated step size will continuously become larger. Therefore, this helps to improve the speed of convergence.

2.2.3.2 Adaptive learning rates

Since the learning rate is another important factor that influences convergence, methods have been developed to have *adaptive* learning rates to improve training speed. The fundamental assumption behind these methods is that the loss is more sensitive to some model parameters and less to others. Then, it makes sense to use a separate learning rate for each parameter and automatically adapt these learning rates without introducing too many hyper-parameters. This section will focus on the *Adam* (adaptive momentum) algorithm (Kingma and Ba, 2015), a commonly used method that combines adaptive learning rate and momentum. The Adam algorithm is shown in Algorithm 1, where $\odot$ represents the element-wise product of tensors and δ is a small quantity to ensure numerical stability.

Algorithm 1 The Adam algorithm

Initialise $\rho_1, \rho_2 \in [0,1)$, $\eta \sim 10^{-3}$, $\delta \sim 10^{-8}$, $t = 0$.
Initialise θ
Initialise 1st and 2nd moment variables, $\mathbf{s} = \mathbf{0}$, $\mathbf{r} = \mathbf{0}$
while *stopping criterion not met* **do**
  Sample a minibatch from training set, $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}'_m\}$ with corresponding targets
  Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m'} \sum_{i=1}^{m'} \nabla_\theta \mathcal{L}(f(\mathbf{x}_i; \theta), \mathbf{y}_i)$
  $t \leftarrow t + 1$
  Update first moment estimate: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1)\mathbf{g}$
  Update second moment estimate: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$
  Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$
  Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$
  Compute and apply update: $\theta \leftarrow \theta - \eta \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$
end

Even though various adaptive learning rate and momentum methods have claimed superior performance, no consensus has been reached on which learning algorithm is the best so far, because each aforementioned algorithm (SGD, SGD+momentum and Adam) has its own strengths and weaknesses, and is suitable for some specific tasks.

2.2.4 Further Optimisation Strategies

In addition to a suitable update algorithm, the success of DNNs is also attributed to a further set of optimisation strategies. In this section, parameter initialisation, various normalisation

techniques and learning rate schedules will be introduced, as three influential techniques in training.

2.2.4.1 Initalisation

In deep learning, the optimisation of neural networks is an iterative process that necessitates a suitable starting point. The performance of the model can be heavily reliant on the initial weight values, which can determine the speed and extent to which training will converge, as well as the generalisation ability of the model. Many initialisation methods are random and heuristic, with weights being sampled from Gaussian or uniform distributions and biases being initialised to constant values. However, the range of the initialised weights plays a critical role. While larger weights aid in distinguishing each unit, thereby reducing the number of redundant units, they may lead to exploding or vanishing gradients based on the activation functions. One common approach for initialisation is the LeCun initialisation which is also used in most of the model training in this thesis.

LeCun initialisation (LeCun et al., 2012) draws initial parameters from a Gaussian distribution, with variance determined by the dimension of the layer input, see Eqn. 2.28.

$$w \sim \mathcal{N}(0, \frac{1}{n}) \tag{2.28}$$

where n is the dimension of the layer input and w is a specific model parameter. As DNN forward pass performs matrix multiplication, this technique constrains the expected variance of the output to be one, which leads to fast convergence.

2.2.4.2 Normalisation

There are three commonly applied normalisation tricks in DNN training, namely feature normalisation, batch normalisation and layer normalisation. *Feature normalisation* calculates the first and second-order statistics for all training data and normalises by subtracting the mean and then dividing the standard deviation before training the DNN. The normalised data is used as input to the DNN for training.

Batch normalisation (Ioffe and Szegedy, 2015), motivated by the difficulty of training very deep DNNs such as CNNs, provides a universal way of re-parameterisation. For very deep networks, updates to the parameters in the previous layers will change the input distribution and hence slow down training by requiring a lower learning rate, which is referred to as the *internal covariate shift*. Batch normalisation achieves faster and better training. Let $\{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_M\}$ be the output of the layer to be normalised, the algorithm first calculates the

mean μ and standard deviation σ following Eqn. 2.29 and Eqn. 2.30 respectively.

$$\mu = \frac{1}{M}\sum_{i=1}^{M}\mathbf{h}_i, \tag{2.29}$$

$$\sigma = \sqrt{\delta + \frac{1}{M}\sum_{i=1}^{M}(\mathbf{h}_i - \mu)^2}, \tag{2.30}$$

where δ is a small positive value imposed to avoid encountering the undefined gradient of $\sqrt{z}$. During the test, the mean and the standard deviation are replaced by the running average collected during training. The most important property of batch normalisation is that gradients can be back-propagated through the operations above. This means the gradient will never act simply to increase the mean or standard deviation.

Batch normalisation has been successful in improving the performance of deep neural networks, but its application to RNNs is challenging due to the impracticality of batch normalisation for each time step. A potential solution is *layer normalisation* (Ba et al., 2016), which normalises across all feature dimensions for each feature at each layer instead of normalising across each feature dimension in the batch of input features. This approach is independent of other samples and can be applied to train RNNs with long sequences and small minibatches. Studies have shown that layer normalisation can speed up the training of RNNs and it is also effective in pure-attention-based sequence-to-sequence models such as Transformer (Vaswani et al., 2017).

2.2.4.3 Learning rate schedule

Despite the adaptive learning rate approaches, the hyper-parameter η still dominants the size of an update step. Empirically, using a scheduled learning rate often leads to a better convergence point (Bergstra and Bengio, 2012). With the schedule, the learning rate η changes with the number of updates or according to a certain criterion. The schedule is highly dependent on the choice of the DNN structure and the optimisation algorithm. For example, one commonly adopted approach to train LSTMs using SGD is to reduce the learning rate by half whenever the performance on a held-out validation set degrades after a certain epoch. Another widely adopted approach is to gradually increase the learning rate against the number of updates from the start until a maximum is reached (referred to as *warm-up*), and then gradually decrease it until the designated number of updates is reached (Vaswani et al., 2017).

2.3 Regularisation

With the advancement in computing resources over the years, DNN models have been developed to have many millions of parameters. This put forward a problem with the limited amount of data where the model performs well on the training set but poorly on the test set, known as *over-fitting*. *Regularisation* collectively refers to a set of methods trying to address these over-fitting issue. This section introduces techniques that help address overfitting.

2.3.1 Early Stopping

The practice of early stopping is a commonly used strategy in deep learning, which involves stopping the training process before complete convergence based on a specified criterion, often the validation performance. This involves reserving a separate subset of the training data for validation purposes, whereby the model is periodically evaluated on this set after a predetermined number of update steps. If the validation performance continues to degrade while the training loss still decreases, training can be terminated before reaching convergence to avoid overfitting. The learning rate may also be adjusted based on the validation performance to improve the training process.

2.3.2 Parameter Sharing

In order to address overfitting, parameter sharing has proven to be an effective method as it reduces the number of parameters in a DNN. This approach is commonly used in CNNs and RNNs, where model parameters are shared across different parts of the network. Specifically, in CNNs, parameters are shared for each kernel that convolves across the entire input, while in RNNs, parameters are tied across each time step when viewed from the perspective of the network unfolding.

2.3.3 L2 Parameter Regularisation

L2 parameter regularisation, or *weight decay* is another commonly-used approach in gradient-based training. It encourages the weights to be closer to the origin to avoid a sharp mapping in a specific layer by adding a regularisation term to the loss function, as shown in Eqn. 2.31.

$$\tilde{\mathcal{L}}(\mathbf{x},\mathbf{y};\omega) = \mathcal{L}(\mathbf{x},\mathbf{y};\omega) + \frac{\alpha}{2}\omega^T\omega, \tag{2.31}$$

where ω particularly refers to the vectorised version of parameters in the weight matrix. The derivative w.r.t. ω is shown in Eqn. 2.32.

$$\nabla_\omega \tilde{\mathcal{L}}(\omega; \mathbf{x}, \mathbf{y}) = \alpha\omega + \nabla_\omega \mathcal{L}(\omega; \mathbf{x}, \mathbf{y}). \tag{2.32}$$

For a single gradient update step, the update is performed as shown in Eqn. 2.33.

$$\omega \leftarrow (1 - \eta\alpha)\omega - \eta\nabla_\omega \mathcal{L}(\omega; \mathbf{x}, \mathbf{y}). \tag{2.33}$$

The introduction of the weight decay term modifies the learning to multiplicatively shrink the weight vector by a constant factor which is a hyper-parameter that needs to be determined.

2.3.4 Dropout

Dropout is an effective and computationally efficient method of regularisation. It can be considered a practical approximation of bootstrap aggregating (bagging) for large DNN ensembles. Bagging is a technique aimed at reducing generalisation error by training a set of models and using them to vote on the test sample, resulting in ensemble averaging. The dropout method achieves ensemble averaging by randomly eliminating non-output units from an underlying base network. A common way to implement dropout is to multiply a set of units for a specific layer in the base network by zero.

To train with dropout in a minibatch-based learning algorithm, random binary masks are generated for each layer except the output one in the DNN whenever a sample is included in the minibatch. These masks will be applied to the outputs of corresponding layers when computing the loss from the given input. The probability of an element in the mask being set to one is a hyper-parameter that can be adjusted.

It is infeasible to calculate the average of all models generated by random masks to mimic the accumulation of votes from ensemble models. Instead, scaling the weight by the dropout probability and evaluating the underlying base model has been shown to be a useful approximation. Additionally, dropout can be seen as introducing random masking noise to hidden units. The resulting zero masks eliminate the information captured by certain nodes, requiring predictions to be based on other information. Therefore, dropout is a technique that prevents the model from relying solely on a few significant features, thus serving as a regularisation method for the network.

2.3.5 Data Augmentation

In order to improve a model's ability to generalise, it is important that it can handle various types of noise present in the data (Sietsma and Dow, 1991). While increasing the amount of training data sampled from the real data distribution can improve a model's noise robustness, this may not always be feasible due to limited data resources. Data augmentation is a widely used approach to increase the amount of training data available by generating new data from the existing data. For instance, images can be transformed in various ways such as cropping, scaling, translating, and rotating (Krizhevsky et al., 2012), while speech data can be augmented using techniques such as vocal tract length perturbation or speed perturbation (Jaitly and Hinton, 2013, Ko et al., 2015). Data augmentation enables the space of training data to be enriched and helps the model to become invariant to the applied transformations and more robust to noisy data. In the context of speech recognition, SpecAugment (Park et al., 2019) is a commonly used method to augment input log-mel spectrograms through multiple instances of time warping, frequency masking and time masking. By introducing randomly corrupted input, this method prevents the model from overfitting specific features and enhances its ability to generalise to different acoustic conditions.

2.3.6 Label Smoothing

The output can be also injected with noise to improve its robustness, similar to data augmentation for the input, which serves as another effective regularisation technique for training neural networks with CE loss (Szegedy et al., 2016). After setting a smoothing hyper-parameter ξ, the 1s in the target are replaced with $1-\xi$ and the 0s in the target are replaced with $\xi/(|\mathbf{y}|-1)$. Label smoothing injects noise into the output labels by smoothing hard targets to soft ones. This approach acts as a regulariser that prevents the model from overfitting to hard output distributions with Softmax, which may have larger weights. Additionally, label smoothing can be viewed as a way to simulate the situation where training labels are not entirely accurate.

2.3.7 Weight Noise

In addition to noises added to the input and the target, noise can also be added to model parameters that results in a minimal change in the final output to regularise the network. Gaussian noise with a fixed variance and a zero mean is often added to network weights during training to achieve this purpose, leading to improved performance in RNNs (Jim et al., 1996, Graves, 2013). During each minibatch, the gradient is computed based on model

parameters with the added Gaussian noise, but the update to the parameters via gradient descent is applied to the original parameters without the noise. Weight noise has the effect of "simplifying" neural networks, as it reduces the precision required to describe the weights. Simpler networks are preferred since they tend to generalise better. In addition, more complex approaches such as adaptive weight noise have been proposed (Graves, 2011). Furthermore, weight quantisation (Woodland, 1989, Hubara et al., 2017) can also be seen as a form of weight noise that enhances the generalisation of neural networks.

2.4 Transformer and Foundation Models

2.4.1 Transformer Model

Transformer (Vaswani et al., 2017) models have emerged as a powerful and flexible approach for sequence-to-sequence problems, achieving state-of-the-art results in a wide range of applications such as machine translation, language modelling, and end-to-end automatic speech recognition (ASR). Unlike RNNs which rely on the recurrent connection to model the inter-dependency across positions in the sequence, the Transformer employs self-attention mechanisms which process each position in the sequence in a feed-forward manner, which achieves a higher level of parallelisation that can take further advantage of the Graphics Processing Unit (GPU) capabilities. Moreover, attention removes the vanishing gradient problem as it provides a shortcut to faraway inputs. Therefore, the Transformer particularly benefits the processing of long sequences, such as speech sequences which often contain over a thousand input frames.

The transformer is an attention-based encoder-decoder (AED) model which comprises an encoder and a decoder that are connected by attention mechanisms. The encoder transforms input features into a sequence of high-level vector representations, and the decoder generates model predictions given the encoder output and previous predictions. In the Transformer, both the encoder and decoder contain repeated blocks that perform the same set of computations and are referred to as Transformer encoder/decoder blocks. The computation in each Transformer block only contains the attention mechanism and feedforward neural networks.

The core of the Transformer is the attention mechanism, which is scaled dot-product attention. Denoting the query matrix as $\mathbf{Q} = [\mathbf{q}_1, ..., \mathbf{q}_N]^T$, the key matrix as $\mathbf{K} = [\mathbf{k}_1, ..., \mathbf{k}_N]^T$, and the value matrix as $\mathbf{Q} = [\mathbf{v}_1, ..., \mathbf{v}_N]^T$, scaled dot-product attention is computed as follows:

$$\mathbf{A} = \text{Attention}(\mathbf{Q}, \mathbf{K}) = \text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}). \tag{2.34}$$

$$\mathbf{E} = \mathbf{AV}, \tag{2.35}$$

where d_k is the dimension of each key vector, and $\mathbf{E} = [\mathbf{e}_i, ..., \mathbf{e}_N]^T$ is the combined value matrix using attention weights. The operation inside the Softmax on the numerator is the dot-product between each pair of query and key vectors, and the result is a measure of the distance between the two vectors. The product is scaled by $\sqrt{d_k}$ to ensure the output vector has the same variance as the input. By assuming that each dimension of the query and key vectors is an independent random variable with zero mean and unit variance, the value of the dot product has zero means and variance d_k. Dividing the product by $\sqrt{d_k}$ reduces the variance to 1 which is consistent with the input. When the query, key and values are all computed from the same input $\mathbf{X}$, this attention becomes the self-attention as shown below:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \qquad \mathbf{Q} \in \mathbb{R}^{N \times d_q} \tag{2.36}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K, \qquad \mathbf{K} \in \mathbb{R}^{N \times d_k} \tag{2.37}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_V, \qquad \mathbf{V} \in \mathbb{R}^{N \times d_v} \tag{2.38}$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$ and $\mathbf{W}_V$ are parameter matrices, and $d_q = d_k$ to enable the dot-product to be performed. In general, keys and values usually come from the same source and queries may come from another source (referred to as *cross-attention* in this thesis).

Moreover, the Transformer uses multi-head attention (MHA) which allows the model to jointly attend to information from different parts of the value sequence. Assuming there are H attention heads, one specific set of queries, keys and values, $\mathbf{Q}_i$, $\mathbf{K}_i$ and $\mathbf{V}_i$ is derived from the same input source for each attention head. The H output matrices $\mathbf{E}_1, ..., \mathbf{E}_H$ are concatenated along the d_v dimension to form a single vector as the output of attention.

$$\mathbf{E}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i)\mathbf{V}_i \tag{2.39}$$

$$\mathbf{E} = \text{Concat}(\mathbf{E}_1, ..., \mathbf{E}_H^T)\mathbf{W} \tag{2.40}$$

where $\mathbf{W}$ is a model parameter matrix to mix and transform the information captured in each head. The detailed Transformer structure is shown in Fig. 2.6. MHA is applied in both the encoder and decoder blocks. The MHA module in the encoder and the masked MHA module in the decoder perform self-attention, while the MHA module in the decoder performs cross-attention which derives the query from the decoder hidden state $\mathbf{d}_{1:N}$ and the key and value from the encoder output $\mathbf{h}_{1:T}$. Cross-attention is analogous to the attention used in the RNN AED structure, but it is performed in every decoder block rather than just once, and hence the alignment is implicitly learnt and hard to interpret.

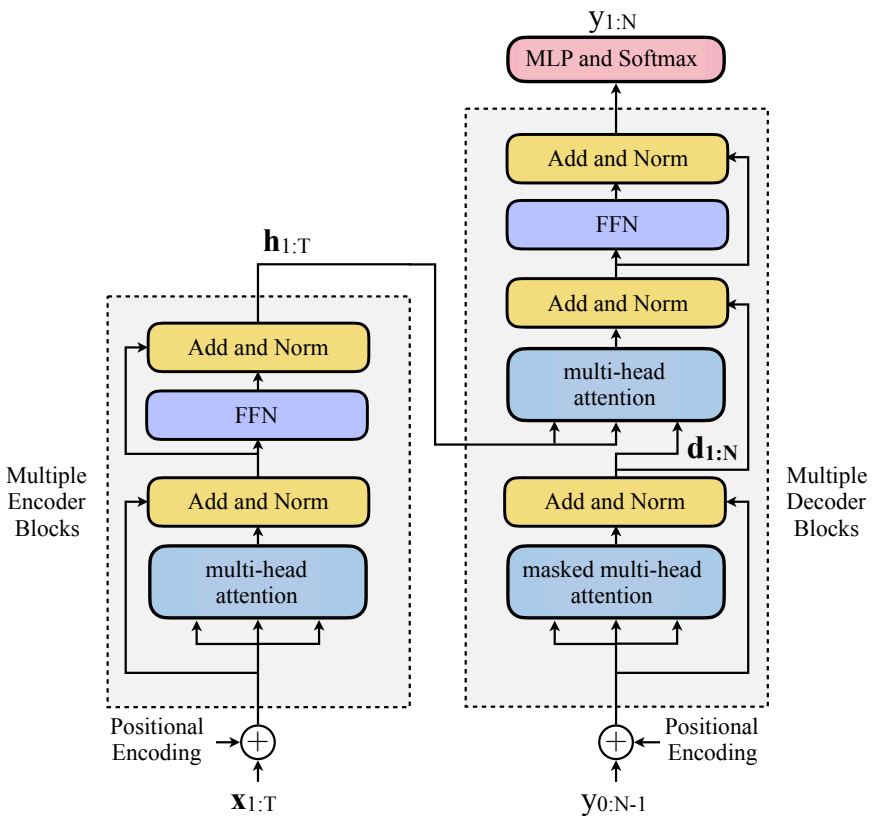


Fig. 2.6 Transformer model structure (Vaswani et al., 2017). Add and norm denotes residual connections and layer normalisation. FFN represents the position-wise feed-forward network.

In addition to attention, each block also contains a position-wise feed-forward layer which applies the same MLP to each position in the sequence. There is also the add and norm module which provides residual connections followed by layer normalisation, which prevents the gradient from becoming too noisy when backpropagated through a very deep network (He et al., 2015).

As with RNN AED, the decoder makes predictions of the next token y_i given the previous tokens $y_{0:i-1}$ where y_0 is the start-of-sequence token. To ensure this causality, triangular masks are applied to each attention head, as shown in Eqn. 2.41.

$$\mathbf{A} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{M}) = \text{Softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}} \odot \mathbf{M}). \tag{2.41}$$

where $\mathbf{M} \in \mathbb{R}^{N \times N}$ represents a binary triangular mask and $\odot$ represents the element-wise product between two matrices. This is also illustrated in Fig. 2.41. Note that the i-th row of $\mathbf{A}$ is an attention weight that is used to combine value vectors for the i-th query. After masking, the attention weights for positions i to N become zero, and hence no future information is

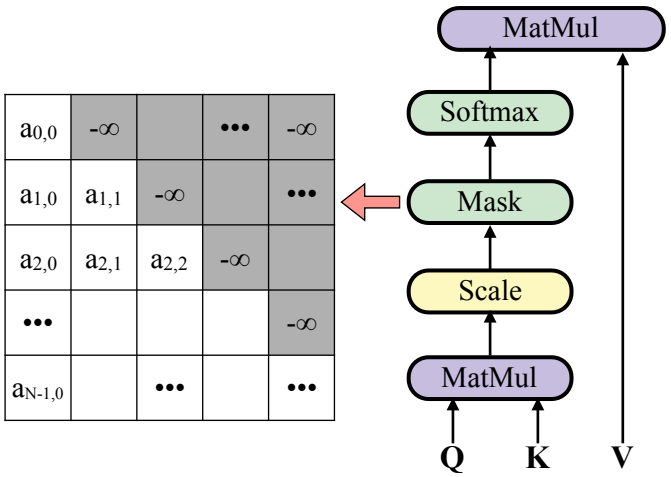


Fig. 2.7 Masked MHA computation flow and the score matrix after applying the diagonal mask where the grey grids are replaced with $-\infty$.

used when obtaining the output at the current position. This ensures the causal information flow and hence allows the model to be trained by predicting the next target with teacher forcing.

As Transformer blocks treat each position identically and process them in parallel, the positional information of the input sequence is not explicitly maintained. Therefore, additional positional embeddings are added to the input before sending it to the Transformer blocks. Positional encoding describes the position of a vector in a sequence so that each position is assigned a unique representation. Positional encoding can either be fixed or learned and for the fixed form proposed by Vaswani et al. (2017), it employs sinusoidal functions where each element in the embedding vector at position i, indexed by j is given by

$$\mathrm{PE}(i,j) = \begin{cases} \sin\left(i/10000^{j/D}\right) & j \text{ is even}, \\ \cos\left(i/10000^{(j-1)/D}\right) & j \text{ is odd}. \end{cases} \tag{2.42}$$

The positional embedding is directly added to the input vector at position i. The sinusoidal version of positional embedding enables the model to extrapolate to sequence lengths longer than the ones seen in the training data.

2.4.2 Self-supervised Pre-training with Transformer

Acquiring labelled data can be expensive and time-consuming which often requires domain expertise, whereas there exists a vast amount of unlabelled speech and text data. Recent

research has been trying to make use of unlabelled data to obtain a foundation model that provides a better initialisation point or powerful neural representations of the data.

A foundation model is a DNN model, often a Transformer, that is trained on a vast quantity of unlabelled data at scale (Devlin et al., 2019). It is usually trained with self-supervised representation learning to learn useful representations. Self-supervised learning (SSL) is a type of unsupervised learning which uses information extracted from the input data itself as the label to learn representations useful for downstream tasks. With SSL representation learning or pre-training, the model can automatically discover the representations needed for feature detection or classification from raw data. The pre-trained models can be applied to a wide range of related downstream tasks via transfer learning, which is often referred to as fine-tuning. With fine-tuning, pre-trained LMs can be applied for language generation, question answering and machine reading comprehension, etc, while pre-trained speech models can be used for automatic ASR, speaker tasks and spoken language understanding (SLU).

This section introduces foundation models that are either trained on text or trained on speech data. Their applications in ASR and SLU will be introduced in Chapter 3.

2.4.2.1 Foundation Language Models

Language model pre-training by leveraging abundant unlabelled text has been shown to be effective in improving many natural language processing tasks (Devlin et al., 2019, Radford et al., 2018, 2019, Raffel et al., 2019, Yang et al., 2019). Such models usually involve training a Transformer structure where the input and output come from the same text source. There are three major types of foundation models depending on the pre-training task design, including bi-directional encoder representations from Transformer (BERT) (Devlin et al., 2019), the generative pre-trained Transformer (GPT) (Radford et al., 2019) as well as text-to-text transfer Transformer (T5) (Raffel et al., 2019).

There is a group of foundation models (Devlin et al., 2019, Liu et al., 2019a) that adopt only the Transformer encoder to encode text tokens into representations, and BERT is one typical example. The BERT model is shown in the top part of Fig. 2.8. BERT pre-trains a bi-directional Transformer encoder using masked token prediction and next-sentence prediction tasks. As shown in Fig. 2.8, a certain portion of the input text is masked, and the BERT model is trained by predicting the tokens that correspond to those masks (with CE loss). This is the masked token prediction task. Meanwhile, BERT takes pairs of sentences as input, and the output corresponds to the “[CLS]" token at the start of the sequence to determine whether the second sentence follows the first sentence, by projecting through a binary classification layer. Through this training scheme, correlations within each sentence can be captured by the

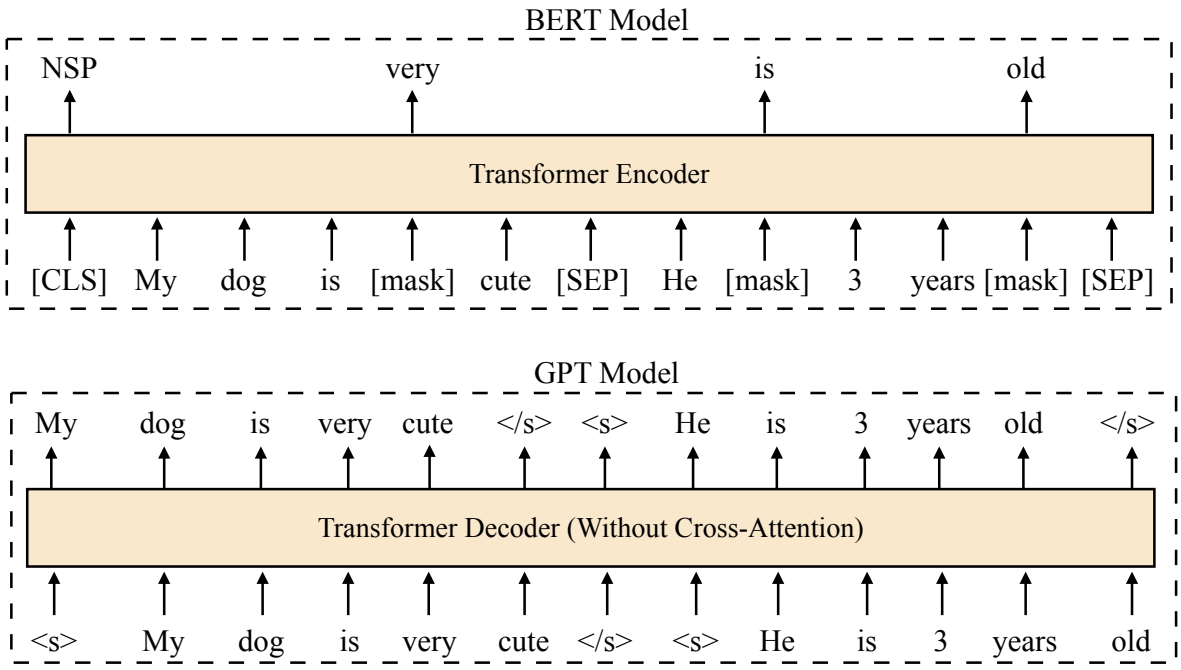


Fig. 2.8 BERT (top) and GPT (bottom) foundation models with their pre-training tasks.

mask prediction while sentence-level information is extracted by the output representation for the "[CLS]" token. In addition, Liu et al. (2019a) proposed a series of robust training approaches to improve the pre-training of BERT, and the resulting foundation model, termed RoBERTa, is used in Chapter 6.

BERT and all bi-directional models using a Transformer encoder are usually suitable for sequence tagging (Chen et al., 2019a), span prediction (Heck et al., 2020) and classification tasks (Wang et al., 2018). However, bi-directional models are not capable of performing sequence generation tasks as they are trained with future content information. Therefore, for sequence generation tasks, GPT models were proposed.

GPT (Radford et al., 2018), GPT2 (Radford et al., 2019) and GPT3 (Brown et al., 2020) were introduced as a series following the same training scheme but with increasing model size and data scale. The training task of GPT models is shown in the bottom part of Fig. 2.8. GPT models use the Transformer decoder but remove the cross-attention, which is equivalent to using the Transformer encoder with a causal mask. GPT is trained with next-token prediction using the CE loss, which is the standard language model training approach as explained in Sec. 3.4.1. GPT models can directly be used as a text generator, as well as being fine-tuned for downstream tasks such as question-answering (Rajpurkar et al., 2016) or response generation in spoken dialogue systems (Hosseini-Asl et al., 2020). Note that as a causal language model, GPT models can also be used to re-rank ASR hypotheses (Zheng

et al., 2021) as discussed in Sec. 4.8. As with BERT, GPT models are also able to perform sequence classification tasks. More recently, the text generation ability of GPT models has been further boosted by using Reinforcement Learning from Human Feedback, leading to state-of-the-art chatbots such as instructGPT (Ouyang et al., 2022) and ChatGPT.

The last type of language foundation model introduced in this section is the T5 model (Raffel et al., 2019) which uses the entire Transformer structure. The main idea of T5 is to transform all existing text-processing tasks into a "text-to-text" task, where there is a text-based prompt (e.g. the question) and a text-based response (e.g. the answer). The basic pre-training scheme for T5 is shown in Fig. 2.9.

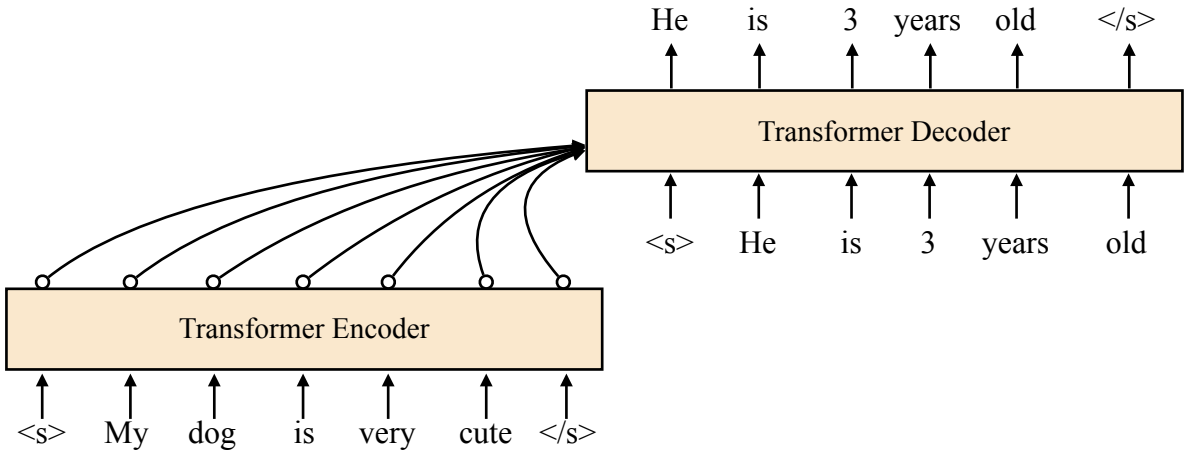


Fig. 2.9 T5 model pre-training scheme

The T5 model can be considered as the GPT model augmented with further contextual dependencies from a bi-directional Transformer encoder. During fine-tuning, T5 transforms many tasks, such as sequence classification and machine translation, into question-answering form, e.g. "translate the following sentence into German: this is good" with the target sequence "das ist gut". This idea is also widely adopted in spoken language understanding tasks which will be further elaborated in Sec. 3.5.

2.4.2.2 Foundation Speech Models

As for language processing, foundation models have also been developed for speech processing which transforms speech waveforms into high-level vector representations via self-supervised representation learning (van den Oord et al., 2018, Baevski et al., 2020). The key challenge for speech foundation model pre-training is that the speech signal is high-dimensional. One utterance may contain millions of data points, in contrast to the number of word tokens, which makes it computationally infeasible to mask and reconstruct the

original signal. This section mainly focuses on wav2vec 2.0 (Baevski et al., 2020), a specific approach that converts speech signals into word-like token-level representations via vector quantisation.

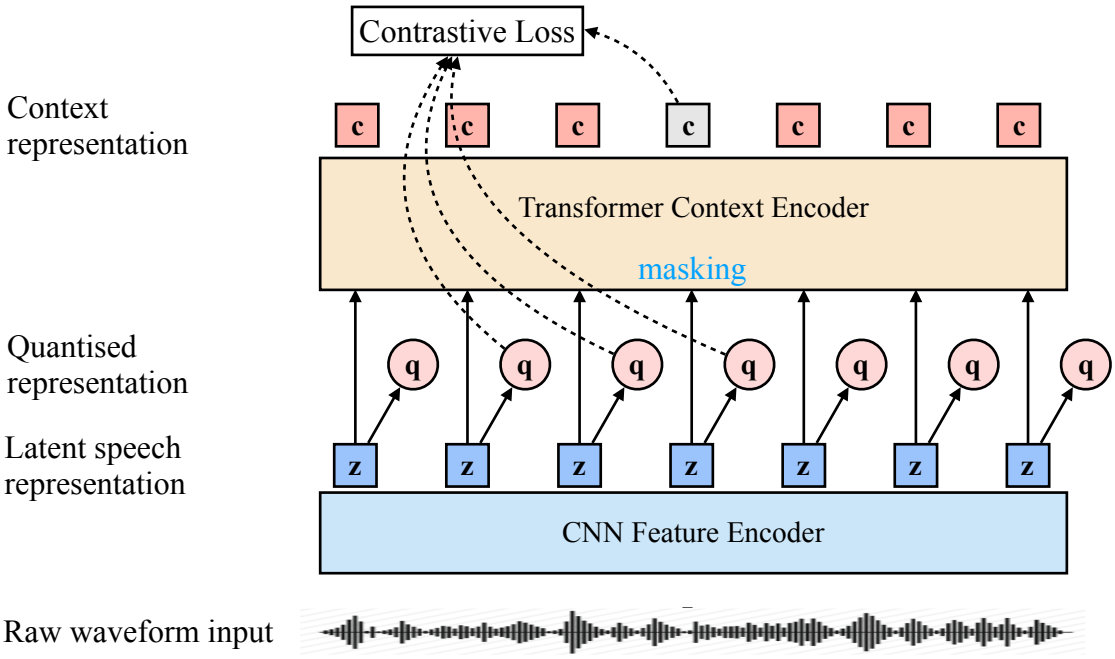


Fig. 2.10 Model structure and the pre-training task for wav2vec 2.0

The wav2vec 2.0 model is shown in Fig. 2.10. The raw waveform input is first transformed into vector representations $\mathbf{z}$ using a CNN for each speech chunk of 25 ms. These vectors are then sent to a Transformer context encoder to be encoded into context representations $\mathbf{c}$, with certain vectors being masked for prediction. The speech representations $\mathbf{z}$ are also quantised into $\mathbf{q}$ which is an entry in a finite-size codebook (analogous to a fixed vocabulary for word tokens) using the product quantisation method (Jégou et al., 2011).

However, since there is no directly available target for pre-training, such as a word token, contrastive loss is employed. Conceptually, contrastive loss minimises a defined distance between positive pairs of data and maximises the same distance between negative pairs of data. In the case of wav2vec 2.0, the positive pair is $\mathbf{c}_t$ and $\mathbf{q}_t$ if it is masked at time t, and negative pairs are $\mathbf{c}_t$ and $\mathbf{q}_k$ where $k \neq t$. Formally, the contrastive loss for training wav2vec 2.0 is shown in Eqn. 2.43.

$$\mathcal{L}_m = -\log \frac{\text{sim}(\mathbf{c}_t, \mathbf{q}_t)}{\sum_{\hat{\mathbf{q}}_k \in \mathbf{Q}_t} \text{sim}(\mathbf{c}_t, \hat{\mathbf{q}}_k)} \tag{2.43}$$

where $\mathbf{Q}_t$ contains $K+1$ quantised representations including the true entry $\mathbf{q}_t$ and K distractors. Distractors are uniformly sampled from other masked time steps of the same utterance. The similarity measure in wav2vec 2.0 is the cosine distance. The full wav2vec 2.0 pre-training process also includes a diversity loss to encourage the model to explore all entries in the codebook and avoid mode collapse.

Other speech foundation models such as HuBERT (Hsu et al., 2021) have also explored unit discovery mechanisms to create pseudo labels, and WavLM (Chen et al., 2022) explored an addition denoising task in pre-training. These foundation models, after fine-tuning, often achieve superior performance than supervised training models trained on specific data on a wide range of speech-related tasks (wen Yang et al., 2021).

2.5 Summary

This chapter introduced the basic building blocks of deep learning, together with optimisation algorithms for DNNs. The training techniques and regularisation methods were also introduced. Those approaches will be frequently referred to in the rest of this thesis. The pre-trained foundation models mentioned in this section are widely applied to both end-to-end ASR and SLU, which are introduced in Chapter 3.

Chapter 3

Automatic Speech Recognition and Understanding

Automatic Speech Recognition (ASR) involves the transformation of speech signals into their corresponding textual representation. The task is challenging for real-world applications due to the presence of both inter-speaker and intra-speaker variability, the former being characterised by physiological differences and pronunciation distinctions across accents or dialects, while the latter is marked by differences in speech styles. The efficacy of speech-to-text conversion is further complicated by other factors such as channel distortion, reverberation, and background noise, which exacerbate the challenges faced by the ASR system. (Jurafsky and Martin, 2009).

There exist two main contemporary frameworks for ASR. The first framework is based on the noisy Source-Channel Model (SCM) which employs Hidden Markov Models (HMMs) to model the acoustic sequence from a generative perspective (Shannon, 1948, Jelinek, 1998, MacKay, 2003). This approach comprises several modules, including an acoustic model, a language model, a pronunciation model and a decoder. The modular structure necessitates the optimisation of each component separately, leading to a lengthy pipeline and potential error accumulation over stages. Nonetheless, the framework's advantage lies in the ease of incorporating various forms of structured knowledge, such as phonetic and linguistic information.

In recent years, the use of end-to-end trainable systems has emerged as a promising alternative to the noisy SCM for ASR. Unlike the SCM, end-to-end systems adopt a joint optimisation approach towards a single objective from a discriminative perspective. This leads to a simpler system design as only a single model needs to be optimised, in contrast to the modular design of SCM-based systems. The use of very large models with billions of

parameters and a vast amount of training data has allowed end-to-end systems to outperform SCMs and achieve state-of-the-art results in ASR tasks.

This section focuses on two widely used end-to-end trainable models: the attention-based encoder-decoder (AED) (Chorowski et al., 2015) model and the recurrent neural network transducer model (RNN-T) (Graves, 2012). Both models directly learn the probability of the transcription sequence given the input acoustic sequence.

In addition to ASR, this section also introduces spoken language understanding (SLU) as another task where contextual biasing can be applied. SLU aims to comprehend spoken language accurately and extract relevant information, including slot filling and intent classification (Tur and Mori, 2011). The former refers to the task of filling in the appropriate value for a given slot (e.g., restaurant or hotel names), while the latter involves classifying the user's intent in an utterance into one of several predetermined categories. SLU is a critical component in facilitating successful human-machine interactions in various types of conversational AI systems, particularly task-oriented spoken dialogue systems. The recent advancement in end-to-end SLU systems has been made possible by integrating ASR and NLU modules through neural connections. These systems either use a trainable neural interface between the ASR decoder and the NLU module or directly feed the output of the ASR encoder into a sequence generation-based NLU module.

ASR and SLU models require a large amount of labelled data for practical training. However, acquiring labelled data is a costly process. To overcome this challenge, foundation models trained with self-supervised learning (SSL) have been proposed as a viable solution, enabling models to learn meaningful feature representations from a large amount of unlabelled data (van den Oord et al., 2018, Devlin et al., 2019, Radford et al., 2019). Pre-trained models can be used to initialise various parts of ASR and SLU systems for subsequent fine-tuning to improve their performance (Baevski et al., 2020, Chen et al., 2019a). Based on the foundation models introduced in Sec. 2.4, this chapter also presents applications of those foundation models to end-to-end ASR and SLU, respectively.

3.1 Source-Channel Models

The SCM approach starts by converting the waveform into a sequence of input features $\mathbf{O} = \mathbf{o}_1, \mathbf{o}_2, ..., \mathbf{o}_T$ through a feature extraction stage. Then, the task of the SCM can be formulated as Eqn. 3.1.

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\text{argmax}}\, P(\mathbf{W}|\mathbf{O}) = \underset{\mathbf{W}}{\text{argmax}}\, p(\mathbf{O}|\mathbf{W})P(\mathbf{W}), \tag{3.1}$$

where $\mathbf{W} = w_1, w_2, ..., w_n$ is the word sequence and the final expression is derived using Bayes' rule and the normalising factor $P(\mathbf{O})$ is independent of $\mathbf{W}$. This section introduces important components of SCMs, including feature extraction, DNN-HMM acoustic modelling, discriminative training and decoding algorithms for DNN-HMM systems.

3.1.1 Feature Extraction

Feature extraction as the front-end for most ASR systems, converts the waveform data in a fixed-duration segment of speech, known as a *frame*, into a fixed-length vector $\mathbf{o}_t$ using signal processing techniques. In this report, frame lengths are 25 ms with a 10ms shift between adjacent frames. These input features (a.k.a. acoustic features) are expected to carry sufficient information from the speech signal while suppressing the influence of artefacts such as microphone, channel and background noise. Commonly used acoustic features are Mel-Frequency Cepstral Coefficients (MFCCs) (Davis and Mermelstein, 1980), Mel-scale Filter Banks (FBANK) (Mohamed et al., 2012) and Perceptual Linear Prediction (PLP) (Hermansky, 1990). The choice of features depends on the acoustic models, and for the DNN-HMM acoustic models covered in this section, FBANK features will be mainly used.

To incorporate dynamic information into speech features, all the aforementioned features can be augmented by delta features. The first-order delta feature can be obtained by a linear combination of the static features of a certain number of frames surrounding the current one. The second-order delta features can be computed analogously. The augmented feature with the first and second-order statistics is done by concatenating them into one single feature vector. The obtained acoustic features are usually normalised using mean and variance computed using frames in an utterance or using all frames in the training set.

3.1.2 Modelling Units

Modelling from acoustics to words directly poses significant challenges due to the data sparsity problem (Soltau et al., 2017). In ASR tasks with a large vocabulary, most words occur only a few times or not at all in the dataset, making it difficult to model them directly. Moreover, expanding the vocabulary size is not a trivial task when using words as modelling units. Therefore, phone-based units or character-based units are commonly employed in ASR to address this issue.

Phonetic-based units are frequently utilized in combination with a pronunciation dictionary or lexicon, as phones frequently function as natural units for defining word pronunciations. While the use of context-independent phones is straightforward, context-dependent phones such as biphones or triphones (Lee, 1988) are more commonly employed due to their

provision of more fine-grained subword units. This is advantageous in speech recognition tasks where co-articulation effects play a significant role.

In contrast to phonetic-based units, graphemic-based units do not rely on the use of a lexicon, and their set can be extended based on context or position within words (Kanthak and Ney, 2002). However, the use of graphemic units in acoustic modelling poses a greater challenge to acoustic models, particularly for languages such as English that exhibit orthographic irregularities, where a single pronunciation may correspond to multiple graphemes or multiple pronunciations may correspond to a single grapheme.

3.1.3 DNN-HMM Acoustic Modelling

HMM provides effective modelling of the continuous and dynamic characteristics in the speech signal. This type of model is based on the rightmost expression in Eqn. 3.1, where $P(\mathbf{O}|\mathbf{W})$ is predicted by the HMM-based *acoustic model* (AM) described later in this section, and $P(\mathbf{W})$ is predicted by the language model (LM).

HMMs are a type of *Markov model* which consists of states following a *Markov process* and observations depending on the states. States in HMM are not directly observable ("hidden"), but can be inferred from the observations. In HMM-based acoustic models, each modelling unit such as a letter or a phone is associated with an HMM based on the assumption that signals corresponding to the unit can be characterised by a first-order HMM. Specifically, given sequence of discrete states $s_1, s_2, ..., s_T$ where $s_i \in \{0, 1, ..., N\}$, the joint probability of states can be factorised using the first-order Markov chain in Eqn. 3.2.

$$P(s_1, s_2, ..., s_T|\lambda) = \prod_{i=1}^{T} P(s_i|s_{i-1}, \lambda), \tag{3.2}$$

where λ refers to a specific HMM. Note that a virtual entry state s_0 is introduced in Eqn. 3.2. Each conditional probability in Eqn. 3.2, known as a *transition probability*, can be characterised by a matrix A where $A_{mn} = P(s_i = m|s_{i-1} = n)$ which is, in general, different for different HMMs and is pre-determined for each HMM used in ASR. Observations of the HMM depend on the hidden states, and the probability of those can be written as Eqn. 3.3.

$$p(\mathbf{O}|s_1, s_2, ..., s_T, \lambda) = \prod_{t=1}^{T} p(\mathbf{o}_t|s_t, \lambda), \tag{3.3}$$

where each probability, known as the *emission probability*, can be estimated using specific models such as a *Gaussian Mixture Model* (GMM) or a DNN. Particularly in ASR, the first hidden state and the last hidden state are assumed to be non-emitting states which do not

have observations. By marginalising Eqn. 3.3 over all hidden states, the probability of the output sequence for a given HMM is shown in Eqn. 3.4.

$$p(\mathbf{O}|\lambda) = \sum_{s_{1:T} \in S} p(\mathbf{O}, s_1, s_2, ..., s_T|\lambda) \tag{3.4}$$

The following sub-sections will focus on using DNNs to model emission probabilities.

3.1.4 DNNs in Hybrid ASR Systems

DNNs are used to estimate the emission probability of HMM states in an HMM-based ASR system. There has been an interest in DNN acoustic modelling for decades (Waibel et al., 1989, Bourlard and Morgan, 1993), but the dominance came after the recent advancement of deep learning and efficient training techniques. DNNs are used to predict the probability of an individual observation vector $\mathbf{o}_t$ belonging to a specific state in a specific HMM, $p(s_t|\mathbf{o}_t;\theta)$ which is known as the state posterior distribution. Note that we compress the combination of s_t and λ into s_t here representing a specific phoneme state. This type of DNN-HMM system is also called the *hybrid system* and many DNN structures introduced in Chapter 2 have been applied, including MLPs and RNNs. To convert this distribution into an estimation of the emission probability distribution, Bayes' rule is applied again as shown in Eqn. 3.5.

$$p(\mathbf{o}_t|s_t;\theta) = \frac{P(s_t|\mathbf{o}_t;\theta)p(\mathbf{o}_t)}{P(s_t)}, \tag{3.5}$$

where $P(s_t|\mathbf{o}_t)$ is predicted by the DNN with $\mathbf{o}_t$ being the input and the output is a Softmax distribution over states. Note that $p(\mathbf{o}_t)$ can be ignored as it is irrelevant to the prediction. $P(s_t)$ are unigram probabilities estimated from the training data. This is done by aligning the training data with transcription to get a sequence of HMM states representing the most probable path through the concatenated HMMs of phones. Then the state-level unigram probabilities can be estimated using a standard frequency count.

3.1.5 Discriminative Sequence Training for DNN-HMM Systems

Optimisation of the DNN-HMM system is another challenging task. DNN-HMM systems can be trained with the maximum likelihood estimation (MLE) criterion. According to Eqn. 3.5, maximising the likelihood of the joint distribution of observation and state sequences is equivalent to maximising each individual state posterior probability $p(s_t|\mathbf{o}_t;\theta)$. This can be done using the cross-entropy criterion to perform a state classification where the target state

at each frame, s_t, can be obtained by running the Viterbi algorithm to align phoneme states with acoustic features using a GMM-HMM model.

However, the MLE approach is an approximately optimal solution to ASR provided that the acoustic and language models are close to the true distribution and that the training data is sufficiently large (Brown, 1987, Juang et al., 1997). Unfortunately, neither of these assumptions holds given the HMM assumptions and resource limitations for many tasks. Consequently, the maximum likelihood training of the acoustic model deviates from optimal performance. The MLE approach aims to maximize the likelihood of the training data with respect to the acoustic model for the associated reference sequences, thereby treating all alternative hypotheses put forth by the acoustic model without any penalisation.

Discriminative sequence training methods were introduced to accommodate the realistic condition where the amount of data is finite and the model is only an approximation rather than exactly correct. When either condition is not satisfied, the maximum likelihood criterion that directly optimises $P(\mathbf{O}|\mathbf{W})$ is not optimal. Discriminative sequence training aims to maximise the posterior probability given the observation, which can be applied at the sequence level. Several efforts have been made to improve the performance, among which the maximum mutual information (MMI) and minimum Bayes risk (MBR) training criteria have shown promising results.

MMI (Bahl et al., 1986, Woodland and Povey, 2002) maximises the mutual information between the target sentence $\mathbf{W}$ and the acoustic observation sequence $\mathbf{O}$. The objective can be followed as shown in Eqn. 3.6.

$$\mathcal{L}_{MMI}(\theta) = I(\mathbf{W},\mathbf{O}|\theta) = \log \frac{p(\mathbf{W},\mathbf{O}|\theta)}{P(\mathbf{W})p(\mathbf{O}|\theta)}. \tag{3.6}$$

θ is the collection of model parameters. The language model probability is not jointly optimised with the acoustic model as the LM is usually trained on very large corpora. The criterion can be re-written in Eqn. 3.7 with $P(\mathbf{W})$ with a fixed $P(\mathbf{W})$.

$$\mathcal{L}_{MMI}(\theta) = \log \frac{p(\mathbf{O}|\mathbf{W},\theta)^{1/k}P(\mathbf{W}|\theta)}{\sum_{\mathbf{W}'} p(\mathbf{O}|\mathbf{W}',\theta)^{1/k}P(\mathbf{W}'|\theta)}. \tag{3.7}$$

The denominator is the probability of the observation sequence given the model which is marginalised over all possible word sequences. In practice, a lattice containing the highest probable word sequences is used to approximate the total search space of word sequences, and k is the language model scaling factor which is to scale down the dynamic range of acoustic score to yield a broader posterior probability distribution. More recently, to release

the demand of calculating the denominator lattice, Povey et al. (2016) proposed a lattice-free MMI approximation to calculate the denominator efficiently without lattice.

Alternatively, to close the gap between the training objective and evaluation metrics, which is the error rate, error-based training has been developed using the MBR training criterion. MBR training minimises the expected value of the Bayes' risk function as shown in Eqn. 3.8.

$$\mathcal{L}_{MBR}(\theta) = \sum_{\mathbf{W}'} P(\mathbf{W}'|\mathbf{O};\theta)\mathcal{L}'(\mathbf{W}',\mathbf{W}), \tag{3.8}$$

where $\mathcal{L}'(\mathbf{W}',\mathbf{W})$ is the Bayes' risk function which can be the number of word or phone errors measured between the hypothesis sequence and the reference sequence. It is termed minimum word error (MWE) training if measured at the word level and minimum phone error (MPE) training (Povey and Woodland, 2002) if measured at the phoneme level. This training objective can also be applied to end-to-end trainable ASR systems.

3.1.6 Language Modelling and N-gram LMs

Language modelling computes the probability of the occurrence of a given sequence of words or sub-word units, $P(\mathbf{W})$. It provides the prior distribution for Eqn. 3.1 and is core to a wide range of speech and language tasks including language understanding, machine translation and dialogue systems. The following sections will explain approaches based on the word level, but the same approaches can be easily extended to sub-word units. The joint probability of all words in the sequence can be decomposed into the multiplication of conditional probabilities using the chain rule as shown in Eqn. 3.9.

$$P(\mathbf{W}) = P(w_0, w_1, ..., w_N) = \prod_{i=1}^{N} P(w_i|w_{i-1}, ..., w_1, w_0), \tag{3.9}$$

where N is the total length of the word sequence, and w_0 is always the symbol representing the start of the sequence. Therefore, the task of computing the joint probability can be transformed into computing the word prediction probability given the other words that the model has seen in this sequence which is referred to as the history information. Different LMs have different ways to represent history information, and the quality of the representation of the history information significantly affects the performance of LMs.

The performance of LMs is usually evaluated by the *perplexity*. Given a word sequence of length N, the per word perplexity (PPL) of the LM is shown in Eqn. 3.10.

$$\log_2 \text{PPL} = -\frac{1}{N}\log_2(P(\mathbf{W})) = -\frac{1}{N}\sum_{i=1}^{N}\log_2(P(w_i|w_{i-1}, ..., w_1, w_0)). \tag{3.10}$$

A language model with lower perplexity is considered to provide more accurate word prediction probabilities than the one with higher perplexity. Hence, PPL provides an important metric to evaluate the quality of a language model on its own.

3.1.6.1 N-gram LMs

The n-gram LM has been the dominant LM for decades because of its efficiency. Eqn. 3.9 requires a full-history representation for the joint probability to be calculated, however, word prediction has much stronger dependencies in immediately preceding words than those further distant in the history. Therefore, n-gram LM adopts an $n-1$-th order Markov assumption to truncate the number of words in the history to $n-1$ words as shown in Eqn. 3.11.

$$P(w_i|w_{i-1},...,w_1) \approx P(w_i|w_{i-1},...,w_{i-n+1}) \tag{3.11}$$

Each conditional probability can be estimated by counting the number of occurrences of such n-word sequence normalised by the number of occurrences of such $n-1$ word history sequence in the training corpus. Maximum likelihood estimation is used to train the LM.

The data sparsity problem and the n-th order Markov assumption are considered the two major issues associated with n-gram LMs. Because the estimation is based on the count of occurrences of partial word sequences in the training data, it requires sufficiently large counts so that the probability estimation has a small variance. As the order n increases, the number of possible partial sequences increases exponentially, so the occurrences become more sparse, with sequences not in the training data being assigned zero probability by the model. To mitigate this problem, various *smoothing* methods have been applied to the probability estimation such as Katz Smoothing Katz (1987) and Kneser-Ney Smoothing Kneser and Ney (1995). Smoothing of an n-gram LM usually follows the discounting and back-off procedure. In particular for Kneser-Ney smoothing, if an n-gram appears in the training data, it will be discounted by subtracting a value $C < 1$ known as *absolute discounting*. Otherwise, a smoothed maximum-likelihood estimation for a lower-order n-gram will be used to back off the prediction. More details can be found at Chen (2017).

In addition to the inefficient use of history information and the associated data sparsity problem, the n-th order Markov assumption that ignores words prior to the (n-1)-th preceding also reflects its limitations in capturing long-term dependencies within the sequence. The data sparsity issue can be mitigated by using a feed-forward neural network LM (NNLM), but it was not until the introduction of the RNNLM that the limitation of the Markov assumption has been effectively addressed. RNNLMs and more advanced neural LMs are introduced in later sections.

3.1.7 Decoding with DNN-HMM Systems

Decoding is the process of searching for the best sequence of words from the model predictions. For DNN-HMM systems, it combines the AM and the LM for a *maximum a posteriori* (MAP) prediction. Given the AM and the LM, the decoding process searches for the path with the highest probability defined in Eqn. 3.1 which is a combination of AM probability $p(\mathbf{O}|\mathbf{W})$ and LM probability $P(\mathbf{W})$. The AM gives the probability of the observation given a phone sequence, and the *lexicon* defines the phone sequence associated with each word. Then, the LM calculates the probability of candidate word sequences to select the best one. Specifically, decoding can be written in Eqn. 3.12.

$$\begin{aligned}\hat{\mathbf{W}} &= \underset{\mathbf{W}}{\operatorname{argmax}} p(\mathbf{O}|\mathbf{W})P(\mathbf{W}) \\ &= \underset{\mathbf{W}}{\operatorname{argmax}} \left[\sum_{\mathbf{\Lambda}}\sum_{\mathbf{S}} p(\mathbf{O},\mathbf{S}|\mathbf{\Lambda})P(\mathbf{\Lambda}|\mathbf{W})\right]P(\mathbf{W}) \\ &\approx \underset{\mathbf{W}}{\operatorname{argmax}} \left[\max_{\mathbf{S}} p(\mathbf{O},\mathbf{S}|\mathbf{\Lambda}) \max_{\mathbf{\Lambda}} P(\mathbf{\Lambda}|\mathbf{W})\right]P(\mathbf{W}),\end{aligned} \tag{3.12}$$

where $\mathbf{\Lambda}$ represents possible phone sequences given word $\mathbf{W}$ and $\mathbf{S}$ represents possible state sequence given the phone sequence. Since enumerating all possible sequences is intractable, a maximum approximation is applied to replace the summation. To find the best state sequence given the AM, the Viterbi decoding algorithm is most commonly used which is a dynamic programming algorithm. Define $\Psi_j(t)$ as the maximum likelihood of observing speech sequence $\mathbf{o}_{1:t}$ and being in state j at time t. Given the observation sequence, the best partial state sequence ending at time t in state s_j can be recursively computed as shown in Eqn. 3.13.

$$\Psi_j(t) = \max_{s_1,s_2,\ldots,s_t} p(\mathbf{o}_{1:t}, s_{1:t-1}, s_t = j) = \max_i \{\Psi_i(t-1)P(s_t = j|s_{t-1} = i)\}\ p(\mathbf{o}_t|s_t = j), \tag{3.13}$$

with initialisation $\Psi_1(0) = 1$. The maximum likelihood of the entire observation sequence $\mathbf{O}$ is in Eqn. 3.14.

$$\Psi_N(T) = \underset{i}{\operatorname{argmax}}\{\Psi_i(T)P(s_T = N|s_{T-1} = i)\}, \tag{3.14}$$

where N is the last state.

In continuous speech recognition, the transition probability between the state in one word to the state in another word is combined with the probability from the LM. Specifically, $P(s_t = j|s_{t-1} = i)$ will be multiplied by $P(w_i|w_{1:i-1})^{\beta}$ where β is the LM scaling factor to adjust the dynamic range of LM probabilities. For numerical stability, the computation is

implemented in the log scale. The complete decoding objective is shown in Eqn. 3.15.

$$\hat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmax}} \left\{ \log p(\mathbf{O},\mathbf{S}|\mathbf{\Lambda}) + \mu \log P(\mathbf{\Lambda}|\mathbf{W}) + \beta \log P(\mathbf{W}) + \gamma \log |\mathbf{W}| \right\}, \tag{3.15}$$

where μ is the pronunciation probability scaling factor, γ is the insertion penalty and $L(\mathbf{W})$ is the length of the predicted word sequence. For a large vocabulary, the search space is large which makes it infeasible to perform an exhaustive search. Therefore, beam search is used to prune partial paths with a low likelihood by only retaining partial paths with a fixed beam width (i.e. log probability differences) of the best at a particular time. Beam search algorithms are also used to decode end-to-end ASR systems.

3.1.8 ASR Evaluation

ASR systems are evaluated by the word error rate (WER) which is measured using the Levinstein edit distance between the hypothesis and reference. The WER includes three types of errors, insertion (I), deletion (D) and substitution (S) and is calculated as Eqn. 3.16.

$$WER = \frac{I+D+S}{N} \times 100\%, \tag{3.16}$$

where N is the total number of words in the reference. A lower WER implies a better-performing speech recognition system. An example of WER calculation for a selected utterance in a conversation is shown in Fig. 3.16 where the WER for this specific utterance is $(1+1+1)/11 \times 100\% = 27.3\%$. In languages such as Chinese and Japanese where written

Ref:	We	will	probably	still	have	to	do	some	file	managing
Hyp:	We	will	probably		have	to	do	uh	some	file	manager
Errors:				D				I		S

Fig. 3.1 An example of WER computation with 1 insertion, 1 deletion and 1 substitution.

forms lack word boundaries, a comparable evaluation metric called Character Error Rate (CER) is frequently employed at the character level. It should be noted that both WER and CER can exceed 100% due to unbounded insertion errors. The Sentence Error Rate (SentER), a metric that represents the percentage of utterances containing at least one error, is occasionally used.

3.2 End-to-end ASR with Attention-based Encoder-Decoder

End-to-end trainable ASR models, as alternatives to the SCM approach, directly maximise the posterior probability of the output *word* sequence $\mathbf{W}$ given the input acoustic feature sequence O using a single DNN model, as shown in Eqn. 3.17.

$$\theta^* = \arg\max_{\theta} P(\mathbf{W}|\mathbf{O};\theta) \tag{3.17}$$

Such methods are usually divided into frame-synchronous and label-synchronous systems depending on whether the model generates one output for each frame or for each output token. The attention-based encoder-decoder (AED) model is one typical example of the label-synchronous system, which maps the input sequence of length T directly to the output token sequence of length N. Note that with the AED, the conditional probability is usually factorised via autoregressive decomposition, as shown in Eqn. 3.18.

$$P(\mathbf{W}|\mathbf{O};\theta) = \prod_{i=1}^{N} P(w_i|w_{1:i-1},\mathbf{O};\theta) \tag{3.18}$$

Before the attention mechanism was introduced, sequence-to-sequence models were usually achieved using encoder-decoder models (Sutskever et al., 2014). The encoder, often an RNN, encodes information of the entire sequence into a single vector representation. The decoder takes the encoded vector as input and generates the target sequence. The performance of this approach may be significantly limited by the information captured by the fixed-length vector and the sequential information associated with it, especially for long sequences. However, with the attention mechanism, the decoder is able to "attend to" or "focus on" the most relevant part of the input sequence at each step to generate the output sequence progressively.

The AED model approach has been successfully applied to a range of sequence-to-sequence tasks where irregular input-output mappings are required, such as neural machine translation (Bahdanau et al., 2014) and ASR (Chorowski et al., 2015, Bahdanau et al., 2016, Chan et al., 2016). Different from other sequence-to-sequence tasks, the alignment in ASR, though still irregular, is monotonic, which requires specific attention mechanisms to be designed. This section focuses on the application of AED to end-to-end ASR and introduces the modelling units for AED, popular AED model structures, and training and decoding algorithms.

3.2.1 Modelling Units

This section introduces the modelling units that are adopted by most end-to-end trainable ASR systems in general. While phone-based units may still remain effective for end-to-end ASR models, it relies on the availability of a lexicon and linguistic knowledge which increases the pipeline complexity of end-to-end systems. Therefore, graphemic modelling units are often viewed as more desirable for these systems. With abundant training data and adequate model complexity, graphemic AED systems have shown to yield equivalent or even superior performance to phonetic systems (Sainath et al., 2018), and it is able to learn word spellings from the training data.

On the other hand, pronunciations of graphemes are highly context-dependent, where several characters can map to a single sound. Moreover, certain graphemes may often appear together, such as “ing" for the present continuous tense. In general, a longer output unit keeps more output context. As a result, instead of using single characters as the output unit, word pieces as groups of characters have been widely adopted. To ensure that all words can be expressed as a set of word piece units, single characters are usually combined with the set of word pieces. One example of word piece representations is shown in Table 3.1.

Grapheme	e n c o d e r <space> d e c o d e r <space>
word piece (prefix)	_en co der _de co der _enc o der _dec o der
word piece (suffix)	en co der_ de co der_

Table 3.1 Illustration of word piece representing words in a target sequence. Note that <space> represents the space character which is marked by word boundaries in word pieces. Word boundary may either be marked as the prefix or suffix, using “_".

The set of word pieces is usually generated from a non-neural word piece model trained on the text training data. Two commonly used word piece models are introduced in this section, namely Byte Pair Encoding (BPE) (Gage, 1994) and unigram LM word pieces (Whittaker and Woodland, 2000, Kudo, 2018). BPE starts building the word piece model with graphemes. It iteratively groups the most common byte pair according to the grapheme bi-gram together to form a new unit, until a designated number of word piece units is reached. Alternatively, the unigram word piece model encodes each word using the most probable set of word pieces based on the unigram probabilities of each word piece by applying the Viterbi algorithm. The most probable set of word pieces is found based on the maximum unigram probability of the training corpus. It can be either found by iteratively increasing the number of word pieces until the desired vocabulary size is reached (Whittaker and Woodland, 2000),

or iteratively removing word pieces that cause large probability reductions (Kudo, 2018). Although the most probable encoding of a word is usually unique, there exists a vast number of alternative word piece combinations for a word, as shown in Table 3.1. Regularisation can be applied by taking advantage of multiple possible encodings (Kudo, 2018).

3.2.2 AED Model Structure

Based on the building blocks introduced in Chapter 2, this section introduces the RNN AED model and the Transformer AED model, respectively. Moreover, the convolution-augmented Transformer (Conformer) AED model is also introduced which has achieved state-of-the-art performance on ASR tasks. In this section, the input acoustic feature sequence is denoted using $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_T$, and the corresponding output word piece targets are $\mathbf{Y} = y_1, ..., y_N$.

3.2.2.1 RNN AED Model

The encoder and decoder in the RNN AED model are constructed using RNNs, or variations of RNNs such as the LSTM, which are connected using the attention mechanism (Bahdanau et al., 2014). The RNN encoder maps the input sequence $\mathbf{X}$ into intermediate representation sequence $\mathbf{H}$. If no subsampling is performed to the input, the sequence $\mathbf{H}$ has the same sequence length as $\mathbf{X}$. At each decoding step i, the encoded sequence $\mathbf{H}$ is first converted into a context vector $\mathbf{c}_i$ based on the attention mechanism. The attention mechanism allows the model to focus on relevant parts by assigning high attention weights to those parts, and hence the context vector can be seen as carrying the most relevant acoustic information from the encoder that is needed at the current decoding step i. The decoder, based on an RNN, then takes as input the context vector together with the previous output y_{i-1} and the previous decoder state $\mathbf{d}_{i-1}$ to predict a probability distribution over all modelling units for the current step. The computation procedure can be summarised as follows:

$$\mathbf{H} = \text{Encoder}(\mathbf{X}), \tag{3.19}$$

$$\mathbf{a}_i = \text{Attention}(\mathbf{a}_{i-1}, \mathbf{d}_{i-1}, \mathbf{H}), \tag{3.20}$$

$$\mathbf{c}_i = \mathbf{H}\mathbf{a}_i, \tag{3.21}$$

$$P(y_i|y_{1:i-1}, \mathbf{X}; \theta), \mathbf{d}_i = \text{Decoder}(y_{i-1}, \mathbf{d}_i, \mathbf{c}_i) \tag{3.22}$$

where $\mathbf{a}_i$ is the vector containing attention weights that sum up to 1. During the training process, the optimisation objective of the target unit y_i is to maximise the output probability given the ground truth history sequence $y_{1:i-1}$, commonly referred to as *teacher forcing*. Various decoding algorithms can be used, and greedy decoding is one of the simplest

examples among them. During greedy decoding, the output unit with the highest probability is generated and subsequently fed back into the next decoding step. The decoding procedure starts with the start-of-sentence symbol <s> assigned to the initial input y_0 of the decoder. The process concludes when the end-of-sentence symbol </s> is generated, which allows variable-length output sequences to be produced.

When using the standard RNN structure to implement the AED model. the structure is shown in Fig. 3.2.

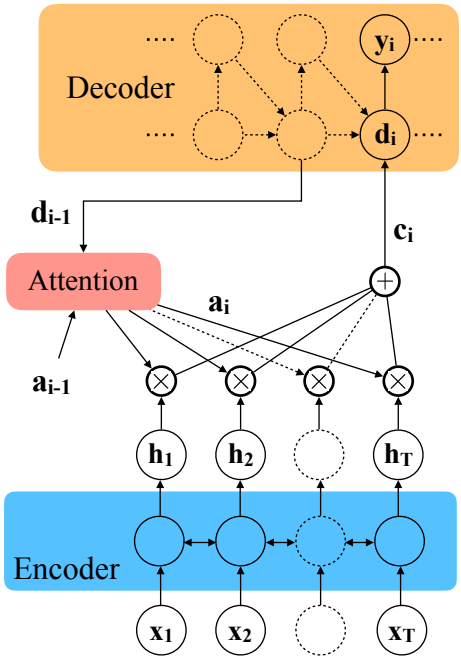


Fig. 3.2 AED model structure using a standard RNN.

Note that the encoder here uses a bi-directional RNN structure that captures information in both directions. Bi-directional RNN encoders usually yield better performance but require the model to see the entire sequence up to time T, which increases the latency of the output. The attention mechanism takes the decoder hidden state from the previous step, $\mathbf{d}_{i-1}$ as the query which may also include information from the previous attention weights $\mathbf{a}_{i-1}$. The keys are derived from $\mathbf{H}$, and values are $\mathbf{H}$. Originally, the AED approach was proposed using additive attention as described in Section 2.1.5.1. When it was adapted to the ASR task, content-based attention (Chan et al., 2016) as a generalised form of attention was used which adopts the following computation to obtain attention weights:

$$\hat{a}_{it} = < \phi(\mathbf{d}_{i-1}), \psi(\mathbf{h}_t) >, \tag{3.23}$$

$$\mathbf{a}_i = \text{Softmax}([\hat{a}_{i1}, ..., \hat{a}_{iT}]^T) \tag{3.24}$$

where $\phi(\cdot)$ and $\psi(\cdot)$ are two MLPs and $<\cdot>$ represents a scoring function which could be addition or cosine similarity. Alignment, which is a crucial part of ASR, is achieved by the attention weights in the AED model after training. This gives a “soft" alignment as shown in the example in Fig. 3.3. Figure 3.3 depicts a matrix of attention weights where each row

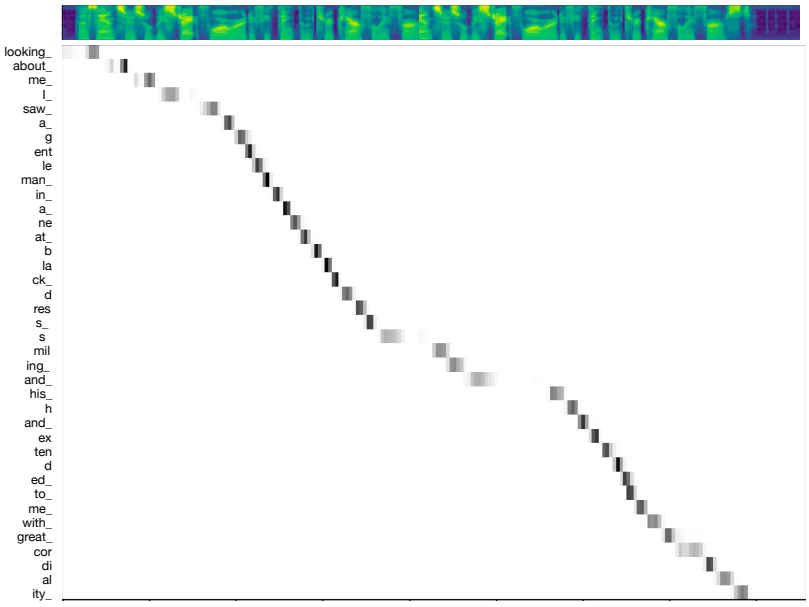


Fig. 3.3 Illustration of alignment achieved by the attention mechanism. The horizontal axis is the time steps of speech input, and the vertical axis contains the sequence of output tokens (word pieces) in top-to-bottom order.

represents a distinct set of weights utilised to generate a corresponding token. Each block within a given row corresponds to a particular a_{it} value, where darker shading indicates a higher attention weight. As a result, after training the model is able to focus on a rough region in the input corresponding to the output token at each decoding step.

Unlike many other tasks, ASR involves a monotonic alignment between the input and output. To encourage such monotonicity, specialised attention mechanisms have been developed. An instance of this is the *location-based* attention mechanism proposed by Chorowski et al. (2015). This approach uses the attention weights from the previous step as input to compute the attention weights for the current step. By considering the previous attention location, the mechanism is able to guide the current attention towards regions further ahead in the input, instead of focusing on past regions. The computation is shown

below.

$$[\mathbf{f}_{i1},...,\mathbf{f}_{iT}] = \mathbf{K} * \mathbf{a}_{i-1} \tag{3.25}$$

$$\hat{a}_{it} = \mathbf{w}^T tanh(\mathbf{W}_1\mathbf{d}_{i-1} + \mathbf{W}_2\mathbf{h}_t + \mathbf{W}_3\mathbf{f}_{it} + \mathbf{b}) \tag{3.26}$$

$$\mathbf{a}_i = \text{Softmax}([\hat{a}_{i1},...,\hat{a}_{iT}]^T) \tag{3.27}$$

where $\mathbf{K} \in \mathbb{R}^{P\times Q}$ is a trainable convolution kernel matrix, and $\mathbf{w}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{b}$ are model parameters. Each one of the P one-dimensional convolution kernels, of dimension Q, is convolved with the previous attention weights at each time step t to generate one vector of dimension T. The stack of these vectors is $\mathbf{F}_i = [\mathbf{f}_{i1},...,\mathbf{f}_{iT}]$ where each $\mathbf{f}_{it}$ is a column vector of dimension P (since there are P kernels).

Other work on achieving better attention quality for AED models has been proposed in ASR under different task specifications, including ones that enforce monotonicity (Raffel et al., 2017, Chiu and Raffel, 2017) or leverage the locality characteristics (Tjandra et al., 2017). It is worthwhile pointing out that, both content-based and location-based attention require the entire input sequence to be present, and hence they are usually not suitable to work in streaming mode. Work in (Raffel et al., 2017, Chiu and Raffel, 2017) provided alternative attention mechanisms as solutions and allowed AED models to work in streaming mode. Moreover, AED can also be combined with SCM as a re-scoring system to achieve improved performance (Li et al., 2023, Li, 2022).

3.2.2.2 Transformer AED Model

The Transformer is naturally suited for the AED approach. It can also be applied to end-to-end trainable ASR tasks, where the acoustic features are fed into the encoder and the decoder takes the word piece token sequence. One difference when applied to speech tasks is the positional embedding. The positional embedding in a standard Transformer is designed to represent the absolute position of each element in a sequence. However, for lengthy utterances with comparable segments in distinct positions, the application of positional embeddings may lead to incorrect positional attention, particularly if the model is not trained on similar data. Speech representations are expected to be insensitive to temporal shifts, such as delays in the onset of an utterance. To address this issue, relative positional encoding (RPE) has been proposed (Yang et al., 2019, Gulati et al., 2020) as a more effective approach for encoding the relative position within a sequence in the context of speech-related tasks using Transformer models. The RPE approach provides advantages in the ability to better handle sequences with similar segments and maintain invariance to time shifts, which are desirable characteristics for speech-related tasks.

The relative position is defined as the position difference between the query at i and the key at j. Then, a learnable embedding matrix $\mathbf{R}$ is used where for each possible value of $i-j$, there is a corresponding representation $\mathbf{r}_{i-j}$. This embedding is added to the computation of the attention score as follows, using self-attention as an example.

$$\mathrm{a}_{ij} = \frac{\mathbf{x}_i \mathbf{W}_Q (\mathbf{x}_j \mathbf{W}_K + \mathbf{r}_{i-j})^T}{\sqrt{d_k}} \tag{3.28}$$

In general, the Transformer architecture presents a compelling alternative to RNN-based AED structures for speech-related tasks. By enabling enhanced modelling of long-range dependencies and enabling high levels of parallelisation of computation, Transformers have demonstrated improved performance over RNN-based models. The Transformer architecture may be utilised either in its entirety for ASR, or the encoder component may be used independently to leverage the benefits of long-sequence modelling in conjunction with an RNN-based decoder. Many modifications to the Transformer architecture have been proposed for ASR tasks, with the Conformer model representing one of the most successful and promising developments in this area.

3.2.2.3 Conformer and Conformer AED

The Conformer model (Gulati et al., 2020) combines the advantage of long-range dependency modelling using an attention mechanism and the local correlation modelling of CNN into one Transformer block. The details of the Conformer structure are shown in Fig. 3.4.

Two main modifications to the standard Transformer in the Conformer are the additional convolution module and the two half-step FFNs which replaces the original single FFN in the Transformer. The convolution module adopts two types of convolution: the pointwise convolution and the 1D depthwise convolution. Pointwise convolution applies convolution kernels of $C_{\text{out}} \times 1$ to the input of shape $C_{\text{in}} \times T$ where C_{in} is the number of input channels (e.g. the dimension of input acoustic features) and C_{out} is the number of output channels. The output of this is of size $C_{\text{out}} \times T$. The first pointwise convolution has $C_{\text{out}} = 2C_{\text{in}}$, and the second one has $C_{\text{out}} = C_{\text{in}}$. The gated linear unit (GLU) (Dauphin et al., 2017) after the first pointwise convolution uses the second half (i.e. channels C_{in} to $2C_{\text{in}}$) to perform an element-wise gating activation on the first half (i.e. channels 1 to C_{in}). The depthwise convolution uses a kernel of size $C_{\text{out}} \times K$ to convolve along the time dimension, where K is the kernel size.

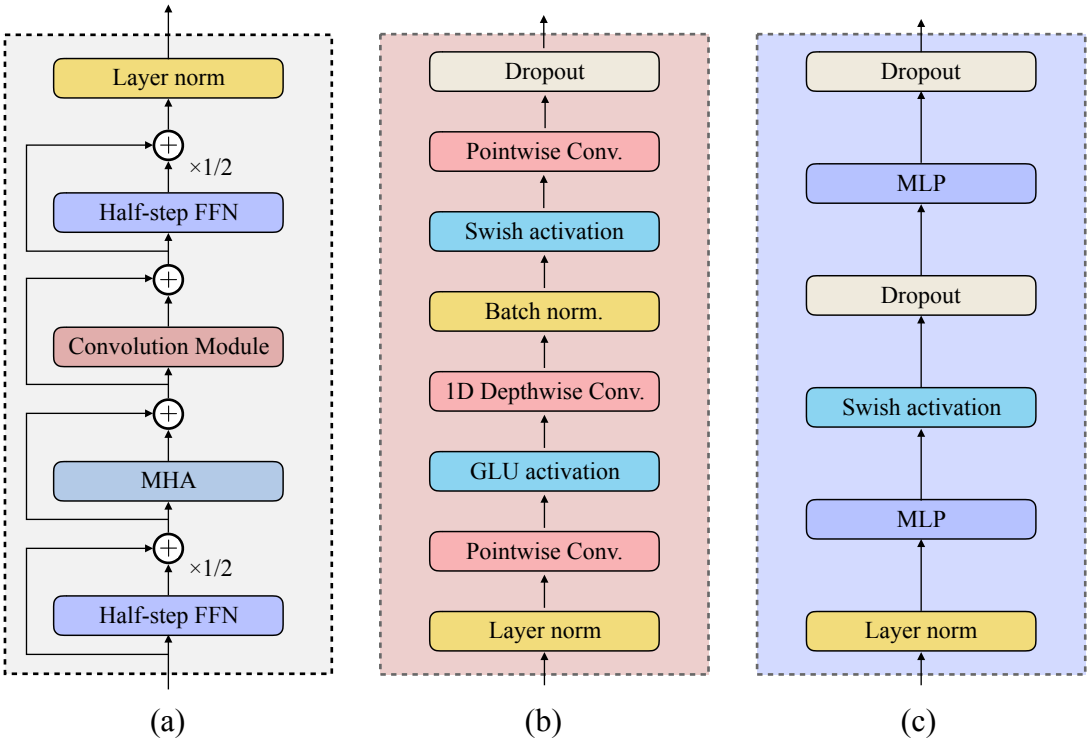


Fig. 3.4 The Conformer Structure. (a). Overall structure. (b). The convolution module. (c). The half-step FFN module.

Both the convolution module and the FFN module adopt the Swish (Ramachandran et al., 2018) activation function, which takes the following form.

$$\text{Swish}(x) = x \cdot \sigma(x) \tag{3.29}$$

This activation function is often regarded as a smoothed version of ReLU, which is differentiable everywhere. Swish tends to perform better than ReLU for very deep networks.

As for the Transformer encoder, the Conformer can be used as the encoder in the AED model where the decoder can either be a Transformer or an RNN. It can also be used in other ASR models such as in the AM of the SCM or in RNN-T which is introduced in Section 3.3.

3.2.2.4 Frontend

In ASR, the input sequence is often much longer than the output word piece sequence, which makes it very challenging for attention mechanisms to learn the alignment. Meanwhile, modelling very long sequences without subsampling can be computationally very expensive, especially for RNNs. Therefore, frontend processing approaches have been adopted to downsample acoustic features before sending them to the model. Subsampling frontend

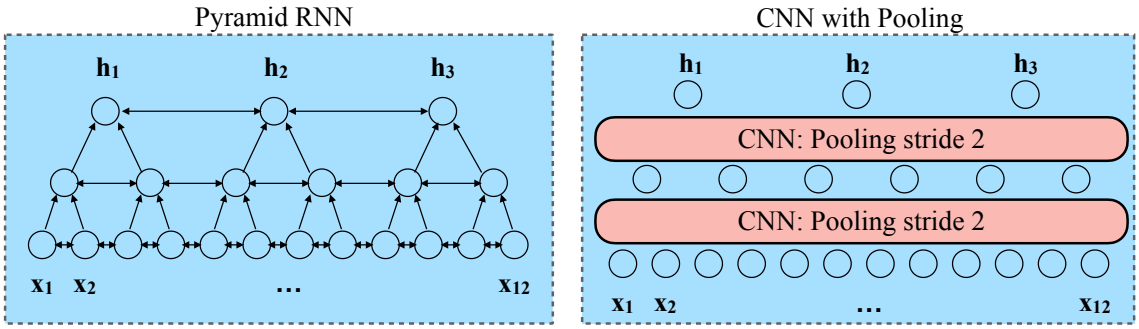


Fig. 3.5 Frontend processing using a pyramid RNN (left) or a stack of CNNs (right) with stride 2 pooling.

modules are often achieved by using a pyramid RNN (Chan et al., 2016, Kim et al., 2016) or a stack of CNN layers with pooling (Hori et al., 2017, Zhang et al., 2016, Gulati et al., 2020), as shown in Fig. 3.5.

Note that both the pyramid RNN and the CNN shown in Fig. 3.5 achieve an effective subsampling of 4. CNNs are also able to achieve robustness to noise and channel distortions in ASR (Qian and Woodland, 2016).

3.2.2.5 Foundation Models as Encoders

Speech foundation models, such as wav2vec2.0 (Baevski et al., 2020), HuBERT (Hsu et al., 2021) and wavLM (Chen et al., 2022), have also been applied to ASR, where most of them work as encoders or feature extractors. When fine-tuning these models, the parameters of the CNN feature encoder are usually not updated, and it often requires the foundation model to be frozen for a number of updates. A tri-state learning rate scheduler comprising a warmup state, a steady state and a linear decay stage is often used to fine-tune these models. Foundation models as feature encoders often achieve superior performance compared to models such as the Conformer trained from scratch (wen Yang et al., 2021), especially with a limited amount of supervised training data.

However, speech foundation models usually have hundreds of millions of parameters, and hence the inference latency of these models will be a bottleneck in their applications, leading to high cost and a significant environmental footprint. This can be mitigated by performing knowledge distillation (Hinton et al., 2014, Yang et al., 2022). Moreover, the size of foundation models may significantly limit their ability to be combined with other complex ASR models, e.g. the Conformer. Universal ASR models leveraging complex ASR structures and trained on a very large amount of available ASR data have shown a similar level of performance on many tasks without fine-tuning (Radford et al., 2023).

3.2.3 AED Training

For standard attention-based model training, a teacher-forcing approach is used. That means for each step forward in the decoder, the ground truth subword unit is used as the input to obtain the next output. For each output step, the CE loss is minimised between the output distribution over subword units and the target token represented using a one-hot vector with 1 for the index of the target and 0 otherwise. The loss function is shown in Eqn. 3.30.

$$\mathcal{L}^{\text{CE}} = \sum_{u} \sum_{i=1}^{N} -\log P(y_i^{(u)} | y_{0:i-1}^{(u)}, \mathbf{X}) \tag{3.30}$$

where the first sum is performed over all utterances in the training set, and the second is performed over the sequence of output tokens. The model can thus be optimised using gradient-based methods. Specifically for the Transformer and Conformer models, the Noam learning rate scheduler (Vaswani et al., 2017) is often used which takes the form in Eqn. 3.31.

$$\text{learning rate} = \frac{\min(\text{step}^{-0.5}, \text{step} \times \text{warmup}^{-1.5})}{\sqrt{d_{\text{model}}}} \tag{3.31}$$

where the step is the current counter for the number of updates, warmup is a hyper-parameter determining the number of steps where the learning rate is increasing, and d_{model} is the input/output dimension of the Transformer blocks. This learning rate scheduler can be plotted as shown in Fig. 3.31 for different choices of hyper-parameters.

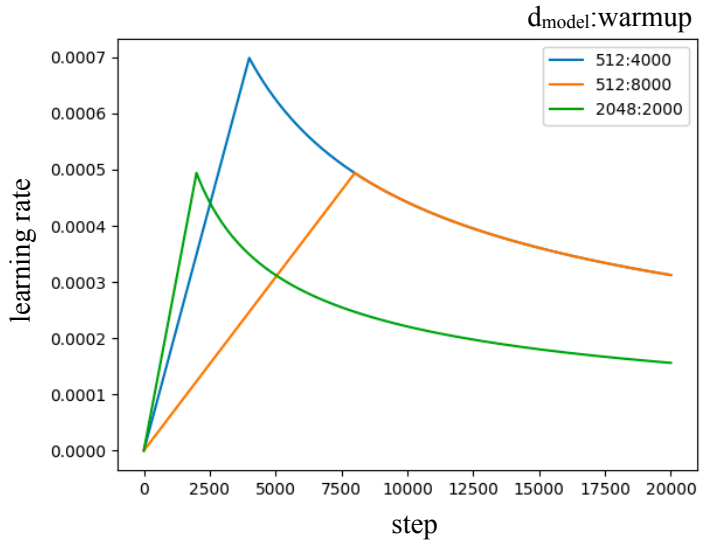


Fig. 3.6 Plot of learning rate against the number of update steps using the Noam learning rate scheduler under different settings of d_{model} and warmup.

In addition to the CE loss, sequence discriminative training approaches (Povey and Woodland, 2002), such as MWE training, can also be applied to AED models.

3.2.4 AED Decoding

Decoding is the process to find the hypothesis $\hat{\mathbf{Y}}$ that maximises the following probability

$$\hat{\mathbf{Y}} = \arg\max_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X};\theta) \tag{3.32}$$

The most straightforward decoding strategy is greedy decoding which selects the subword with the highest probability at each step during the decoding process. However, this approach often leads to a significant number of search errors. This is due to the decoder being autoregressive, where the distribution of the next output depends on the previously output tokens. As a result, the current most probable subword may not be included in the overall best sequence. Consequently, it is ideal to maintain multiple options at each decoding stage. Nevertheless, the number of hypotheses that must be tracked increases exponentially with decoding steps.

Beam search is a popular breadth-first heuristic search algorithm, which is commonly employed to generate high-quality results within the computational constraints of the system. For AED models, a straightforward search without pruning would result in an exponential expansion of the search tree as the sequence length increases. To overcome this problem, beam search for AED models only maintains at most N paths within the search at each stage, which are the most promising based on their likelihood scores. However, selecting a very small beam size would increase the risk of search errors, whereas a larger beam size would extend the decoding time linearly. To optimise the search, the hidden state of the decoder is cached for each path in the search tree to avoid redundant computation during the tree expansion. The decoding process terminates when the end-of-sentence token is generated, and the maximum number of decoding steps is typically limited to prevent excessively long sentences (often set to the input sequence length in AED).

As the alignment achieved by attention is not restricted to being strictly monotonic and covers the entire input sequence, the model may either get trapped in a certain region and repeatedly generate the same token or skip a segment of speech. Therefore, AED models often suffer more from insertion and deletion errors. To overcome both issues in decoding, a length penalty term is often added to penalise long hypotheses, and a coverage penalty term is applied to encourage the attention mechanism to cover the entire sequence. The criterion

in Eqn 3.32 can be rewritten as follows.

$$\hat{\mathbf{Y}} = \arg\max_{\mathbf{Y}} \Big\{ \log P(\mathbf{Y}|\mathbf{X};\theta) + \kappa_1 |\mathbf{Y}| + \kappa_2 \sum_{t=1}^{T} \mathbb{1}\Big(\sum_{i=1}^{K} a_{kt} > \tau \Big) \Big\}, \tag{3.33}$$

where the hyper-parameters κ's and τ require tuning on a separate validation set, and $|\mathbf{Y}|$ denotes the length of the hypothesis. The indicator function $\mathbb{1}(\cdot)$ returns the value 1 if the argument is true and zero otherwise. The final term represents the coverage penalty term, which takes into account the number of frames that have cumulative attention values greater than τ. The coverage term serves to prevent repetition issues, as it discourages the model from focusing excessively on previously attended frames. This is because revisiting multiple frames does not contribute to the overall coverage (Chorowski and Jaitly, 2016).

3.2.5 MWE Training for AED

To further improve the performance of the AED model trained with the CE loss, the MWE loss can be applied. The MWE loss in end-to-end ASR systems, in general, minimises the expected value of word errors across all possible output sequences of a given input. Denoting the word-level output sequence $\mathbf{W} = \{w_1, ..., w_L\}$ and input sequence $\mathbf{X} = \mathbf{x}_{1:T}$ for convenience, the MWE loss can be written as Eqn. (3.34).

$$\mathcal{L}^{\text{MWE}} = \sum_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) \mathcal{W}^{\text{MWE}}(\mathbf{W}, \mathbf{W}^*), \tag{3.34}$$

where $\mathbf{W}^*$ is the ground-truth output sequence and $\mathcal{W}^{\text{MWE}}(\cdot)$ is the risk function representing the number of word errors calculated using an edit distance between each possible sequence $\mathbf{W}$ and the ground-truth sequence $\mathbf{W}^*$. The sum is performed over all possible sequences and $P(\mathbf{W}|\mathbf{X})$ is the probability of a specific sequence calculated from the end-to-end ASR model output. As it is intractable to enumerate all possible sequences and calculate their probabilities, a common practice which has been widely adopted in MWE training for end-to-end ASR systems (Prabhavalkar et al., 2018, Weng et al., 2018, Wynands et al., 2022) is to use the N-best hypotheses to approximate the probable set of likely outputs and hence the expected word errors. The N-best hypotheses are obtained by applying the beam search algorithm for each utterance in a minibatch during training, and the approximation is shown

in Eqn. (3.35).

$$\begin{aligned}\mathcal{L}^{\text{MWE}} &\approx \frac{\sum_{\mathbf{W}_i \in \text{Beam}_N(\mathbf{X})} P(\mathbf{W}_i|\mathbf{X}) \mathcal{W}^{\text{MWE}}(\mathbf{W}_i, \mathbf{W}^*)}{\sum_{\mathbf{W}_i \in \text{Beam}_N(\mathbf{X})} P(\mathbf{W}_i|\mathbf{X})} \\ &= \sum\nolimits_{\mathbf{W}_i \in \text{Beam}_N(\mathbf{X})} \hat{P}(\mathbf{W}_i|\mathbf{X}) \mathcal{W}^{\text{MWE}}(\mathbf{W}_i, \mathbf{W}^*),\end{aligned} \tag{3.35}$$

where $\text{Beam}_N(\mathbf{W}) = \{\mathbf{W}_1, ..., \mathbf{W}_B\}$ is the N-best hypotheses obtained via beam search, and $\mathcal{W}^{\text{MWE}}(\mathbf{W}_i, \mathbf{W}^*)$ represents the edit-distance between two sequences. The probabilities of the N-best hypotheses are normalised to form a valid distribution, where each normalised probability is represented as $\hat{P}(\mathbf{W}_i|\mathbf{X})$. Moreover, the average number of word errors over the N-best hypotheses is often subtracted from the number of word errors in each hypothesis as a form of variance reduction.

The MWE loss can be applied to the AED model following the training scheme proposed by Prabhavalkar et al. (2018) which also interpolated with the CE loss for better training stability, as shown in Eqn. (3.36).

$$\mathcal{L} = \mathcal{L}^{\text{MWE}} + \mu \mathcal{L}^{\text{CE}}, \tag{3.36}$$

where $\mathcal{L}$ is the total loss function to be minimised, and $\mathcal{L}^{\text{CE}}$ is scaled by a hyper-parameter μ. As it is hard to train a randomly initialised model with the MWE loss, the MWE loss is applied from the epoch when the model is reasonably trained with the CE loss, which depends on the optimisation algorithms used. Moreover, to boost the efficiency of beam search which is the bottleneck of time complexity for MWE, parallelised beam search algorithms are often implemented (Sun et al., 2022a) by parallelising the model forward computation of all beams of all utterances in the same mini-batch on a GPU.

3.3 End-to-end ASR with RNN-T

AED is an end-to-end trainable ASR system design that generates one output per token while aligning the input speech with output tokens through an attention mechanism. However, since the attention mechanism typically requires the entire sequence as input, the ability of the AED model to work in a streaming mode is limited without the dedicated design of attention mechanisms (Raffel et al., 2017, Chiu and Raffel, 2017). Thus, the development of a frame-synchronous system design is essential to enable the model to generate output whenever it receives a new frame of speech input.

Two typical end-to-end ASR models that are frame-synchronous are the connectionist temporal classification (CTC) (Graves et al., 2006) and the neural transducer (NT) (Graves,

2012). In particular, this thesis specifically focuses on the neural transducer, specifically, the RNN Transducer (RNN-T) Note that transducer models with a Transformer/Conformer encoder are also referred to as an RNN-T throughout this thesis. The RNN-T provides a natural way for streaming ASR because of its dependence on the previous output tokens and the speech sequence up until the current frame. With this natural streaming capability, RNN-T has become the most widely-used end-to-end model in industry (Prabhavalkar et al., 2017, Sainath et al., 2020, Li et al., 2020a, He et al., 2019).

This section introduces the model structure of the RNN-T approach, and how it can be trained with error back-propagation together with the efficient decoding algorithms for RNN-T.

3.3.1 Model Structure

A typical RNN-T model comprises an encoder, a predictor and a joint network, as shown in Fig. 3.7 (a). The encoder encodes the acoustic feature sequence $\mathbf{x}_1, ..., \mathbf{x}_T$ into the sequence

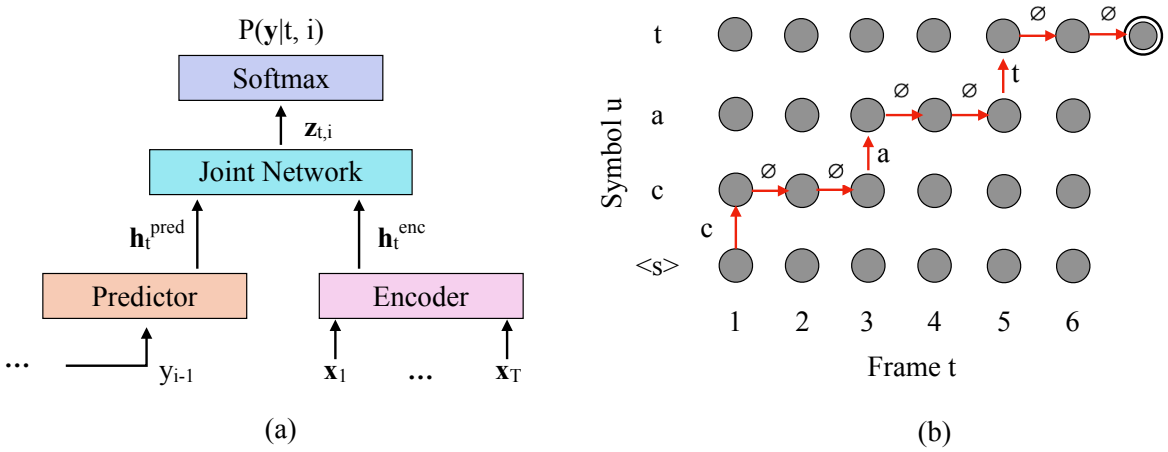


Fig. 3.7 RNN-T model and alignment illustration: (a) The overall RNN-T model. (b) A trellis diagram showing one possible alignment of the example text “cat" with 6 input frames.

of encoder hidden states $\mathbf{h}_t^{\text{enc}}$, similar to the encoder in AED and is analogous to the AM in SCM. Therefore, any techniques such as the frontend, Conformer structure and foundation models that can be applied to the AED encoder, can also be applied to the RNN-T encoder. The predictor, typically a uni-directional LSTM, encodes the previous tokens up to step $i-1$ into a compact vector representation $\mathbf{h}_i^{\text{pred}}$, which is analogous to the LM in an SCM. The joint network, which is often a shallow MLP, combines the text information from the predictor at step i and the acoustic information from the encoder at time step t, and output

$\mathbf{z}_{t,i}$ for each combination of (t,i).

$$\mathbf{z}_{t,i} = \tanh(\mathbf{W}^{enc}\mathbf{h}_t^{\text{enc}} + \mathbf{W}^{pred}\mathbf{h}_i^{\text{pred}} + \mathbf{b}) \tag{3.37}$$

After Softmax, this vector is applied and the distribution over all possible output tokens is produced.

$$P(y_i = a|y_{1:i-1}, x_{1:t}) = P_{t,i}(a) = \text{Softmax}(\mathbf{z}_{t,i})_a \tag{3.38}$$

To achieve alignment, in addition to all word piece units, RNN-T also includes a blank symbol, $\varnothing$, in the output space.

A trained RNN-T generates recognition results in the following way (assuming greedy decoding). At each encoder step t with decoded history $y_{1:i-1}$, a distribution $P(y_i|t,i)$ over all output tokens is calculated, and the token with the highest probability is chosen as the next token. Then, if the decoded token is $\varnothing$, the model moves on to the next encoder step and repeats the procedure. Otherwise, the model stays at the current encoder step and continues to generate the next token with the new history sequence $y_{1:i}$. The resulting hypothesis contains $N+T$ tokens including T $\varnothing$ symbols and N word piece tokens.

Therefore, the RNN-T achieves alignment by inserting T blank symbols. For instance, if the target sequence is “cat" and there are 6 frames, two possible alignments are “c$\varnothing\varnothing$a$\varnothing\varnothing$t$\varnothing\varnothing$" and “$\varnothing$ca$\varnothing\varnothing\varnothing\varnothing$t$\varnothing$". In the trellis diagram shown in Fig. 3.7 (b), moving upward corresponds to outputting a non-blank symbol and staying at the current frame, while moving rightward corresponds to outputting $\varnothing$ and moving to the next time step. Define each path leading from the bottom-left corner to the top-right one as one alignment $H \in \text{Align}(\mathbf{Y})$, the original target sequence $\mathbf{Y}$ or the recognition results can be obtained by removing all blank symbols from the obtained alignment, denoted as $\mathbf{Y} = \text{Align}^{-1}(H)$.

3.3.2 RNN-T Training

The RNN-T is trained by maximising the probability of the target sequence given the speech input, which is expressed as the sum of probabilities of all possible alignments. Given the input acoustic features $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_T$ and output sequence $\mathbf{Y} = y_1, ..., y_N$ as before, the RNN-T loss is shown in Eqn. 3.39.

$$\mathcal{L}_{\text{RNN-T}} = -\log P(\mathbf{Y}|\mathbf{X}) = -\log \sum_{H \in \text{Align}(\mathbf{Y})} P(H|\mathbf{X}) \tag{3.39}$$

The loss can be computed efficiently using the forward-backward algorithm (Baum, 1972), and the gradient w.r.t. the model parameters can be calculated by the chain rule starting from

the partial derivatives w.r.t. each arrow (i.e. each element in $P(y_i|t,i)$) can be calculated as:

$$\frac{\partial \mathcal{L}_{\text{RNN-T}}}{\partial \theta} = \sum_i \sum_t \sum_k \frac{\partial \mathcal{L}_{\text{RNN-T}}}{\partial P_{t,i}(k)} \frac{\partial P_{t,i}(k)}{\partial \theta} \tag{3.40}$$

The first term can then be arranged as

$$\frac{\partial \mathcal{L}_{\text{RNN-T}}}{\partial P_{t,i}(k)} = -\frac{1}{\sum_H P(H|\mathbf{X})} \frac{\partial \sum_H P(H|\mathbf{X})}{\partial P_{t,i}(k)} \tag{3.41}$$

Denoting $\mathcal{H}_{i,t,k}$ as the set of all alignments that pass through arrow $P_{t,i}(k)$, the second partial derivative term in Eqn. 3.41 can be written as:

$$\frac{\partial \sum_H P(H|\mathbf{X})}{\partial P_{t,i}(k)} = \frac{\partial \sum_{H \in \mathcal{H}_{i,t,k}} P(H|\mathbf{X})}{\partial P_{t,i}(k)} \tag{3.42}$$

Then, define forward variable $\alpha_{i,t}$ as the sum of probabilities of paths leading to position (t,i), and $\beta_{i,t}$ as the sum of probabilities of paths starting from position (t,i) to the top-right. The $\sum_{H \in \mathcal{H}_{i,t,k}} P(H|\mathbf{X})$ can be written as:

$$\sum_{H \in \mathcal{H}_{i,t,k}} P(H|\mathbf{X}) = \begin{cases} \alpha_{i,t} P_{t,i}(k) \beta_{i,t+1}, k = \varnothing \\ \alpha_{i,t} P_{t,i}(k) \beta_{i+1,t}, k \neq \varnothing \end{cases} \tag{3.43}$$

Note that neither $\alpha_{i,t}$, $\beta_{i+1,t}$ nor $\beta_{i,t+1}$ contains a factor of $P_{t,i}(k)$. Hence the derivative in Eqn. 3.42 can be written as.

$$\frac{\partial \sum_H P(H|\mathbf{X})}{\partial P_{t,i}(k)} = \begin{cases} \alpha_{i,t} \beta_{i,t+1}, k = \varnothing \\ \alpha_{i,t} \beta_{i+1,t}, k \neq \varnothing \end{cases} \tag{3.44}$$

This results in the closed-form solution of the gradient to each element in the vector after Softmax using the forward-backward algorithm, and hence error back-propagation can be performed in a standard way for the rest of the network.

While RNN-T gives good performance on small footprint on-device models (He et al., 2019), it can also perform well with large-scale models trained with a very large amount of data. Training for RNN-T with a Conformer/Transformer encoder also uses the same optimisation and learning rate scheduling methods as the Conformer/Transformer AED. However, when using a bi-directional encoder, such as Conformer, limited future context can be used if working in streaming mode. This results in a trade-off between the latency and future context that needs to be balanced.

3.3.3 RNN-T Decoding

As for the AED approach, RNN-T decoding aims to find the most likely output token sequence

$$Y^* = \arg\max_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X}) = \arg\max_{\mathbf{Y}} \sum_{H \in \text{Align}(\mathbf{Y})} P(H|\mathbf{X}) \tag{3.45}$$

As this expression is intractable, it is usually approximated by finding the most likely alignment H, as shown below.

$$H^* = \arg\max_{H} P(H|\mathbf{X}) = \arg\max_{H} \prod_{H_i} P(H_i|H_{1:i-1}, \mathbf{X}) \tag{3.46}$$

In this formulation, the best alignment sequence can be found using a beam search algorithm that is modified for the RNN-T. The beam search algorithm for RNN-T iterates over each encoder step whereas for AED it is synchronous with the decoder. At each encoder step, the RNN-T model will continue to generate new tokens until B probable sequences ending with $\varnothing$ are obtained, where B is the beam size. Details are shown in Algorithm 2.

Algorithm 2 Beam Search Decoding for RNN-T

Initialise: $B = \{\}$, Beam size W
for *t from 1 to T* **do**
  Merge alignments in B corresponding to the same $\mathbf{Y}$,
  Set $A = B$, $B = \{\}$,
  ▷ A stores beams that need to be expanded by a further symbol
  ▷ B stores beams that ends with $\varnothing$ and are ready to move to $t+1$
  ▷ Keep expanding the best beams in A and storing those expanded with ϕ into B, until there is no change to the top W beam in B
  while *B contain fewer than W elements more probable than most probable in A* **do**
    Take the best current beam b=($y_{1:i-1}$, $P(y_{1:i-1}|\mathbf{X})$) from A
    Compute distribution $P_{t,i}(k)$
    Add $P_{t,i}(\varnothing)$ to $P(y_{1:i-1}|\mathbf{X})$ to make b^*, save b^* to B
    for *Most probable W non-blank symbols* **do**
      Add each $P_{t,i}(k)$ to $P(y_{1:i-1}|\mathbf{X})$ and append k to $y_{1:i-1}$ to form b^*.
      Add b^* to A
    end
  end
  Keep only the top W beams in B
end
Return the best beam in B

The most computationally intensive part of Algorithm 2 is the “while" loop that continues to generate multiple tokens until all W most probable beams end with $\varnothing$. As an effective

simplification, one-step-constrained (OSC) decoding (Kim et al., 2020) can also be used where it limits one non-blank token to be output at each encoder step.

3.3.4 MWE Training for RNN-T

MWE training can also be used for RNN-T, where the same N-best list approximation as for AED can be used Weng et al. (2020), Guo et al. (2020). The N-best list can be obtained by running a beam search for each utterance and getting the N-best alignment hypothesis and transforming those alignments into word-level sequences. As for AED, the total loss is also the sum of the CE loss and RNN-T loss scaled by μ. However, as the beam search algorithm for RNN-T has a much higher time complexity than AED due to its frame synchronicity, the training speed could be very slow. To mitigate this problem, Guo et al. (2020) improved MWE training by relating the gradient calculation for each alignment of a hypothesis in the N-best list to the original RNN-T loss function, which enabled offline decoding of N-best lists when abundant CPU resource was available. Moreover, (Sun et al., 2022a) adopted OSC decoding to find the N-best list which can be efficiently implemented on a GPU which will be discussed in Chapter 4 in more detail.

3.4 Neural LMs

Instead of relying on frequency count statistics from a text corpus to estimate the LM, an alternative approach is to train DNNs for word prediction, which can be used as LMs (Bengio et al., 2003). The advantage of this approach is that the Softmax output layer over the vocabulary does not assign zero probability to any predicted word, and non-linear functions can extract word embeddings that contain semantic and contextual information. The MLP and CNN can both serve as LMs, where the input is the concatenation of embeddings corresponding to the previous n words, as in the standard n-gram LM. To model long-term dependencies, Recurrent Neural Network LMs (RNNLMs) are commonly employed, where instead of directly sending word embeddings, RNNs (or LSTMs) encode the previous word embeddings into a single compact representation (Mikolov et al., 2010, Oparin et al., 2012, Sundermeyer et al., 2012, Chen et al., 2014, 2016). More recently, the Transformer has emerged as a powerful language model that can capture longer-term dependencies (Radford et al., 2018, Al-Rfou et al., 2019, Wang et al., 2019, Yang et al., 2019, Irie et al., 2019). Furthermore, Transformer-based LMs have enabled large-scale training with extensive data, resulting in language foundation models that serve as initialisation points for achieving better performance on related language tasks (Radford et al., 2019, Brown et al., 2020).

NNLMs can be used in both SCM and end-to-end ASR systems. In an SCM system, due to the long-term dependency, second-pass decoding is usually needed where the LM serves to re-score to re-rank multiple hypotheses. In end-to-end models, LM log probabilities (or scores) can be interpolated with the model output log probabilities in the log-linear space to achieve joint decoding, known as shallow fusion (SF) (Gulcehre et al., 2015). The LM can also be used as a part of the input to an AED model via various neural interfaces (Gulcehre et al., 2015, Sriram et al., 2017, Shan et al., 2019). Note that the RNN AED model without an encoder and attention is essentially a word piece RNNLM, and the same for the Transformer AED. The predictor in the RNN-T also has a similar functionality as an RNNLM. Therefore, all end-to-end ASR models mentioned in this thesis learn an implicit internal LM during training that might degrade the system's performance when tested on data that does not share the same language patterns as the training data.

This section introduces the RNNLM and Transformer LM, and their application in both SCM and end-to-end ASR systems. LM discounting methods which aim at mitigating the aforementioned language pattern mismatch issue will also be introduced.

3.4.1 RNNLM

To model long-term dependency, the RNNLM has been developed and shown its ability to incorporate information from a much longer history than n-gram LMs. The RNNLM was first proposed in (Mikolov et al., 2010) where a single layer RNN was used to show its superiority over a 5-gram LM with KN-smoothing. The work in (Sundermeyer et al., 2012) extended RNNLMs to use the LSTM architecture that enriched the history representation. Meanwhile, it was discovered in (Oparin et al., 2012) that further improvements could be achieved by linearly interpolating the RNNLM with n-gram LMs. Moreover, the work in (Chen et al., 2014) enabled efficient GPU training of RNNLMs with sentence "bunches". The toolkit released by (Chen et al., 2016), in addition, supported faster training algorithms and loss function (e.g. noise contrastive training (Gutmann and Hyvärinen, 2010)) which allow RNNLMs to be trained efficiently on large corpora, and hence improve the performance evaluated by PPL and also WER on ASR tasks. The basic RNNLM structure is illustrated in Fig. 3.8 (left). Each word w_t in the input sequence will be assigned a unique index, and also a fixed-length vector representation $\mathbf{w}_t$, known as the *word embedding* by looking up an embedding matrix using the index. The RNNLM models the conditional probability distribution using the complete back history which is represented with a hidden state vector $\mathbf{h}_{t-1}$. The output which is the predicted distribution of the next word over the entire vocabulary is derived from a Softmax output layer. By using the RNN structure, the prediction is not

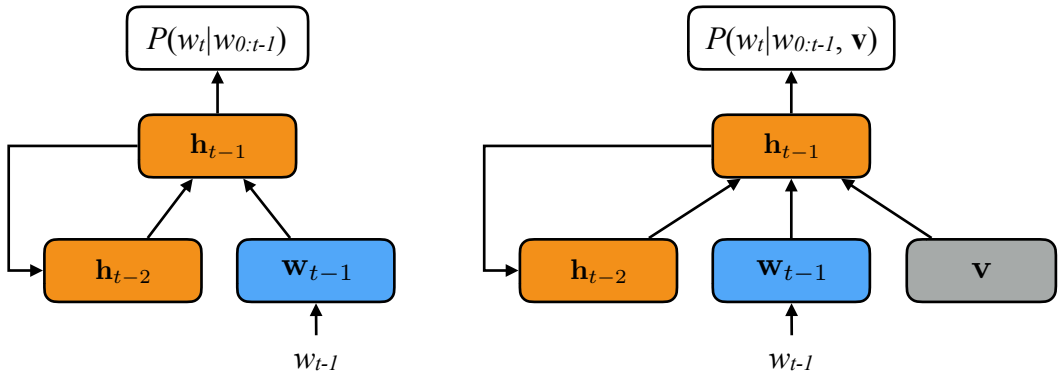


Fig. 3.8 RNNLM. Left: vanilla single layer RNNLM structure for the word sequence $w_{1:N}$. Right: RNNLM adaptation with auxiliary input feature.

limited to a fixed number of preceding words. Moreover, the LSTM structure may be used to replace the basic RNN to achieve better performance.

Since surrounding utterances may also provide useful information for the recognition of the current utterance, whereas it is difficult to incorporate cross-utterance information directly into an n-gram-based SCM, cross-utterance LMs are designed to include cross-utterance information from the language aspect. A typical simple design is to extend the representation of $\mathbf{h}_{i-2}$ to cover previous utterances. However, limited by the single compact vector representation, it often yields a suboptimal performance. Another way is to incorporate cross-utterance information, or other global information using LM adaptation approaches, as shown in Fig. 3.8 (right). This extra information is encoded in a fixed length vector as an auxiliary feature input. This includes topic information Mikolov and Zweig (2012), genre information Chen et al. (2015b) and surrounding utterance information encoded using a DNN (Jaech and Ostendorf, 2018) or an attention mechanism (Sun et al., 2020b). When incorporating long-range cross-utterance information, Transformer LM is often a better choice than RNNLM.

3.4.2 Transformer LM

The Transformer structure can also be used in LMs which are usually built on the Transformer decoder part as shown in Fig. 3.9 without cross-attention. For auto-regressive decomposition, the lower-triangular mask is used such that information from the current and future ground-truth words will not flow into the probability prediction of the current word. Early work in this area included Radford et al. (2018, 2019), Al-Rfou et al. (2019) who used transformer decoders and achieved improved performance in terms of PPL. Later research Wang et al.

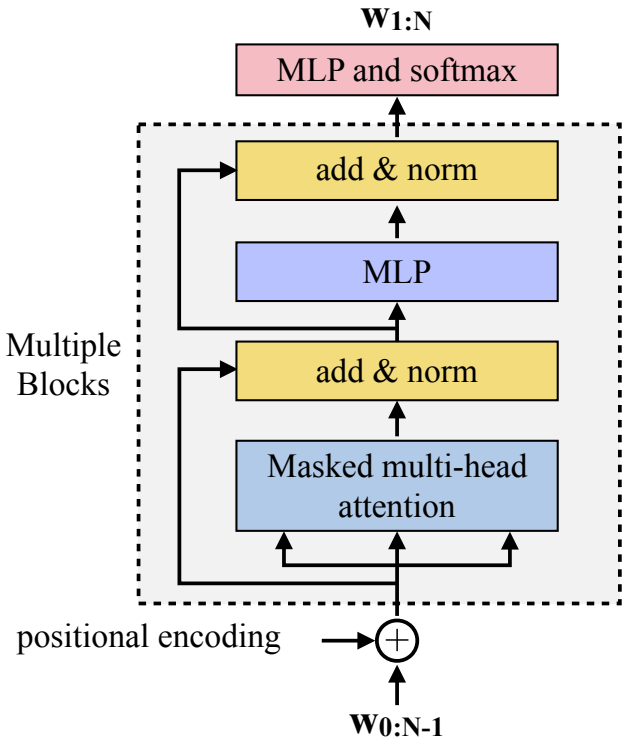


Fig. 3.9 Transformer LM using the Transformer decoder structure without cross-attention.

(2019), Yang et al. (2019) improved the performance further by an architecture search and the use of recurrent connections between transformer blocks (Sun et al., 2021b). Work in Irie et al. (2019) showed the effectiveness of transformer LMs in ASR.

3.4.3 LM Fusion in End-to-end Trainable ASR Models

Despite the implicit internal LM in AED and RNN-T, integration of an external LM is often found useful empirically, especially when the external LM brings abundant or domain-specific knowledge that is not covered in the training set.

3.4.3.1 Shallow Fusion

LM shallow fusion (SF) integrates the LM log probability via log-linear interpolation. When used in SF, the LM usually operates at the word piece level with the same word piece vocabulary as the end-to-end ASR model. Denoting the sequence of word pieces output from the ASR model as $\mathbf{Y}$, the operation of SF is given in Eqn. 3.47.

$$\hat{\mathbf{Y}} = \arg\max_{\mathbf{Y}} \{\log P(\mathbf{Y}|\mathbf{X}) + \alpha P_{\text{LM}}(\mathbf{Y})\} \tag{3.47}$$

where α is an interpolation hyper-parameter that needs to be tuned on the validation set. When applied to AED, SF could be performed for each output token at each decoding step as the decoding is synchronous with LM output. When applied to RNN-T, the interpolation happens whenever there is a new word piece (non-blank) output from RNN-T as the LM does not provide scores for $\varnothing$. Note that LM SF changes the hypotheses kept at each beam search step, and hence the final result hypotheses are in general different from not using SF.

For SF to be effective, the external LM is usually trained on a larger corpus or on the target domain for cross-domain evaluation. However, as the λ is often a fixed value that is applied to all utterances during decoding, it lacks the flexibility of dealing with different utterances under a different context, and other earlier fusion methods have been proposed to mitigate this problem by performing model-level fusion. Note that this issue with SF also exists for SF-based contextual biasing, which is analysed in Chapter 4.

3.4.3.2 Other Fusion Methods

Two typical alternative fusion approaches are introduced here, namely deep fusion and "cold" fusion, that are most commonly seen in AED models, as shown in Fig. 3.10. Both

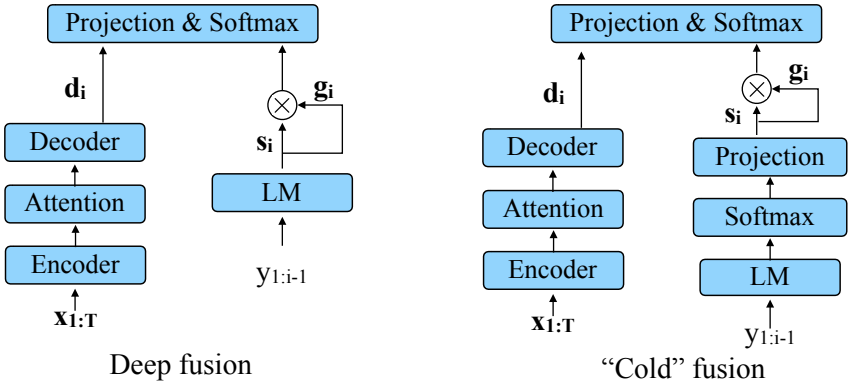


Fig. 3.10 LM deep fusion (left) and "cold" fusion (right) for AED model.

fusion methods use the self-gating mechanism to extract useful information from the vector representation $\mathbf{s}_i$. In deep fusion, the LM hidden state is used to derive $\mathbf{s}_i$, whereas in "cold fusion", the LM output distribution is used. The gated vector from the LM is then concatenated with the decoder output state, which is similar to LM adaptation in Sec. 3.4.1.

3.4.3.3 Internal LM Discounting

Since each output of an AED model depends on the full history of previous outputs, the ASR model has implicitly learned an internal LM. Although internal LM scores can not

be computed explicitly, various internal LM estimation and discounting algorithms have been developed to minimise the internal LM effect on end-to-end ASR models, in particular when applying the model to a test set in a different domain from the training data. In (McDermott et al., 2019), a density ratio method was introduced to estimate the score from a source-domain external LM that is to be subtracted from the target-domain LM score. Later, the HAT model was proposed to preserve the modularity of a hybrid system and allowed internal LM scores to be estimated and discounted (Variani et al., 2020). An internal LM estimation method was proposed in (Meng et al., 2021b) to estimate the source-domain LM score directly from the end-to-end ASR model. This method was further improved by performing internal LM training (Meng et al., 2021a) in order to better estimate the internal LM score. Moreover, (Zhang et al., 2022) proposed regularising the internal LM in RNN-T training to avoid over-fitting to text priors.

The density ratio-based approach (McDermott et al., 2019) which directly relates to the biasing-driven LM discounting method introduced in Chapter 4 is explained in detail here. After decomposing the probability of each possible sequence $\mathbf{Y}$ into a token-level sequence, $P(y_i)$, the probability of each output token y_i after SF, can be written as

$$P(y_i) = P^{\text{mdl}}(y_i|\mathbf{X})P^{\text{tgt}}(y_i)^{\alpha}, \tag{3.48}$$

where $P^{\text{mdl}}(y_i|\mathbf{X})$ is the model output probability and $P^{\text{tgt}}(y_i)$ is the external LM probability for y_i, and the conditions on acoustic and history information were omitted for clarity. The density ratio method provides a Bayes' rule-grounded way to reduce the effect of the internal LM in the end-to-end system especially when there is a difference between the source and target domain data. That is,

$$P(y_i) = P^{\text{mdl}}(y_i|\mathbf{X})\frac{P^{\text{tgt}}(y_i)^{\alpha}}{P^{\text{src}}(y_i)^{\beta}}, \tag{3.49}$$

where $P^{\text{src}}(\mathbf{Y})$ refers to the probability of the output sequence predicted by an LM trained on the source domain. The factors α and β are hyper-parameters. In this way, the probabilities of commonly seen text patterns in the source domain are penalised, whereas those of text patterns specific to the target domain are boosted.

3.5 End-to-end Spoken Language Understanding

The aim of Spoken Language Understanding (SLU) (Tur and Mori, 2011) is to extract important information from speech input to interpret the meaning of the speech. In contrast

to ASR which transcribes “what they said", SLU aims at extracting “what they meant". SLU is a core component that connects ASR and desired response generation in a task-oriented dialogue system, which aims to capture the semantics of user queries. A typical SLU system performs intent detection and slot-filling tasks. Intent detection is the task that determines the user intent in the input utterance and slot filling is the task that aims to fill predefined slots with the correct values. Two examples from SLU tasks are shown in Fig. 3.11.

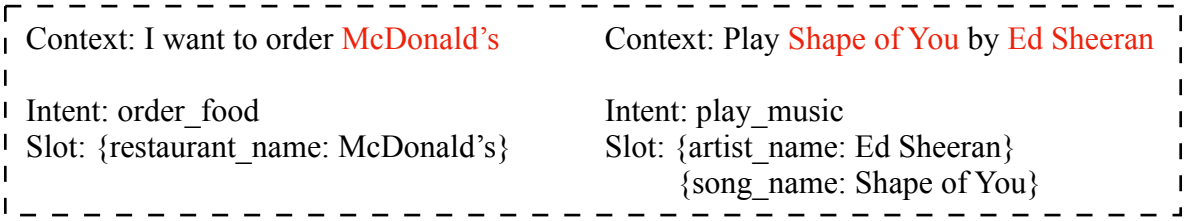


Fig. 3.11 Two examples showing SLU tasks where the text marked in red is the expected string value in the utterance for slot filling.

In contrast to natural language understanding (NLU), SLU takes speech as input, which means there are challenges with dealing with noisy transcriptions from an ASR system. A commonly adopted approach is to construct a pipeline system, where a standalone ASR system is used to transcribe speech into text which may contain errors and a standalone NLU system is used to extract intent and slot information from text. Such a pipeline system can not be jointly optimised as the gradient is unable to propagate through transcriptions. However, by leveraging pre-trained foundation models and advanced end-to-end trainable ASR technologies, pipeline systems usually achieve reasonably good performance and may outperform end-to-end trainable systems in certain scenarios (Desot et al., 2022, Lichouri et al., 2022).

On the other hand, as for end-to-end trainable ASR, end-to-end trainable SLU systems still possess an inherent advantage in the simplified pipeline and a jointly optimised objective (Serdyuk et al., 2018, Haghani et al., 2018, Radfar et al., 2020). Therefore, designing more effective end-to-end SLU systems that carry forward these advantages is the main research focus in SLU. This section introduces commonly used SLU systems, with a particular focus on end-to-end system designs. It also discusses how pre-trained foundation models can benefit SLU systems, followed by commonly used evaluation metrics for SLU.

3.5.1 SLU System Design

This section introduces three different SLU system designs that are frequently used in recent research. This includes the pipeline system, the end-to-end trainable system with a

differentiable neural interface (Rao et al., 2021, Raju et al., 2022, Seo et al., 2022), and the end-to-end trainable system without an explicit ASR module (Radfar et al., 2020, Wang et al., 2021). These systems are illustrated in Fig. 3.12.

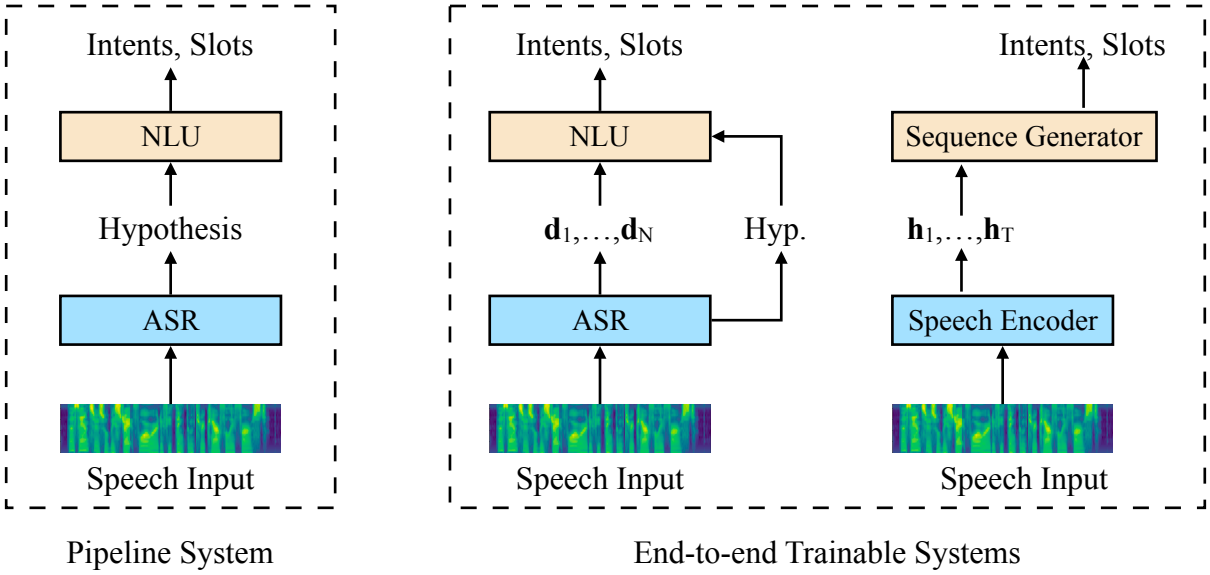


Fig. 3.12 Illustration of three types of SLU models commonly used in recent research.

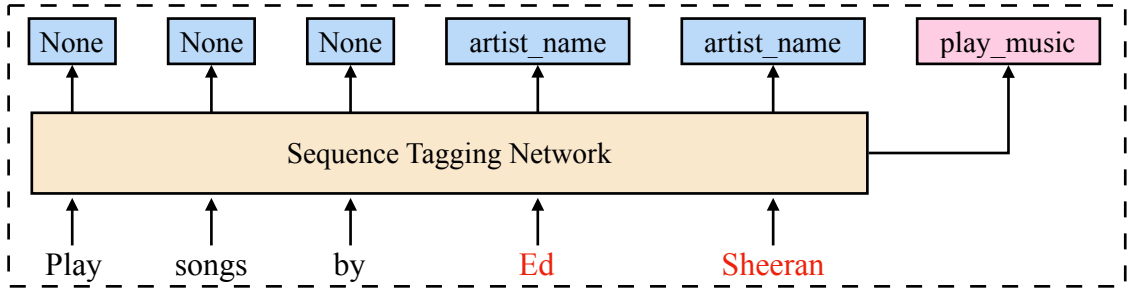
In the pipeline system, 1-best ASR hypotheses are obtained from an ASR sub-system, and the NLU module, which is usually trained on the oracle transcriptions, is used to extract intents and slots from the hypotheses. To mitigate any mismatch and increase the robustness of the NLU module to ASR errors, N-best hypotheses may be used instead of only 1-best for pipeline systems (Ganesan et al., 2021).

Two typical end-to-end trainable SLU systems are shown on the right of Fig. 3.12. The left one adopts a differentiable neural interface that takes the sequence of vector representations from an end-to-end trainable ASR system, e.g. hidden states of the final decoder layer (Rao et al., 2021) or the output distribution over word pieces (Seo et al., 2022), and the NLU module takes these vectors as input and predicts the targets. Gradient back-propagation is allowed through this neural interface, hence achieving end-to-end training. Moreover, such systems have an explicit ASR module and hence explicit 1-best hypotheses which can be used to bootstrap the performance of the NLU module in a deliberation manner (Raju et al., 2022), especially for those that use pre-trained foundation models.

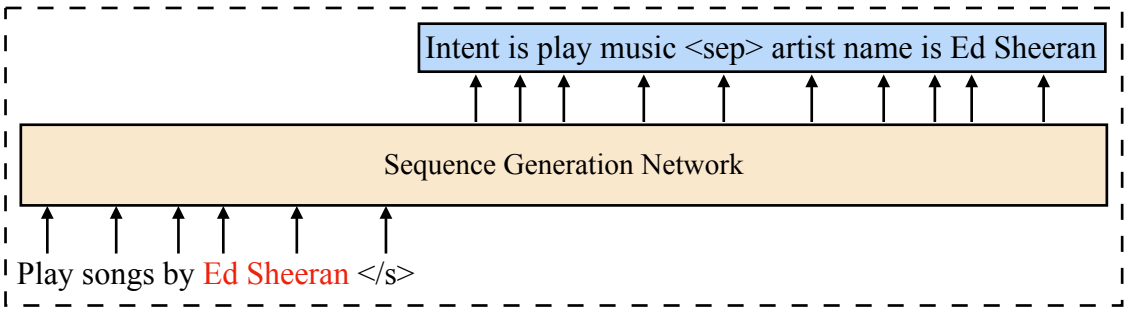
The other possible end-to-end system design directly extracts semantic information from speech, without explicit ASR and NLU modules. This type of system usually adopts a powerful speech encoder that transforms speech into a sequence of hidden states $\mathbf{h}_1,\ldots,\mathbf{h}_T$ that is synchronous with the frames. Then, a sequence generation module, such as an LSTM

or a Transformer LM, is used to generate text sequences containing slot and intent information based on the speech encoder output. These systems usually heavily rely on powerful speech representations, such as pre-trained speech foundation models. Also, there are no explicit ASR hypotheses bootstrapping the performance.

This thesis focuses on the end-to-end trainable SLU system with neural interfaces. After taking the decoder hidden states and ASR hypotheses, there are mainly two ways for the NLU module to generate intents and slot values, as shown in Fig. 3.13.



Sequence Tagging System



Sequence Generation System

Fig. 3.13 Two types of NLU modules to perform intent detection and slot filling.

The top part of Fig. 3.13 is the sequence tagging system which generates one slot label for each word or sub-word unit in the input sequence. This assumes that one word only belongs to one slot type. On the other hand, NLU can be formulated as a sequence generation task as shown at the bottom of Fig. 3.13. This system takes the utterance as the prompt to a generation network and performs slot filling either via question-answering (QA) (Shah et al., 2019, Budzianowski and Vulić, 2019) by asking the value of each slot type or as a pure language generation that generates the slot type followed by its value in natural language (Mehri and Eskenazi, 2021). In particular, the QA formulation is able to handle unseen slot types when applied in a cross-domain scenario, whereas the natural language generation formulation and the entity tagging formulation are in theory only able to cope with slot types

in the training data unless specific mechanisms are used (Sun et al., 2023b). Note that the formulation of the intent detection here is not critical as the generation-based formulation with fixed syntax for a finite set of intents is essentially the same as classification.

3.5.2 End-to-end SLU with Pre-trained Models

End-to-end SLU systems significantly benefit from pre-trained speech and language foundation models. The pre-trained speech foundation model acting as the speech encoder achieves better ASR performance and provides more powerful representations that can be used by the NLU or sequence generator (Wang et al., 2021, Peng et al., 2022). On the NLU side, BERT-like (Devlin et al., 2019, Chen et al., 2019a, Liu et al., 2019a, Hosseini-Asl et al., 2020) models can be used to achieve effective sequence tagging while GPT series (Radford et al., 2019, Brown et al., 2020) and T5 (Raffel et al., 2019) causal foundation models are able to achieve superior performance on sequence generation (Namazifar et al., 2020, Liu et al., 2022, Hosseini-Asl et al., 2020, Madotto et al., 2020).

As manual fine-grained annotation for slot labels is expensive, time-consuming, and usually requires domain expertise, foundation models are also used to cope with the increasing demands on the performance of SLU systems for *few-shot* or even *zero-shot* learning setups (Fuisz et al., 2022, Henderson and Vulić, 2021, Du et al., 2021). Such work often leverages the few-shot learning ability of foundation LMs (Brown et al., 2020) and adopts the QA or text-generation formulation for slot filling. The combination of end-to-end trainable SLU and foundation LMs will be further reviewed and discussed in Chapter 6.

3.5.3 SLU Evaluation

Commonly adopted evaluation metrics for SLU systems use percentage accuracy for intent detection, and micro-averaging of the entity F1 scores for slot filling (Bastianelli et al., 2020). The intent accuracy is the ratio between the number of utterances with correctly classified intents and the total number of utterances in the test set. The micro entity F1 score computes the harmonic mean between *precision* and *recall* across the test set. Precision and recall values can be computed by finding the True Positives (TP), False Negatives (FN), and False Positives (FP) that occurred in the test set, as shown in Eqn. 3.50.

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}} \tag{3.50}$$

For each utterance, all possible slot types will either have an entity value if mentioned or a “none" value if not mentioned. TP is defined as the number of slots with entity values that are

filled with the correct entities. FP is the number of slots that are filled in with an incorrect entity value. FN is the number of slots with an entity value but filled with an incorrect value or “none". Note that a slot with an entity value but filled in with a wrong entity value is counted in both FP and FN, as it misses the correct entity and also gives a wrong prediction. Therefore, precision is a measure that indicates how many of the predictions made by the model are true slot entities, while recall indicates how many true slot entities have been covered by the model prediction. The F1 score is then computed as:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.51}$$

In addition to micro-averaging, domain macro-averaging may also be used to evaluate the performance which first computes micro F1 within each domain (e.g. computes for restaurant-related slots and hotel-related slots separately), and then averages across all domains in the test set. This is often used when clear domain definitions are given.

Algorithm 3 Distance-based F1 score for utterance s

Input: $\mathcal{E}$, $\hat{\mathcal{E}}$ ▷ gold and predicted entities respectively
Initialise: TP, FP, FN $\leftarrow 0$
Initialise: $\mathcal{L}_s \leftarrow$ set of gold slot types in s
Define: dist$(\cdot,\cdot) \leftarrow$ edit distance measure **for** $\hat{e}$ *in* $\hat{\mathcal{E}}$ **do**
  if $\hat{e}$.*slot_type* $\in \mathcal{L}_s$ **then**
    $\mathcal{P}_l \leftarrow \{(e,\hat{e}) | \forall e \in \mathcal{E}, e.\text{label} = \hat{e}.\text{label}\}$ **if** $|\mathcal{P}_l| > 0$ **then**
      $(e,\hat{e}) \leftarrow \arg\min_{(e,\hat{e})\in\mathcal{P}_l} \text{dist}(e,\hat{e})$
      TP $+= 1$
      FP $+= \text{dist}(e.\text{slot\_value}, \hat{e}.\text{slot\_value})$
      FN $+= \text{dist}(e.\text{slot\_value}, \hat{e}.\text{slot\_value})$
      $\mathcal{E}$.remove(e), $\hat{\mathcal{E}}$.remove($\hat{e}$)
    else
      FP $+= 1$, $\hat{\mathcal{E}}$.remove($\hat{e}$)
  else
    FP $+= 1$, $\hat{\mathcal{E}}$.remove($\hat{e}$)
end
for e *in* $\mathcal{E}$ **do**
  FN $+= 1$, $\mathcal{E}$.remove(e)
end
Return TP, FP, FN

The main problem with the entity F1 score is that it does not give partial credit to the model prediction. For example, if the model only predicts “Sheeran" for the target “Ed Sheeran", this is entirely counted as a False positive. To give partial credit to the model prediction, SLU-F1 is proposed in (Bastianelli et al., 2020) as the unweighted average

of distance-based F1 scores at word and character levels. The calculation of each of the distance-based F1 scores is shown in Algorithm 3. The key feature of this algorithm is that when the reference and the hypothesis have different values for a specific slot type, it first acknowledges the correctness of having that slot type extracted by increment 1 in TP, and then increase FP and FN by the edit-distance. The use of SLU-F1 ensures that a non-zero F1 score will be given even if the slot value is incorrect, as long as there is an overlap in the text. SLU-F1 is used in all slot-filling evaluations in Chapter 6.

3.6 Summary

Based on the building blocks introduced in Chapter 2, this chapter introduced end-to-end trainable ASR and SLU systems as two main tasks considered in this thesis. While both tasks heavily rely on DNNs that are trained with specific training data and static model parameters, they often exhibit degraded performance on infrequent or unseen words or entities. The following chapters aim at addressing this problem via contextual knowledge integration, using the proposed tree-constrained pointer generator (TCPGen) component.

Chapter 4

Tree-Constrained Pointer Generator for End-to-End Trainable ASR

4.1 Introduction

End-to-end ASR systems often suffer from high recognition errors on words that are rare or not included in the training set. This is known as the long-tail word problem. Contextual biasing integrates external contextual knowledge into ASR systems at test-time, which plays an increasingly important role in addressing the long-tail word problem in many applications (Williams et al., 2018, Chen et al., 2019b, Zhao et al., 2019, Pundak et al., 2018, Chen et al., 2019c, Alon et al., 2019, Le et al., 2021b,a, Garg et al., 2020, Kang and Zhou, 2020, Huang et al., 2020, Liu et al., 2020a). Contextual knowledge is often represented as a list (referred to as a *biasing list*) of words or phrases (referred to as *biasing words*) that are likely to appear in a given context. There exist a variety of resources from which biasing lists can be extracted or organised, such as a user's contact book or playlist, recently visited websites and the ontology of a dialogue system *etc*. Although biasing words occur infrequently and hence may only have a small impact on the overall word error rate (WER), they are mostly content words, such as nouns or proper nouns, and are thus particularly important to downstream tasks and highly valuable. A word is more likely to be recognised if it is incorporated in the biasing list, which makes contextual biasing critical to the correct recognition of those rare content words in an end-to-end ASR system.

As end-to-end ASR systems are often designed to incorporate all of the required knowledge into a single static model, it is particularly challenging for such systems to integrate contextual knowledge specific to a dynamic test-time context. Therefore, dedicated contextual biasing approaches have been proposed, including shallow fusion (SF) with a special

weighted finite-state transducer (WFST) or a language model (LM) adapted for the contextual knowledge (Williams et al., 2018, Chen et al., 2019b, Zhao et al., 2019, Liu et al., 2020a, Kang and Zhou, 2020, Huang et al., 2020), attention-based deep context approaches (Pundak et al., 2018, Chen et al., 2019c, Alon et al., 2019), as well as a combination of both approaches (Le et al., 2021b,a). More recently, contextual biasing components with a neural shortcut that directly modifies the ASR model output distribution have been proposed (Sun et al., 2021c, Huber et al., 2021, Sun et al., 2022a,b), which can be jointly optimised along with the ASR system.

In this chapter, the tree-constrained pointer generator (TCPGen) component is introduced for end-to-end contextual speech recognition (Sun et al., 2021c, 2022a). TCPGen directly interpolates the original model distribution with an extra distribution (the TCPGen distribution) estimated from contextual knowledge, based on a dynamic interpolation weight predicted by the TCPGen component. TCPGen creates a neural shortcut between biasing lists and the final ASR output distribution which allows end-to-end training of the entire model. In contrast to the original work on pointer generators (See et al., 2017, Liu et al., 2019b, Li et al., 2020c) which rely on the attention mechanism to attend to all biasing words, TCPGen represents biasing lists as word piece-level symbolic prefix trees and only attends to the valid subset of the biasing words at each time step in decoding, and can thus handle large biasing lists with high efficiency. Furthermore, TCPGen uses word pieces instead of whole words for pointer generators, which allows the entire system to use word pieces to address the out-of-vocabulary (OOV) issue. Therefore, as not only a few frequent words but also OOV words (i.e. words that have not appeared in the training set) can exist in the TCPGen biasing lists. TCPGen is essentially a structure that achieves zero-shot learning of previously unseen words without changing the model parameters. As a result, TCPGen combines the advantages of both neural and symbolic methods and improves pointer generators for ASR applications with large biasing lists.

To further improve the effectiveness of contextual biasing with TCPGen, a minimum biasing word error (MBWE) loss is also introduced in order to directly optimise the model performance on the biasing words. By changing the risk function of the widely used minimum word error (MWE) loss (Povey and Woodland, 2002, Prabhavalkar et al., 2018, Weng et al., 2018, Wynands et al., 2022, Weng et al., 2020, Guo et al., 2020), the MBWE loss has a greater focus on minimising the expected errors of the rare and OOV words in the biasing list associated with that utterance. An efficient beam search algorithm is proposed for MBWE training in RNN-T by limiting the number of output word pieces at each encoder step to one (Kim et al., 2020). Moreover, to address the issue that end-to-end models often suffer from the internal LM effect that biases towards common words, a biasing-driven LM discounting

(BLMD) method is proposed. The BLMD method extends the density ratio-based LM discounting method (McDermott et al., 2019) to incorporate an additional discounting factor for the TCPGen distribution before interpolation.

TCPGen, as a generic component for end-to-end ASR systems, is integrated into both attention-based encoder-decoder (AED) and recurrent neural network transducer (RNN-T) models. MBWE and BLMD methods are applied to both types of end-to-end model in conjunction with TCPGen. In addition to AED and RNN-T models trained from scratch, TCPGen also provides a solution for contextual biasing on universal speech models trained on a large-scale of supervised data. This chapter also introduces the integration of TCPGen, as a distribution-level adaptation component, in Whisper (Radford et al., 2023) by proposing a dedicated training scheme without updating any Whisper parameters.

This chapter is structured as follows. Section 4.2 introduces the background knowledge about contextual biasing and pointer generators. Section 4.3 explains the TCPGen component in detail, followed by Section 4.4 and 4.5 introducing MBWE and BLMD methods respectively. The integration of TCPGen in Whisper is covered in Section 4.8. Experiments are discussed in Section 4.9 and the chapter concludes in Section 4.10.

4.2 Background

4.2.1 Contextual Biasing for End-to-end ASR

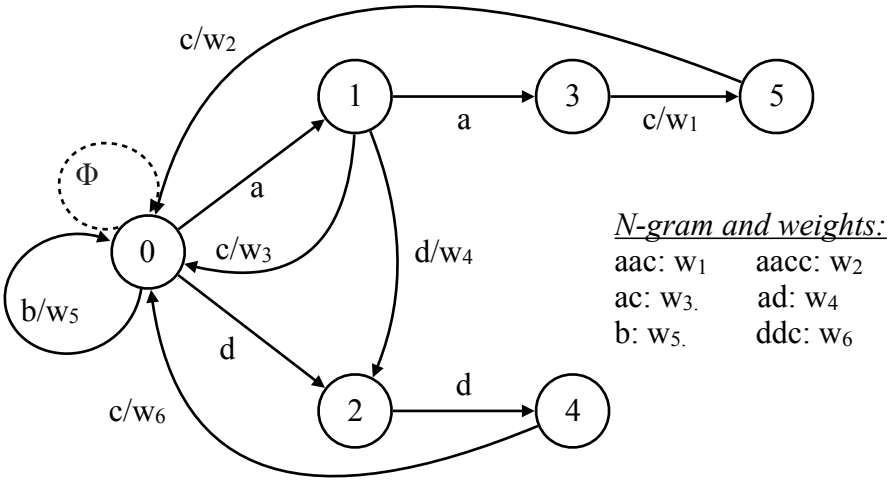
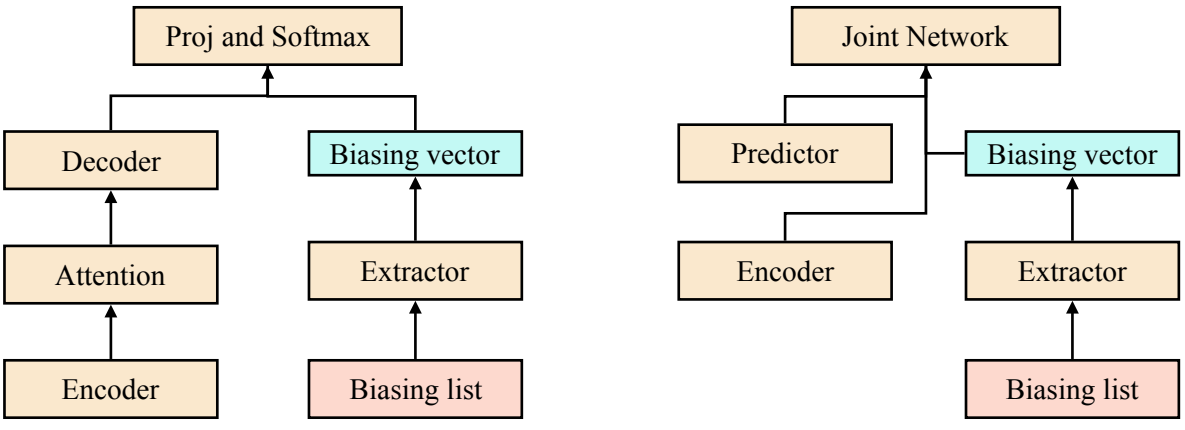


Fig. 4.1 Example of WFST for contextual biasing, where 6 biasing words are shown with their corresponding weights. Note that each state has an additional failure arc (not shown) leading back to state 0, representing no arc found with the next decoded token.

Various contextual biasing algorithms have been recently developed for end-to-end ASR. One of the major streams of research in this area focuses on representing biasing lists as an external WFST incorporated into a class-based LM via shallow fusion. The method was first introduced in Hall et al. (2015) and then subsequently used by Williams et al. (2018), Chen et al. (2019b), Zhao et al. (2019), Huang et al. (2020). A WFST is a compact representation of an n-gram LM in a given training set, as shown in the example in Fig. 4.1, where the weights are calculated from the n-gram LM. When reaching the weight-carrying arcs, the weights are added to the running score of the hypothesis via log-linear interpolation. This is analogous to the LM shallow fusion (SF) introduced in Sec. 3.4.3.1 with a specialised n-gram LM.

Despite the effective score-level modification, such methods usually rely on special context prefixes like "call" or "play" to know where to incorporate the specialised LM by interpolation, which limits their flexibility in handling more diverse grammar in natural speech. More recent methods have used class LMs (CLM) to determine the interpolation in a soft way (Bruguier et al., 2022), although the model is trained only on text which thus lacks a deep interaction with the ASR model. As WFSTs in nearly all papers are determined on industry datasets and in-house code bases without a publicly available implementation or model release, shallow fusion-based methods in this Thesis are in general referred to without direct comparison.

On the other hand, neural network-based deep biasing approaches using attention mechanisms have also been proposed in recent years. These approaches encode a biasing list into a vector that is used as a part of the input to the end-to-end ASR models Pundak et al. (2018), Chen et al. (2019c), Alon et al. (2019), where the basic framework is shown in Fig. 4.2.



Deep biasing AED Deep biasing RNN-T

Fig. 4.2 Deep biasing framework for AED and RNN-T

Deep biasing approaches are analogous to LM deep fusion, where the extractor usually adopts an attention mechanism (Pundak et al., 2018) across all biasing words. These methods address the dependency issue on syntactic prefixes in the SF methods and establish a direct neural pathway to send contextual information to the ASR model. However, deep biasing methods are more memory intensive and less effective for handling large biasing lists.

More recently, work in (Le et al., 2021b) jointly adopted the use of deep context and the SF of a WFST together in an RNN-T. It also improved the efficiency by extracting the biasing vector from only a subset of word pieces constrained by a prefix tree representing the biasing list. Work in (Le et al., 2021a) extended the prefix-tree-based method to RNN LMs which are used for SF to achieve further improvements in recognising biasing words. Moreover, while previous studies only focused on industry datasets, researchers in (Le et al., 2021a) proposed and justified a simulation of the contextual biasing task on open-source data by adding a large number of distractors to the list of biasing words in an utterance. More recently, (Sun et al., 2021c, Huber et al., 2021) have simultaneously proposed creating a neural shortcut between the biasing list and the final model output distribution.

4.2.2 Pointer Generator Mechanism

The pointer generator, initially proposed for abstractive summarisation (See et al., 2017), is a neural mechanism that allows the system to directly copy tokens from the input to the output. The structure of a typical pointer generator is shown in Fig. 4.3.

The task of summarisation involves generating a concise summary of a given input paragraph of text. This summarisation is achieved through the adoption of the AED structure, wherein an encoder, decoder and attention mechanism function in a similar manner as described in Section 3.2. The attention distribution is then rearranged into a distribution across the vocabulary by summing probabilities corresponding to the same word, while words not present in the input are assigned zero probability. Furthermore, a copy probability denoted as P_i^{gen} is computed for the i-th decoding step using decoder and attention outputs. The final distribution is the sum over the original vocabulary distribution predicted by the decoder and the transformed attention distribution, weighted by $1 - P_i^{\text{gen}}$ and P_i^{gen} respectively. Consequently, significant words in the input sequence can be utilised directly in the output without requiring the decoder to summarise or paraphrase them.

The use of a pointer generator has demonstrated its advantages, particularly concerning factual sentences and unseen words (See et al., 2017, Liu et al., 2019b). Consequently, applying such a copying mechanism to end-to-end ASR systems can be deemed natural and advantageous to address the long-tailed word problem. In this regard, the following sections

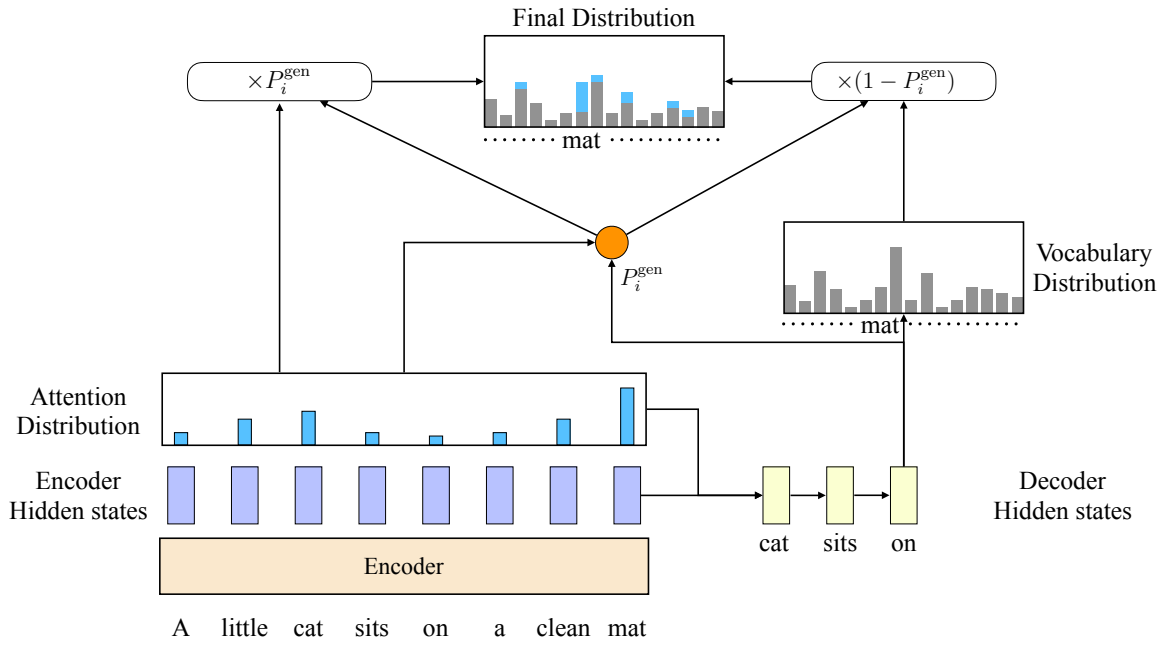


Fig. 4.3 Pointer generator mechanism for summarisation with the AED structure.

introduce TCPGen, which successfully incorporates a pointer generator into end-to-end ASR systems to facilitate contextual biasing at the sub-word level.

4.3 TCPGen

4.3.1 Model Structure

TCPGen is a neural network-based component combining symbolic prefix-tree search with a neural pointer generator for contextual biasing, which also enables end-to-end optimisation. TCPGen represents the biasing list as a word piece-level prefix tree. At each output step, TCPGen calculates a distribution over all valid word pieces constrained by the prefix tree. TCPGen also predicts a generation probability which indicates how much contextual biasing is needed at a specific output step. The final output distribution is the weighted sum of the TCPGen distribution and the original AED or RNN-T output distribution (Fig. 4.4).

The key symbolic representation of the external contextual knowledge in TCPGen is the prefix tree. For simplicity, examples and equations in this section are presented for a specific search path, which can be generalised easily to beam-search with multiple paths. In the example prefix tree with biasing words (`turner`, `vignette` and `turin`) shown in Fig. 4.5, if the previously decoded word piece is `Tur`, word pieces `in_` and `n` form the set of valid word pieces $\mathcal{Y}_i^{\text{tree}}$. Denoting $\mathbf{x}_{1:T}$ and y_i as input acoustic features and output word piece, $\mathbf{q}_i$ as

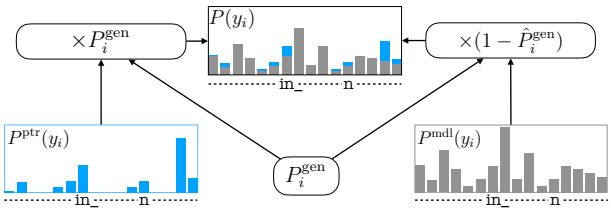


Fig. 4.4 Illustration of interpolation in TCPGen with corresponding terms in Eqn. (4.3). $P^{\text{ptr}}(y_i)$ is the TCPGen distribution. $P^{\text{mdl}}(y_i)$ is the distribution from a standard end-to-end model. $P(y_i)$ is the final output distribution. $\hat{P}_i^{\text{gen}}$ and P_i^{gen} are the scaled and unscaled generation probabilities.

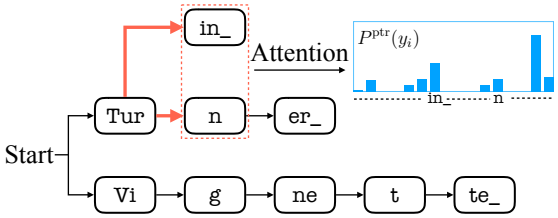


Fig. 4.5 An example of prefix tree search and attention in TCPGen. With previous output `Tur`, `in_` and `n` are two valid word pieces on which attention will be performed. A word end unit is denoted by `_`.

the query vector carrying the decoding history and acoustic information, $\mathbf{K} = [..., \mathbf{k}_j, ...]$ as the key vectors, scaled dot-product attention is performed between $\mathbf{q}_i$ and $\mathbf{K}$ to compute the TCPGen distribution P^{ptr} over all valid word pieces. The way keys and queries are computed for AED and RNN-T is introduced in Section 4.3.2 and Section 4.3.3 respectively. An output vector $\mathbf{h}_i^{\text{ptr}}$ as shown in Eqns. (4.1) and (4.2).

$$P^{\text{ptr}}(y_i|y_{1:i-1}, \mathbf{x}_{1:T}) = \text{Softmax}(\text{Mask}(\mathbf{q}_i\mathbf{K}^{\text{T}}/\sqrt{d})), \tag{4.1}$$

$$\mathbf{h}_i^{\text{ptr}} = \sum\nolimits_j P^{\text{ptr}}(y_i = j|y_{1:i-1}, \mathbf{x}_{1:T})\,\mathbf{v}_j^{\text{T}}, \tag{4.2}$$

where d is the size of $\mathbf{q}_i$ (see Vaswani et al. (2017)), Mask$(\cdot)$ sets the probabilities of word pieces that are not in $\mathcal{Y}_i^{\text{tree}}$ to zero, and $\mathbf{v}_j$ is the value vector relevant to j. For more flexibility, an *out-of-list* (OOL) token is included in $\mathcal{Y}_i^{\text{tree}}$ indicating that no suitable word piece can be found in the set of valid word pieces. To ensure that the final distribution sums to 1, the generation probability, P_i^{gen}, is scaled as $\hat{P}_i^{\text{gen}} = P_i^{\text{gen}}(1 - P^{\text{ptr}}(\text{OOL}))$, and the final output can be calculated as shown in Eqn. (4.3).

$$P(y_i) = P^{\text{mdl}}(y_i)(1 - \hat{P}_i^{\text{gen}}) + P^{\text{ptr}}(y_i)P_i^{\text{gen}}, \tag{4.3}$$

where conditions, $y_{1:i-1}, \mathbf{x}_{1:T}$, are omitted for clarity. $P^{\text{mdl}}(y_i)$ represents the output distribution from the standard end-to-end model.

The pointer generator mechanism in TCPGen operates on the same set of word pieces, which allows contextual information to be directly copied from the output. In contrast to (Huber et al., 2021), the prefix-tree structure handles a large biasing list containing thousands of words efficiently, as the attention computation only needs to be performed over a valid subset of word pieces at each decoding step. Moreover, the dynamic interpolation factor P_i^{gen} provides more flexibility than other distribution-level modifications, such as shallow fusion.

4.3.2 TCPGen in AED

The integration of TCPGen into an AED model is shown in Fig. 4.6

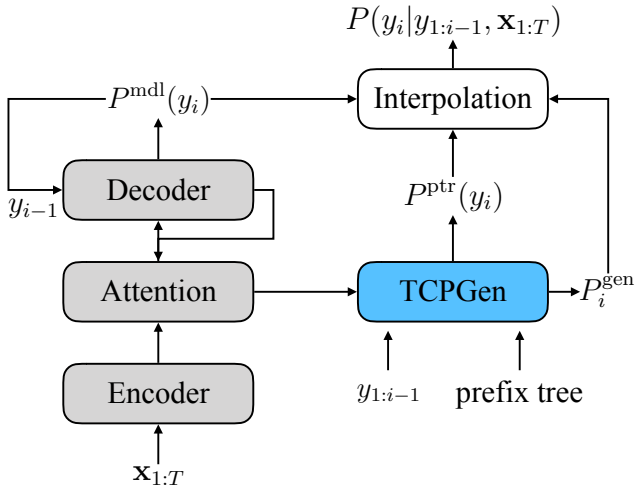


Fig. 4.6 Integration of TCPGen into AED model.

Denoting the input to the encoder, $\mathbf{x}_{1:T}$, which is encoded into $\mathbf{h}_{1:T}^{\text{enc}}$, at each decoding step i, the decoder computes $\mathbf{h}_i^{\text{dec}}$ based on encoder hidden states and preceding word piece embedding $\mathbf{y}_{i-1}$. To calculate the TCPGen distribution in an AED model, the query combines the context vector $\mathbf{c}_i$ and the previously decoded token embedding $\mathbf{y}_{i-1}$ as shown in Eqn. (4.4).

$$\mathbf{q}_i = \mathbf{W}_c^{\text{Q}}\mathbf{c}_i + \mathbf{W}_y^{\text{Q}}\mathbf{y}_{i-1}, \tag{4.4}$$

where $\mathbf{W}_c^{\text{Q}}$ and $\mathbf{W}_y^{\text{Q}}$ are parameter matrices. The keys and values are computed from the decoder word piece embedding matrix as shown in Eqn. (4.5).

$$\mathbf{k}_j = \mathbf{W}^{\text{K}}\mathbf{y}_j \qquad \mathbf{v}_j = \mathbf{W}^{\text{V}}\mathbf{y}_j, \tag{4.5}$$

where $\mathbf{y}_j$ denotes the j-th row of the embedding matrix. $\mathbf{W}^{\mathrm{K}}$ and $\mathbf{W}^{\mathrm{V}}$ are key and value parameter matrices which are shared (i.e. equal) throughout this chapter. The TCPGen distribution and the TCPGen output can be computed using Eqns. (4.1) and (4.2) respectively. The generation probability is calculated from the decoder hidden state $\mathbf{h}_i^{\mathrm{dec}}$ and the TCPGen output $\mathbf{h}_i^{\mathrm{ptr}}$, as shown in Eqn. (4.6).

$$P_i^{\mathrm{gen}} = \sigma(\mathbf{W}^{\mathrm{gen}}[\mathbf{h}_i^{\mathrm{dec}}; \mathbf{h}_i^{\mathrm{ptr}}]), \tag{4.6}$$

where $\mathbf{W}^{\mathrm{gen}}$ is a parameter matrix. The distribution of y_i can be calculated using Eqn. (4.3). In an AED model, deep biasing can be applied as shown in Eqn. (4.7).

$$P^{\mathrm{mdl}}(y_i) = \mathrm{Softmax}(\mathbf{W}^{\mathrm{O}}[\mathbf{h}_i^{\mathrm{dec}}; \mathbf{c}_i] + \mathbf{W}^{\mathrm{db}}\mathbf{h}_i^{\mathrm{db}}), \tag{4.7}$$

where the biasing vector $\mathbf{h}_i^{\mathrm{db}}$ is obtained from the sum of embeddings of all word pieces in $\mathcal{Y}_i^{\mathrm{tree}}$, similar to Le et al. (2021b).

4.3.3 TCPGen in RNN-T

The integration of TCPGen into an RNN-T model is shown in Fig. 4.7

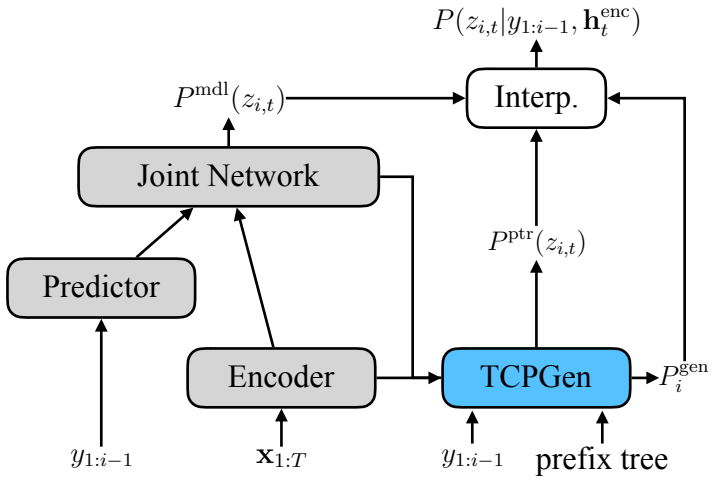


Fig. 4.7 Integration of TCPGen in an RNN-T model.

Denoting the RNN-T encoder output $\mathbf{h}_{1:T}^{\mathrm{enc}}$ and the predictor output up to token i as $\mathbf{h}_i^{\mathrm{pred}}$, the joint network predicts the output distribution $P(z_{i,t}|y_{1:i}, \mathbf{h}_t^{\mathrm{enc}})$ for each combination of i and t where $z_{i,t} \in \mathcal{Y} \cup \{\varnothing\}$ and $\mathcal{Y}$ represents the set of all word pieces. In RNN-T, the query for the TCPGen distribution is computed for each combination of i, t as shown in Eqn. (4.8).

$$\mathbf{q}_{i,t} = \mathbf{W}_c^{\mathrm{Q}}\mathbf{h}_t^{\mathrm{enc}} + \mathbf{W}_y^{\mathrm{Q}}\mathbf{y}_{i-1}, \tag{4.8}$$

where $\mathbf{y}_{i-1}$ is the word piece embedding from the predictor embedding matrix. The key and value vectors are derived from the predictor embedding matrix. The generation probability for each i,t-pair is computed using the penultimate layer output of the joint network and the TCPGen output vector

$$P_{i,t}^{\text{gen}} = \sigma(\mathbf{W}^{\text{gen}}[\mathbf{h}_{i,t}^{\text{joint}}; \mathbf{h}_{i,t}^{\text{ptr}}]) \tag{4.9}$$

As $\varnothing$ only exists in P^{mdl}, its value is directly copied to the final output distribution as shown in Eqn. (4.10)

$$\hat{P}(z_{i,t}) = \begin{cases} P^{\text{mdl}}(\varnothing), & \text{if } z_{i,t} = \varnothing \\ P(z_{i,t}), & \text{otherwise} \end{cases}, \tag{4.10}$$

where $P(z_{i,t})$ is the interpolated probability for the output token $z_{i,t}$ in Eqn. (4.3), except that $P^{\text{ptr}}(z_{i,t})$ is scaled by a factor of $1 - P^{\text{mdl}}(z_{i,t} \neq \varnothing)$ to ensure all probabilities sum to 1. Moreover, whenever TCPGen is used in RNN-T, the biasing vector, $\mathbf{h}_{i,t}^{\text{ptr}}$ is always sent to the input of the joint network as shown in Eqn. (4.11) which yielded the best results as discussed in Sun et al. (2021c).

$$\mathbf{h}_{i,t}^{\text{joint}} = \tanh(\mathbf{W}^{\text{joint}}[\mathbf{h}_{i}^{\text{pred}}; \mathbf{h}_{t}^{\text{enc}}; \mathbf{h}_{i,t}^{\text{ptr}}]), \tag{4.11}$$

As for AED, deep biasing can be applied by computing $\mathbf{h}_{i}^{\text{db}}$ as the mean of all word piece embeddings at predictor step i, and concatenating it with encoder and predictor output vectors for each combination of i,t before sending it to the joint network (Le et al., 2021a).

4.4 Minimum Biasing Word Error Training for TCPGen

This section introduces minimum biasing word error (MBWE) training. Recall that the MWE loss (see Section 3.2.5) in end-to-end ASR systems minimises the expected value of word errors across all possible output sequences of a given input. Denoting the output sequence $Y = \{y_1, ..., y_L\}$ and input sequence $X = \mathbf{x}_{1:T}$ for convenience, the MWE loss can be written as Eqn. (4.12).

$$\mathcal{L}^{\text{mwe}} = \sum_{Y} P(Y|X)\mathcal{W}^{\text{mwe}}(Y, Y^*), \tag{4.12}$$

where Y^* is the ground-truth output sequence and $\mathcal{W}^{\text{mwe}}(\cdot)$ is the risk function representing the number of word errors calculated using an edit-distance between each possible sequence Y and the ground-truth sequence Y^*. To approximate the intractable enumeration of all possible hypotheses, the N-best list can be applied (see Eqn. (3.35)).

To apply the MWE loss to contextual ASR with TCPGen which focuses on the correct recognition of biasing words, a new risk function that includes an additional biasing word

error term to the word error term is introduced as shown in Eqn. (4.13).

$$\mathcal{W}^{\text{mbwe}}(Y,Y^*) = \mu_1 \mathcal{W}^{\text{mwe}}(Y,Y^*) + \mu_2 \mathcal{W}^{\text{bias}}(Y,Y^*), \tag{4.13}$$

where $\mathcal{W}^{\text{bias}}$ is the additional biasing word error term which is the edit-distance between the sequence of biasing words in Y and the sequence of biasing words in Y^*. Scaling factors μ_1 and μ_2 control the importance of the word error term and the new biasing word error term. As a result, if $\mu_1 = 1$ and $\mu_2 = 0$, $\mathcal{W}^{\text{mbwe}}$ becomes $\mathcal{W}^{\text{mwe}}$. If $\mu_1 = 1$ and $\mu_2 = 1$, it is equivalent to giving a weight of 2 to any rare word errors in the original word error rate. A new MBWE loss function is proposed to use $\mathcal{W}^{\text{mbwe}}$ instead of $\mathcal{W}^{\text{mwe}}$. That is,

$$\begin{aligned} \mathcal{L}^{\text{mbwe}} &= \sum_{Y} P(Y|X)\mathcal{W}^{\text{mbwe}}(Y,Y^*) \\ &\approx \sum\nolimits_{Y_i \in \text{Beam}_N(X)} \hat{P}(Y_i|X)\mathcal{W}^{\text{mbwe}}(Y_i,Y^*). \end{aligned} \tag{4.14}$$

As a generic loss for end-to-end ASR systems, MBWE can be applied to the standard AED and RNN-T models, as well as other deep context models.

4.4.1 MBWE Training for AED

The MBWE loss can be applied to the AED model following a similar MWE training scheme proposed in Prabhavalkar et al. (2018) (see Section 3.2.5) which also interpolated the MBWE loss with the cross-entropy (CE) loss for better training stability, as shown in Eqn. (4.15).

$$\mathcal{L} = \mathcal{L}^{\text{mbwe}} + \mathcal{L}^{\text{ce}}, \tag{4.15}$$

where $\mathcal{L}$ is the total loss function to be minimised, $\mathcal{L}^{\text{mbwe}}$ is the proposed MBWE loss in Eqn. (4.14) and $\mathcal{L}^{\text{ce}}$ is the CE loss. As it is hard to train a randomly initialised model with the MBWE loss, which is similar to MWE, the MBWE loss is applied from the epoch when the model is reasonably trained with the CE loss, which depends on optimisation algorithms and the task. Moreover, to boost the efficiency of beam search which is the bottleneck in the time taken for training, batched beam search is implemented by parallelising the model forward computation of all beams of all utterances in the same mini-batch on a GPU.

4.4.2 MBWE Training for RNN-T

The MBWE loss can also be applied to the RNN-T model following a similar MWE training scheme proposed in Weng et al. (2020), except that the original RNN-T loss is also included

for stable training, as shown in Eqn. (4.16).

$$\mathcal{L} = \mathcal{L}^{\text{mbwe}} + \mathcal{L}^{\text{rnn-t}}, \tag{4.16}$$

where $\mathcal{L}^{\text{rnn-t}}$ represents the original RNN-T loss. Although previous work (Weng et al., 2020, Guo et al., 2020) tried to obtain N-best lists using standard beam search for RNN-T, it is infeasible to perform such a training scheme on a single GPU or with a limited number of CPUs, even with batched decoding. The major obstacle that restricts the level of parallel computation is the unknown number of word piece tokens to output at a given encoder step, as beams requiring two or more output tokens have to be handled separately. However, as the encoder output sequence is usually longer than the number of output word piece tokens, cases where two or more tokens are output at a specific encoder step should be rare. To verify this conjecture, taking a standard RNN-T model trained on the Librispeech training set and decoded on its dev set as an example, the path taken and the number of output tokens at each encoder step for each 1-best hypothesis was recorded, as shown in Table 4.1.

# word piece tokens	0	1	2+
Count	80%	18%	2%

Table 4.1 Statistics of the Number of Output word piece Tokens at Each Encoder Step for RNN-T 1-best Hypothesis on Librispeech Dev Sets. The percentage is of the Total Number of Encoder Steps.

As shown in the table, the vast majority of encoder steps output 0 or 1 word piece tokens, where 0 means a $\varnothing$ token is output. Although in the standard beam search, it is always required to compare the score with a second output token, it is often unnecessary for generating reasonably good N-best hypotheses, especially for MBWE training where a strong approximation using the N-best list has already been made. The restricted beam search results in a relative WER increase of less than 10% on LibriSpeech test sets.

Therefore, a restricted beam search which only allows no more than one output token at each encoder step is used for efficient MBWE training in RNN-T, which is similar to the one-step constrained beam search algorithm in (Kim et al., 2020) (see Section 3.3) but with all neural network computation parallelised across all samples in the mini-batch. With the restricted beam search, the model forward computation can be efficiently parallelised for all beams of all utterances in the same mini-batch on a single GPU.

4.5 Biasing-word-driven LM Discounting for TCPGen

This section introduces the biasing language model discounting (BLMD) method for TCPGen, which starts by revisiting the density ratio method for internal LM discounting (see Section 3.4.3.3). Define the source domain data as the text of the training data for the end-to-end model, and the target domain data as the data used to train an external LM such that it generates better probability estimates for the test data. Then, the recognised sequence for LM shallow fusion can be written as Eqn. (4.17).

$$Y^* = \arg\max_Y \log P^{\text{mdl}}(Y|X) + \alpha \log P^{\text{tgt}}(Y), \tag{4.17}$$

where $P^{\text{mdl}}(Y)$ is the output of the end-to-end system and $P^{\text{tgt}}(Y)$ is the probability of the output sequence predicted by an LM trained on the target domain. The interpolation factor α is a hyper-parameter to be determined. After decomposing the probability of each possible sequence Y into a token-level sequence, $P^{\text{sf}}(y_i)$, the probability of each output token y_i after SF, can be written as

$$P^{\text{sf}}(y_i) = P^{\text{mdl}}(y_i) P^{\text{tgt}}(y_i)^{\alpha}, \tag{4.18}$$

where the dependence on acoustic and history information were omitted for clarity. The density ratio method provides a Bayes' rule-grounded way to reduce the effect of the internal LM in the end-to-end system especially when there is a difference between the source and target domain data. That is,

$$P^{\text{sf}}(y_i) = P^{\text{mdl}}(y_i) \frac{P^{\text{tgt}}(y_i)^{\alpha}}{P^{\text{src}}(y_i)^{\beta}}, \tag{4.19}$$

where $P^{\text{src}}(Y)$ refers to the probability of the output sequence predicted by an LM trained on the source domain. The factors α and β are hyper-parameters. In this way, the probabilities of commonly seen text patterns in the source domain are penalised, whereas those of text patterns specific to the target domain are boosted. Therefore, the density ratio LM discounting method can also be applied to the TCPGen component to further improve performance on the biasing words that are rare in the source domain. As the final distribution comes from both the model and the TCPGen distribution which use different parameters and history

information, density ratio SF is separately performed as

$$P^{\text{sf}}(y_i) = (1 - P^{\text{gen}})P^{\text{mdl}}(y_i)\frac{P^{\text{tgt}}(y_i)^{\alpha_1}}{P^{\text{src}}(y_i)^{\beta_1}} + P^{\text{gen}}P^{\text{ptr}}(y_i)\frac{P^{\text{tgt}}(y_i)^{\alpha_2}}{P^{\text{src}}(y_i)^{\beta_2}}, \tag{4.20}$$

where P^{ptr} is the TCPGen distribution, and the same source and target LMs are used for both distributions, but with different sets of hyper-parameters α_1, β_1 and α_2, β_2. To avoid a complicated hyper-parameter search, the best set of α, β obtained from the standard end-to-end system can be directly applied to α_1, β_1, and only α_2, β_2 need to be tuned to find the best values for the TCPGen distribution.

4.6 Experimental Setup with AED and RNN-T

4.6.1 Data

The experiments in this section were performed on three different datasets, namely the LibriSpeech audiobook corpus, the augmented multi-party interaction (AMI) meeting corpus and the dialogue state tracking challenge (DSTC) 2 and 3. Each individual dataset is introduced below.

4.6.1.1 LibriSpeech

LibriSpeech contains 1000 hours of read English audiobook recordings, which are split into train, dev and test sets. The full training set is further split into train-clean-100, train-clean-360 and train-other-500 subsets where the number in each set indicates the number of hours of recordings. The set named “clean" contain only selected low-error speech that can be assumed to be clean, whereas sets with “other" may contain more challenging audio samples. Note that the train-clean-100 training set is often used on its own to either investigate low-data scenarios, or for quick experiments to determine model-related hyper-parameters before running experiments at a larger scale.

In the same way as the training set, the dev and test sets are also split into “clean" and “other" categories, resulting in four subsets, namely “dev-clean", “dev-other", “test-clean" and “test-other". In this thesis, the decoding-related hyper-parameters are determined on the combination of the dev sets, and results are usually reported on test subsets separately.

In addition to the audio training corpus, LibriSpeech is also accompanied by an LM training corpus containing 800 million word tokens with a vocabulary size of 214k. This is used to train neural LMs for shallow fusion and BLMD and also to determine biasing lists.

4.6.1.2 AMI

AMI is a meeting corpus containing recordings of four or five people discussing technical projects. It is split into train, dev and eval sets, where the train set contains 80 hours of audio and the other two contain 10 hours each. The individual headset microphone (IHM) recordings are used in the experiments.

The AMI corpus is mainly used for two purposes. First, the AMI corpus contains mainly spontaneous conversations in oral English, in contrast to the read audiobook data in LibriSpeech. It is used to demonstrate the effectiveness of the cross-domain application of TCPGen and BLMD. Second, the AMI corpus is used to demonstrate a low-data scenario by finetuning an existing generic speech model with TCPGen. To this end, 10% of the AMI training corpus containing 8 hours of speech is randomly selected to finetune the model trained on the full LibriSpeech training data.

4.6.1.3 DSTC

The Dialogue State Tracking Challenge (DSTC) data contains human-machine task-oriented dialogues where user-side input audio was used for recognition. It is included as a real-world application of TCPGen with a limited amount of audio training resources where the ontology is used to extract contextual knowledge. The ontology in a spoken dialogue system contains slot types and their corresponding possible values in entity lists, e.g. `restaurant names`: [*restaurant_A*, *restaurant_B*,...]. The DSTC track2 train and dev sets were used as the training and validation sets, and the DSTC track3 test set was used for evaluation.

4.6.2 Biasing List Selection

Biasing list selection on LibriSpeech followed the validated simulation proposed in (Le et al., 2021a), as shown in Fig. 4.8. The full rare word list containing 209k distinct words was first defined by removing the most frequent 5000 words from the LibriSpeech LM training vocabulary, which was used as the collection of all biasing words. Rare words throughout this section are defined as words belonging to the full rare word list for each corpus. More than 60% of those 209k rare words are in fact OOV words with respect to the LibriSpeech audio training set. Following the scheme in (Le et al., 2021a), biasing lists were organised by finding words that belong to the full rare word list from the reference transcription of each

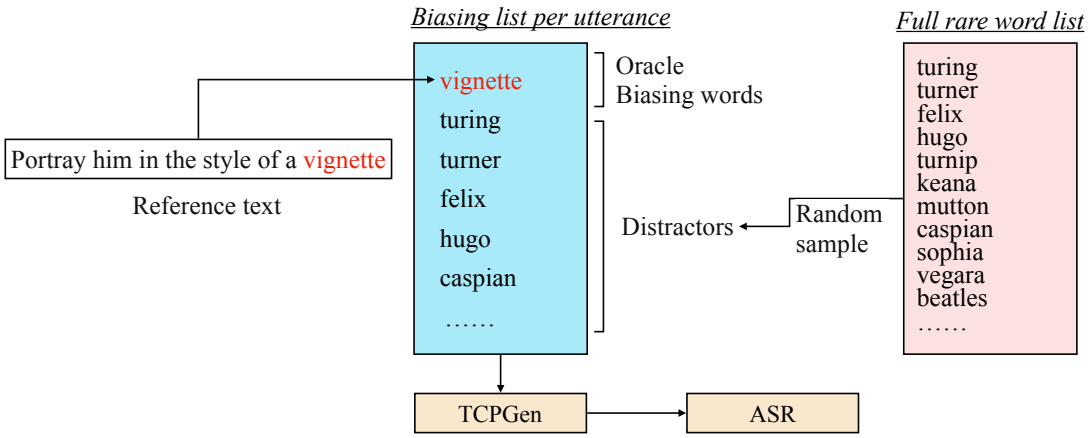


Fig. 4.8 Biasing list selection following Le et al. (2021a) as a simulation of real-world tasks on LibriSpeech data, which was also applied to AMI data.

utterance and adding a certain number of distractors to it. The same biasing list arrangement was applied to both training and testing on LibriSpeech.

For the AMI meeting data, the full rare word list was augmented with words appearing fewer than 100 times in the AMI training corpus (including OOV words that only appear in AMI dev and test sets) to form the full rare word list for AMI. Using this augmented full rare word list, contextual biasing experiments were performed by adopting the same simulation method as with LibriSpeech, which was used for both training and inference.

On the DSTC data, the biasing list arrangement by adding distractors was used for training only, where the full rare word list was augmented with words that occurred fewer than 100 times in the DSTC training data. During inference, the ontology of DSTC3 which contains lists of named entities corresponding to different types was used to form the biasing list by extracting distinct words from those named entities and removing common words with 200 or more occurrences in the training data. This biasing list contained 243 distinct words. The proportion of biasing words in each evaluation set is shown in Table 4.2.

Data	Coverage
Librispeech test-clean and test-other	10.3%
AMI eval set	5.5%
DSTC3 test	4.7%

Table 4.2 Coverage of biasing lists on the evaluation sets. Coverage is the total number of biasing word tokens divided by the total number of word tokens in each set.

4.6.3 Model Specifications

Experiments were performed for both AED and RNN-T systems. The use of an LSTM-based encoder and a Conformer encoder were investigated. The LSTM-based encoder contains 4 bi-directional LSTM layers with 1024-dim hidden states which were then projected to 2048-dim using a fully-connected layer. This encoder was also equipped with a 2-block VGG frontend (Simonyan and Zisserman, 2015) giving a 1/4 subsampling rate. The Conformer encoder contained 16 conformer blocks consisting of four 512-dim attention heads.

In addition, AED uses a single-layer 1024-dim LSTM decoder and a 4-head 1024-dim location-based attention. The RNN-T model had the same encoder, a 2-layer 1024-dim LSTM predictor and a 1024-dim joint network. TCPGen in both systems contained a 256-dim single-head attention layer and other layers were confined by system dimensions. A 1024-dim embedding matrix was used for deep biasing (denoted as DB).

A unigram-based word piece model is used for all the experiments, with 600 word piece tokens estimated from the LibriSpeech speech training corpus. For BLMD, a 2-layer 2048-dim LSTM-LM trained on the Librispeech 800 million-word text training corpus was used as the target domain LM for Librispeech experiments. Each source domain LM trained on the text of the audio training data used a single-layer 1024-dim LSTM. Each LM had the same word pieces as the corresponding ASR system. Note that all models were implemented using PyTorch (Paszke et al., 2019).

4.6.4 Training and Inference Specifications

The 80-dim FBANK features computed at a 10 ms frame rate and concatenated with 3-dim pitch features (Ghahremani et al., 2014) were used as inputs to models. SpecAugment (Park et al., 2019) with the setting $(W, F, m_F, T, p, m_T) = (40, 27, 2, 40, 1.0, 2)$ was used without any other data augmentation or speaker adaptation.

Systems TCPGen were trained with 500 distractors for train-clean-100 experiments, and 1000 distractors for full 960-hour experiments. In addition, systems with the same setup but using deep biasing were also trained. The dropping technique to randomly leave out oracle biasing words from the biasing list per utterance was applied for training but disabled for inference, with a drop rate of 30% to prevent the model from being over-confident about TCPGen or deep biasing outputs. During inference, unless specified, 1000 distractors were used to organise the biasing list for each utterance.

A beam search with a beam width of 30 was used for decoding, and a coverage penalty (Chorowski and Jaitly, 2016) of 0.01 was applied for AED. The Adadelta optimiser (Zeiler, 2012) was used to train the LSTM encoder and the Noam (Vaswani et al., 2017) optimiser

was used for the Conformer. The MBWE loss was applied after 16 epochs. The beam size for MBWE training to obtain N-best lists was 5 for all experiments.

4.6.5 Evaluation Metrics

In addition to WER, a rare word error rate (R-WER) (Le et al., 2021a) was used to evaluate the system performance on biasing words that were “rare" in the training data for that system. R-WER is defined as the total number of *error* word tokens that belong to the biasing list divided by the total number of word tokens in the test set that belong to the biasing list. Insertion errors were counted in R-WER if the inserted word belonged to the biasing list. To measure the performance on unseen words, the OOV WER was also used which was computed in the same way as R-WER, but for audio training set OOV words that appeared in the full rare word list. As WERs on Librispeech test sets were all small, for the rest of this chapter, 2 decimal places were included for WER whereas one decimal place was used for results in other corpora.

Moreover, a significance test was performed for small WER and any R-WER reductions to assess statistical significance, as the total number of rare words only comprises a small portion in the test set. Specifically, a project-by-project one-tailed sign test was performed for LibriSpeech experiments based on the “project ID" of each utterance, where the null hypothesis is that the system with TCPGen was not better than the standard ASR system. The same sign test was performed dialogue-by-dialogue for the DSTC data.

4.7 Results for TCPGen with AED and RNN-T

4.7.1 TCPGen with LSTM-based Encoder

First, experiments performed on the LibriSpeech full training data using LSTM-based encoders are shown in Table 4.3 and Table 4.4 for AED and RNN-T systems respectively. Note that SF here was used without any LM discounting methods, and the interpolation weight for SF was set to 0.3 for all experiments reported in Section 4.7.1.

First, WER and R-WER using TCPGen in AED are shown in Table 4.3. Note that the influence of TCPGen on unbiased words can be calculated by scaling the R-WER with the proportion of rare words in the test sets, and subtracting that from WER. Results with SF are also provided. AED with deep biasing and TCPGen achieved improvements in WER and R-WER on both the test-clean and the test-other sets. The best performance for WER and R-WER on both test-clean and test-other was obtained by the model with TCPGen, with 46.7% relative R-WER reduction on test-clean and 36.4% relative R-WER reduction on

Systems	test-clean		test-other	
	WER (%)	R-WER (%)	WER (%)	R-WER (%)
Standard AED	4.41	15.6	12.02	35.8
+ DB	4.03	12.0	11.62	30.2
+ TCPGen	3.69	8.3	10.91	22.7
Standard AED + SF	3.71	14.4	10.19	32.9
+ DB + SF	3.42	11.5	9.71	28.1
+ TCPGen + SF	**3.00**	**8.3**	**9.04**	**22.6**

Table 4.3 WER and R-WER on LibriSpeech test-clean and test-other set for AED with LSTM encoder trained on 960-hour data. DB uses the sum of word piece embeddings. Biasing list size is 1000 with distractors randomly selected from the full rare word list.

test-other. SF provided further WER reductions for all systems with sizeable improvements for common words and rather limited effects on biasing words. Therefore with SF, the model with TCPGen still achieved the best performance in both WER and R-WER, with a relative 42.2% R-WER reduction on test-clean and a relative 34.9% R-WER reduction on the test-other set.

In general, TCPGen achieved a bigger reduction in AED than in RNN-T, as discussed in Section 4.7.5. As a result, SF had more obvious effect in RNN-T as the room for R-WER improvements was much larger than in AED, i.e. the errors in rare words that can be improved by only using an external LM via SF have been mostly addressed by TCPGen in AED. This was not the case when the dedicated BLMD method waas used, as discussed in Section 4.7.2.

Systems	test-clean		test-other	
	WER (%)	R-WER (%)	WER (%)	R-WER (%)
Standard AED	5.53	18.7	15.31	42.9
+ DB	5.21	15.2	14.62	36.5
+ TCPGen	4.93	13.9	14.01	35.0
+ TCPGen + DB	4.82	13.3	13.91	33.5
Standard AED + SF	4.41	15.6	12.49	36.7
+ DB + SF	4.22	13.1	12.10	31.8
+ TCPGen + SF	3.99	12.2	11.61	31.0
+ TCPGen + DB + SF	**3.77**	**11.3**	**11.52**	**29.0**

Table 4.4 WER and R-WER on LibriSpeech test-clean and test-other set for RNN-T with LSTM encoder trained on 960-hour data. DB uses the sum of word piece embeddings. Biasing list size is 1000 with distractors randomly selected from the full rare word list.

WERs and R-WERs for RNN-T with biasing components trained on the 960-hour data are shown in Table 4.4. As for AED, using biasing components achieved reductions in both WER and R-WER. When TCPGen was used in conjunction with DB, further WER and R-WER reductions were achieved, with a relative 13.4% WER reduction on test-clean, and a relative 9.1% WER reduction on test-other compared to the baseline. Reductions on R-WER were not as large as for AED, with a relative 28.8% reduction on test-clean, and a relative 21.9% reduction on test-other.

Next, the experiments using MBWE training for TCPGen with LSTM encoder AED and RNN-T systems were performed on LibriSpeech train-clean-100 subset. First, LSTM-based AED systems trained on the Librispeech clean-100 set were evaluated on the test-clean set to show the performance of the TCPGen component and the proposed algorithms on a fairly small scale data setup. Results are shown in Table 4.5. Compared to the CE loss,

System	MBWE parameters	%WER	%R-WER
Baseline	$\mu_1 = 0.0, \mu_2 = 0.0$	11.59	40.1
Baseline	$\mu_1 = 1.0, \mu_2 = 0.0$	11.21	39.7
Baseline	$\mu_1 = 1.0, \mu_2 = 1.0$	11.13	38.7
TCPGen	$\mu_1 = 0.0, \mu_2 = 0.0$	9.25	22.7
TCPGen	$\mu_1 = 1.0, \mu_2 = 0.0$	9.18	23.2
TCPGen	$\mu_1 = 1.0, \mu_2 = 1.0$	**8.79**	**21.4**

Table 4.5 WER and R-WER on Librispeech test-clean set for LSTM-based AED models trained on Librispeech clean-100 training set. MBWE parameters include μ_1 and μ_2 in Eqn. (4.13). The baseline here refers to the standard LSTM AED model. Biasing list size is 1000 with distractors randomly selected from the full rare word list.

using the MWE loss reduced WER by 0.4% in absolute value. Using the MBWE loss with $\mu_1 = 1.0, \mu_2 = 1.0$, the WER was further reduced and the reduction in R-WER provided the major contribution to this improvement. The MBWE training achieved further WER and R-WER reduction compared to MWE for both the baseline and TCPGen. When TCPGen is used with CE training alone, a 43% relative R-WER reduction was achieved compared to the baseline with the same training condition. However, this relative reduction was decreased to 42% when only the MWE loss was applied despite the reduction in the overall WER, as the MWE loss, which applies equal weighting to each word error, tends to benefit frequent words more. Finally, using the MBWE loss improved the R-WER which in turn improved the overall WER. As a result, the relative R-WER reduction was increased to 45% using the MBWE loss compared to the baseline with the same training condition.

System	MBWE params.	%WER	%R-WER
Baseline	$\mu_1 = 0.0, \mu_2 = 0.0$	12.82	42.1
Baseline	$\mu_1 = 1.0, \mu_2 = 0.0$	12.61	42.0
Baseline	$\mu_1 = 1.0, \mu_2 = 1.0$	12.60	41.6
Baseline	$\mu_1 = 1.0, \mu_2 = 5.0$	12.68	42.2
TCPGen	$\mu_1 = 0.0, \mu_2 = 0.0$	11.06	29.3
TCPGen	$\mu_1 = 1.0, \mu_2 = 0.0$	10.98	29.3
TCPGen	$\mu_1 = 1.0, \mu_2 = 5.0$	**10.85**	**28.3**

Table 4.6 WER and R-WER on Librispeech test-clean set for LSTM-based RNN-T models trained on Librispeech clean-100 training set. MBWE params. include μ_1 and μ_2 in Eqn. (4.13). The baseline here refers to the standard LSTM RNN-T model. Biasing list size is 1000 with distractors randomly selected from the full rare word list.

Using the restricted beam-search MBWE loss (see Section 4.4.2), LSTM-based RNN-T systems trained on the LibriSpeech clean-100 set were evaluated on the test-clean set as shown in Table 4.6. Compared to the baseline, a 30% relative R-WER reduction was achieved using TCPGen, and when MBWE training was applied, this reduction in R-WER increased to 32%.

4.7.2 TCPGen with Conformer Encoder Experiments on LibriSpeech

The TCPGen component together with proposed MBWE and BLMD algorithms was first applied to the conformer AED model.

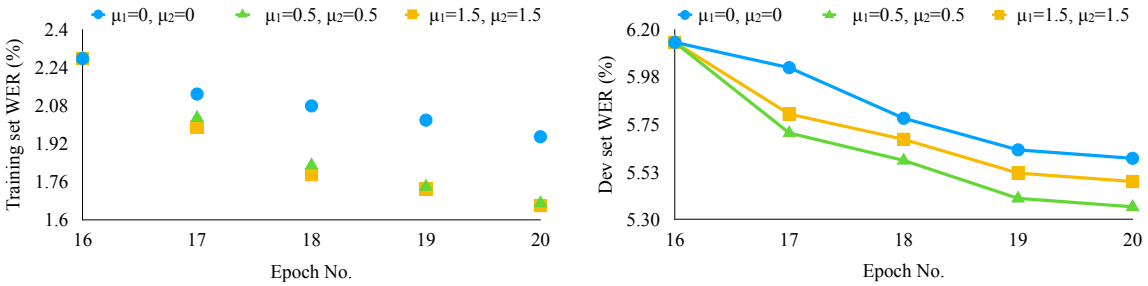


Fig. 4.9 Plots of training (left) and dev (right) set WERs across 4 training epochs. The training set WER was calculated on 5% randomly sampled utterances from the full 960-hour training set. Dev-set combines both dev-clean and dev-other sets. MBWE parameters μ_1, μ_2 were defined in Section (4.13).

The Noam optimiser was used and the learning rate was a smooth function. Preliminary experiments found that adjusting the weight of the cross-entropy loss yielded significantly

worse results as it effectively introduced an abrupt change to the learning rate. Therefore, to adjust the contribution of the MBWE loss, different values of μ_1 and μ_2 in. Eqn. (4.13) were used while keeping the coefficient of the cross-entropy loss set to 1. The effect of using small and large values of μ_1 and μ_2 on the training and dev set WER are shown in Fig. 4.9. As shown in Fig. 4.9, applying MBWE with both small and large values had a similar effect on the training set WER, whereas using smaller values produced better results on the dev set and hence was adopted for the experiments.

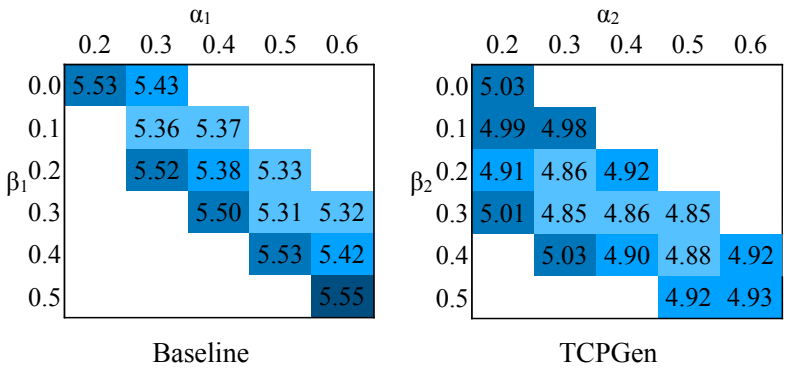


Fig. 4.10 Illustration of tuning BLMD hyper-parameters for the baseline standard Conformer AED model and the Conformer AED model with TCPGen. Numbers in each grid are dev set WER as a percentage. Left: Tuning α_1, β_1 on the baseline model. Right: Tuning α_2, β_2 on the TCPGen model with the best set of α_1, β_1 found from the baseline on the left.

The best set of BLMD parameters was searched for and then applied to the trained models during decoding. The search procedure is illustrated in Fig. 4.10. The left part of Fig. 4.10 shows the dev set WER of different sets of BLMD parameters α_1, β_1 for the baseline standard AED system. The best values found here were $\alpha_1 = 0.5, \beta = 0.3$, which were then fixed for the search of α_2, β_2, as shown on the right part of Fig. 4.10 for the system with TCPGen. As a result, $\alpha_{=}0.3, \beta_2 = 0.3$ were used which indicates that a stronger LM discounting effect was needed for the TCPGen distribution.

The results for the Conformer AED model are summarised in Table 4.7. As shown in Table 4.7, for the baseline standard Conformer AED system, using the MBWE loss benefits the R-WER. When MBWE is applied to TCPGen, there was a 12% relative reduction in R-WER on the test-clean set and 9% on the test-other set compared to the TCPGen system without MBWE training. This increased the relative R-WER improvement by using TCPGen from 33% to 41% on the test-clean set, and from 28% to 32% on the test-other set compared to the baseline with the same training loss (i.e. comparing row 1 with row 4 and row 3 with row 5 in Table 4.7).

Systems	MBWE	BLMD	test-clean WER (R-WER) (%)	test-other WER (R-WER) (%)
Standard AED	$\mu_1 = 0, \mu_2 = 0$	No	3.71 (13.2)	9.36 (29.5)
Standard AED	$\mu_1 = 0.5, \mu_2 = 0$	No	3.65 (13.0)	9.02 (28.9)
Standard AED	$\mu_1 = 0.5, \mu_2 = 0.5$	No	3.62 (12.8)	9.08 (28.6)
+ TCPGen	$\mu_1 = 0, \mu_2 = 0$	No	3.23 (8.6)	8.43 (21.3)
+ TCPGen	$\mu_1 = 0.5, \mu_2 = 1.0$	No	**2.96** (**7.6**)	**7.88** (**19.5**)
Standard AED	$\mu_1 = 0, \mu_2 = 0$	DR	3.33 (12.3)	8.04 (27.6)
Standard AED	$\mu_1 = 0.5, \mu_2 = 0$	DR	3.19 (12.2)	7.95 (27.1)
Standard AED	$\mu_1 = 0.5, \mu_2 = 0.5$	DR	3.17 (11.7)	7.92 (27.3)
+ TCPGen	$\mu_1 = 0, \mu_2 = 0$	BLMD	2.79 (6.9)	7.40 (19.5)
+ TCPGen	$\mu_1 = 0.5, \mu_2 = 1.0$	BLMD	**2.59** (**6.4**)	**7.13** (**18.2**)

Table 4.7 WER and R-WER (in bracket) on LibriSpeech test-clean and test-other sets for Conformer-based AED trained on LibriSpeech full 960-hour training set. MBWE column includes μ_1 and μ_2 in Eqn. (4.13), and the BLMD column indicates whether density ratio (DR) or BLMD was used. Biasing list size is 1000 with distractors randomly selected from the full rare word list.

After applying BLMD, obvious reductions in R-WER were observed for both the baseline standard AED systems and TCPGen systems on both test sets. In particular, large R-WER reductions were found when different discounting factors were applied to the TCPGen distribution, which further increased the relative R-WER improvement by using TCPGen from 41% to 46% on test-clean and from 32% to 33% on test-other (comparing row 3 with row 5 and row 8 with row 10 in Table 4.7).

Experiments were then performed on LibriSpeech full 960-hour training data as shown in Table 4.8. Preliminary experiments on LibriSpeech found that μ_2 for the MBWE loss should be set larger than μ_1 for better performance when using TCPGen in RNN-T. The baseline here is a standard Conformer-based RNN-T model. The MBWE and BLMD hyper-parameters were found in the same way as for the AED experiments. In addition to the standard baseline system, the DB method proposed in Le et al. (2021a) was used as a biasing method for comparison. In general, consistent and significant WER and R-WER reductions were achieved using TCPGen compared to both the baseline and the DB system, with a p-value less than 0.001. MBWE with a restricted beam search achieved WER and R-WER reductions for both baseline and TCPGen systems. In particular, the relative R-WER improvement increased from 37% to 41% on the test-clean set and from 33% to 37% on the

Systems	MBWE	BLMD	test-clean WER (R-WER) (%)	test-other WER (R-WER) (%)
Standard RNN-T	$\mu_1 = 0, \mu_2 = 0$	No	4.02 (14.1)	10.12 (33.1)
Standard RNN-T	$\mu_1 = 0.5, \mu_2 = 0$	No	4.01 (14.0)	9.96 (32.5)
Standard RNN-T	$\mu_1 = 0.5, \mu_2 = 0.5$	No	3.87 (13.8)	9.80 (31.8)
+ DB	$\mu_1 = 0, \mu_2 = 0$	No	3.57 (10.4)	9.45 (25.0)
+ TCPGen	$\mu_1 = 0, \mu_2 = 0$	No	3.40 (8.9)	8.79 (22.2)
+ TCPGen	$\mu_1 = 0.5, \mu_2 = 1.0$	No	**3.12** (**8.1**)	**8.64** (**20.7**)
Standard RNN-T	$\mu_1 = 0, \mu_2 = 0$	DR	3.55 (12.5)	8.90 (30.4)
Standard RNN-T	$\mu_1 = 0.5, \mu_2 = 0$	DR	3.38 (12.6)	8.59 (29.5)
Standard RNN-T	$\mu_1 = 0.5, \mu_2 = 0.5$	DR	3.28 (12.2)	8.50 (28.8)
+ TCPGen	$\mu_1 = 0, \mu_2 = 0$	BLMD	3.02 (8.0)	7.49 (18.6)
+ TCPGen	$\mu_1 = 0.5, \mu_2 = 1.0$	BLMD	**2.79** (**7.0**)	**7.44** (**18.2**)

Table 4.8 WER and R-WER (in bracket) on LibriSpeech test-clean and test-other sets for Conformer-based RNN-T models trained on LibriSpeech full 960-hour training set. MBWE column includes μ_1 and μ_2 in Eqn. (4.13), and the BLMD column indicates whether density ratio (DR) or BLMD was used. Biasing list size is 1000 with distractors randomly selected from the full rare word list.

test-other set compared to the baseline system (comparing row 4 to row 1 and row 6 to row 3 in Table 4.8).

Applying BLMD achieved further WER and R-WER reductions for all systems. In particular, BLMD increased the relative R-WER reduction from 41% to 43% on test-clean and from 37% to 38% on test-other (comparing row 6 to row 3 and row 11 to row 9 in Table 4.8). The sign test was used to compare TCPGen with and without MBWE training, as the WER and R-WER were smaller than those observed in AED. All R-WER improvements after applying MBWE, including those when BLMD was applied, were significant at $p < 0.05$.

4.7.3 TCPGen with Conformer Encoder Experiments on AMI

To further show the effectiveness of TCPGen on a different type of data, models trained on the LibriSpeech full 960-hour data were finetuned on 10% of the AMI training set. For baseline and TCPGen systems trained with CE on LibriSpeech, finetuning was performed only with CE loss, whereas for systems trained with MBWE, MBWE loss was also applied during finetuning. Hyper-parameters for MBWE were found the same way as before on the AMI dev set, with $\mu_1 = 1.0, \mu_2 = 3.0$ found to be the best for AED and $\mu_1 = 0.5, \mu_2 = 2.0$

System	AED(%)	RNN-T(%)
Baseline full AMI	22.53 (54.8)	25.49 (58.4)
Baseline	22.90 (53.4)	25.94 (57.0)
Baseline + MBWE	22.75 (52.8)	25.71 (56.0)
Baseline + MBWE + BLMD	21.79 (49.7)	24.37 (53.7)
TCPGen	21.86 (41.1)	25.58 (45.5)
TCPGen + MBWE	21.66 (39.1)	25.41 (44.2)
TCPGen + MBWE + BLMD	**20.74** (**36.9**)	**24.15** (**41.0**)

Table 4.9 WERs and R-WERs (in bracket) on AMI eval set for Conformer RNN-T models trained on LibriSpeech full 960-hour training set and finetuned on 10% of AMI train set. Baseline referred to the standard Conformer AED or RNN-T systems. Biasing list size is 1000 with distractors randomly selected from the augmented full rare word list for AMI.

for RNN-T. The best BLMD hyper-parameters were $\alpha_1 = 0.3, \beta_1 = 0.2$ for both AED and RNN-T baselines, $\alpha_2 = 0.2, \beta_2 = 0.1$ for TCPGen in AED and $\alpha_2 = 0.2, \beta_2 = 0.2$ in RNN-T. WER and R-WER for both AED and RNN-T were reported in Table 4.9.

Using 10% of the AMI training data to finetune LibriSpeech 960-hour models achieved similar WER and R-WER to those trained on the full AMI training data only. Consistent and significant WER and R-WER reductions were obtained using TCPGen compared to baseline systems, with p-values less than .001. Using MBWE and BLMD for both baseline and TCPGen, relative R-WER reductions were further increased from 23% to 26% for the AED model and from 20% to 24% for RNN-T. The sign test was also performed comparing TCPGen with MBWE training with TCPGen with CE training, and R-WER improvements were significant at a p-value less than 0.05 for both AED and RNN-T.

4.7.4 TCPGen with Conformer Encoder Experiments on DSTC

Finally, TCPGen, MBWE and BLMD were evaluated on the DSTC data where the biasing list was extracted from the ontology. Models trained on the LibriSpeech 960-hour data were finetuned on the DSTC track2 training set. For the baseline and TCPGen systems without MBWE training, finetuning was performed only with the CE loss, whereas for systems trained with MBWE, the MBWE loss was also applied during finetuning. The hyper-parameters for MBWE were found in the same way as before, with $\mu_1 = 0.5$ and $\mu_2 = 5.0$. Moreover, as it is difficult to obtain a large amount of external task-oriented dialogue data for LM training, an LM was trained only on the DSTC2 training data to perform either shallow fusion or LM discounting. For the baseline system, this DSTC LM was found to be more effective as an

SF LM with $\alpha_1 = 0.1, \beta_1 = 0.0$, which achieved a limited WER improvement. For TCPGen, BLMD was applied with the same α_1 and β_1 as the baseline, and $\alpha_2 = 0.0, \beta_2 = 0.1$, which, in addition to the SF, the internal LM effect was discounted in the TCPGen distribution. The WER and R-WER are reported in Table 4.10 where the R-WER was measured for biasing words that appeared in the ontology.

System	AED(%)	RNN-T(%)
Baseline	21.31 (61.5)	21.26 (64.2)
Baseline + MBWE	20.81 (60.4)	21.15 (64.1)
Baseline + MBWE + SF	20.73 (60.4)	20.63 (64.1)
TCPGen	20.38 (45.2)	20.05 (49.2)
TCPGen + MBWE	20.00 (43.6)	19.87 (47.4)
TCPGen + MBWE + BLMD	**19.74** (**40.3**)	**19.13** (**40.9**)

Table 4.10 WER and R-WER (in brackets) on the DSTC3 test set for Conformer AED and RNN-T models trained on LibriSpeech full 960-hour training set and finetuned on DSTC2 train and dev sets. The biasing list was the one extracted from DSTC ontology.

Progressively larger WER and R-WER reductions were achieved by applying MBWE and BLMD successively for both AED and RNN-T using TCPGen. For AED, using TCPGen achieved a 26% relative R-WER reduction compared to the baseline, which became a 33% reduction with BLMD for AED. For RNN-T, TCPGen alone achieved a 23% relative R-WER reduction compared to the baseline, which became a 36% relative R-WER reduction compared to the corresponding baseline with SF. Moreover, a dialogue-by-dialogue sign test was performed between TCPGen and TCPGen with the MBWE loss. For both AED and RNN-T, R-WER improvements were significant at a p-value less than 0.05.

4.7.5 Discussion

Training methods	WER
Standard	13.05
MWE	12.92
MWE restricted	**12.91**

Table 4.11 LibriSpeech test-clean set WER using LSTM-based RNN-T with standard RNN-T training, MWE training and MWE training with restricted beam search after five epochs using a quarter of clean-100 training set.

The following discussion of the TCPGen component is based on LibriSpeech experiments. First, the effectiveness of the restricted beam search for MBWE training is shown in Table 4.11. The standard LSTM-based RNN-T systems were used. Compared to the standard training, both MBWE training methods achieved a similar WER reduction, whereas the unrestricted beam-search MBWE training was 5 times slower than the restricted one. Moreover, the WER curve against training epoch was smoother for the restricted beam search, as the hypotheses had more similar sequence lengths than the unrestricted one, resulting in a more stable training process.

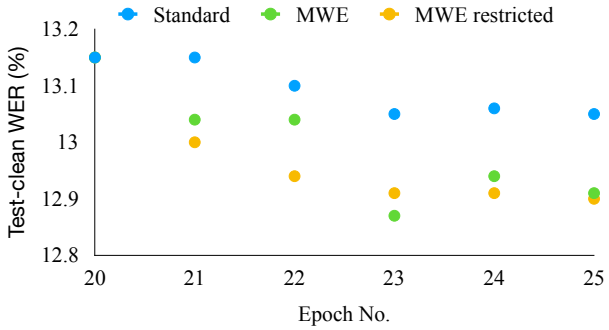


Fig. 4.11 Test-clean set WER using LSTM-based RNN-T with standard RNN-T training, MWE training and MWE training with restricted beam search. A quarter of the clean-100 training set was used. Epoch 20 is the starting epoch of MWE training when the first learning rate annealing happened.

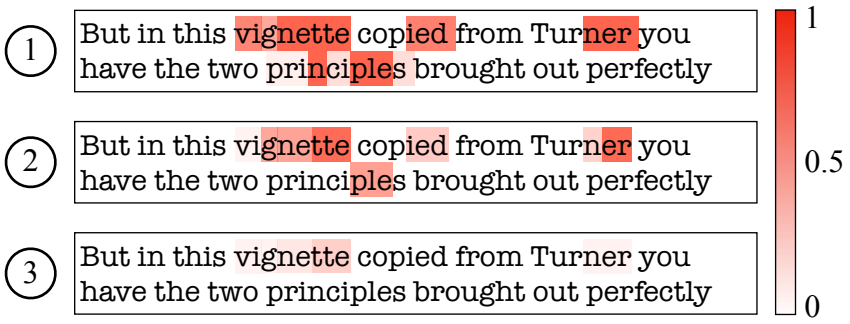


Fig. 4.12 Heat map showing the generation probability for each word piece in an utterance taken from recognition results: ① AED + TCPGen; ② RNN-T + TCPGen; ③: RNN-T + TCPGen + DB, to show how each system spots where to use contextual biasing. Biasing words are vignette and Turner.

As shown in Fig. 4.12, RNN-T with TCPGen in general outputs a lower generation probability than AED with TCPGen. One possible reason to explain this difference is that the loss for RNN-T was calculated at each of the $T \times N$ combinations, but only a small portion of those corresponds to outputting a new token that involved TCPGen. However,

AED only calculated the loss at each decoder step, where TCPGen was always needed. As a result, RNN-T with TCPGen performed better on unbiased words while poorer on biased words than AED. Moreover, DB further reduced the dependency of RNN-T on the TCPGen component, as part of the functionality of the TCPGen component is performed by $\mathbf{h}^{\text{ptr}}$. A similar phenomenon was observed when using DB in AED, but the structural difference meant that improvements were only found in RNN-T.

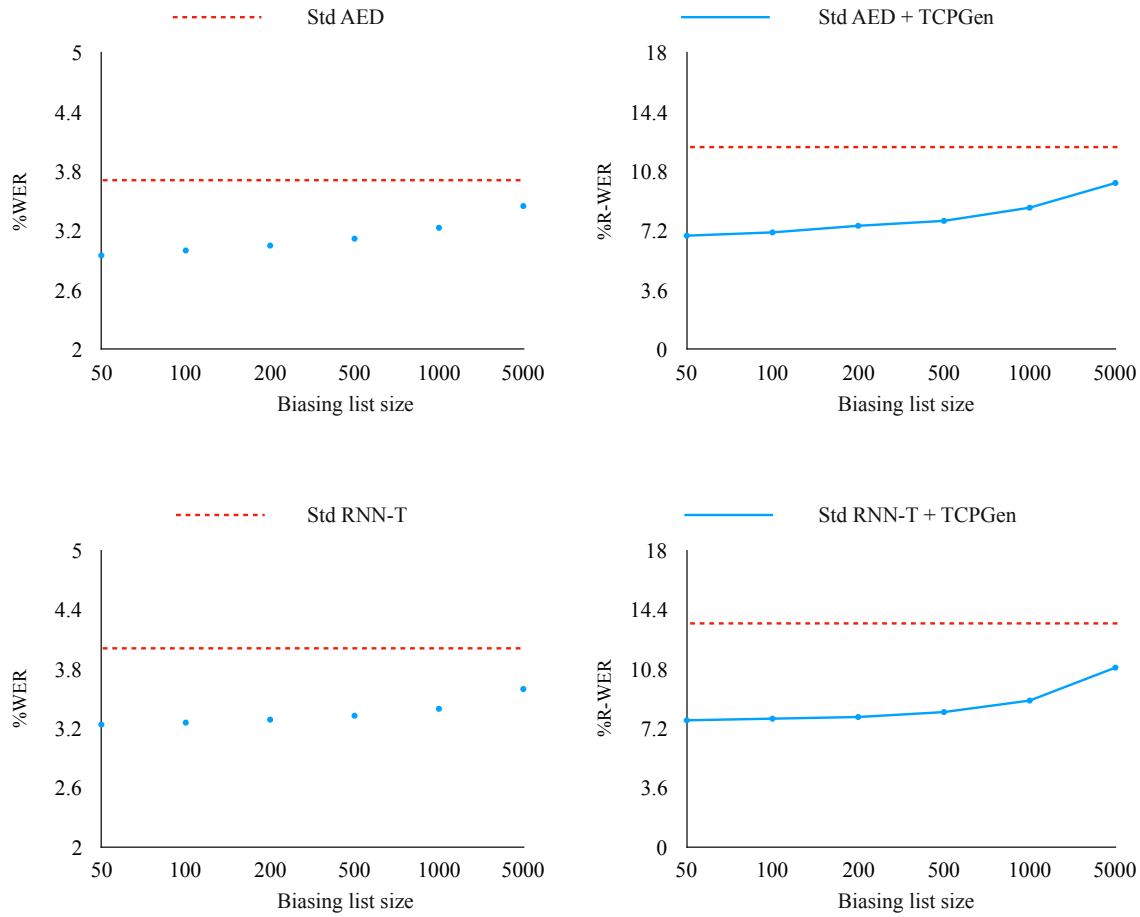


Fig. 4.13 The evolution of WER and R-WER w.r.t. the size of the biasing lists. the upper two plots were for Conformer AED while the lower two plots were for Conformer RNN-T.

Next, the investigation into sizes of biasing lists was performed by plotting the variation in WER and R-WER against biasing list sizes, as shown in Fig. 4.13. As shown in Fig. 4.13, for both AED and RNN-T, WER and R-WER increased as the biasing list size got larger. However, even at a size of 5000 biasing words, TCPGen was still able to achieve a relative WER reduction of 10% and a relative R-WER reduction of 23% for Conformer AED. Similar WER and R-WER effects were also observed with Conformer RNN-T.

In both AED and RNN-T with LSTM or Conformer decoder, TCPGen improved the error rates of those unbiased words in addition to the biasing words. In fact, the most common error pattern of rare word errors is to replace it with a couple of other common words, and such replacement will influence the recognition of surrounding unbiased words as well. For example, “FARRINDER SUPPOSED" was recognised as “FANNING JUST SURPRISED", and “OLIVE CHANCELLOR" was recognised as “ON A CHANCE NOW". In both cases, the correct recognition of the rare words (i.e. “FARRINDER" and “OLIVE") would benefit unbiased words (i.e. “JUST", “SURPRISED" and “CHANCELLOR") a lot. As a result, improvements achieved in rare words by using TCPGen also benefit unbiased words.

Systems	AED	RNN-T
Baseline + MBWE + BLMD	75.6%	78.0%
TCPGen + MBWE + BLMD	**37.8**%	**41.7**%

Table 4.12 Zero-shot WERs on OOV words on the combined Librispeech test-clean and test-other set using the baseline and TCPGen systems with MBWE and BLMD. Same biasing lists were used as those in Table 4.7 and Table 4.8.

TCPGen also achieved *zero-shot learning* [1] of audio training set OOV words incorporated in the biasing list. There were 468 OOV word tokens in the combined test-clean and test-other set that were covered by the biasing list. The OOV WER which was measured in the same way as R-WER but for OOV words on the combined test sets was separately reported in Table 4.12 for the baseline and for TCPGen systems using MBWE and BLMD. These systems used exactly the same biasing lists with 1000 distractors as those in Table 4.7 and Table 4.8, so they had the same WER and R-WER as those corresponding systems. As a result, TCPGen achieves a large OOV WER reduction compared to the best baseline system, and the majority of OOV words could be correctly recognised once included in the biasing list.

4.7.6 Summary

The performance characteristics of TCPGen has been investigated by experiments on the LibriSpeech, AMI and DSTC datasets. TCPGen achieved significant and consistent WER and R-WER reductions across all three datasets and was shown to be an effective and efficient mechanism to integrate dynamic contextual knowledge. Moreover, advanced training and decoding algorithms including MBWE and BLMD achieved further significant improvements in the contextual biasing performance of TCPGen.

[1] Zero-shot was referred to in this thesis with the assumption that the word is incorporated in the biasing list during inference.

Although experiments on LibriSpeech corpora are already viewed as fairly large-scale in an academic setting, with the fast-paced developments of software and hardware for deep learning, models now can be trained on hundreds of thousands of hours of speech from a wide range of speaking styles, background noises and channel properties. The question of whether contextual biasing, and specifically TCPGen, can remain effective with models trained on this scale needs to be answered, and the following sections will make the attempt to address this question.

4.8 TCPGen for Whisper

Recent advancements in software and hardware have facilitated the significant expansion of end-to-end trainable automatic speech recognition (ASR) systems through the use of copious amounts of training data. In this regard, large-scale supervised models with extensive training, such as *Whisper* (Radford et al., 2023), have been developed in addition to numerous foundation models featuring self-supervised training, e.g. wav2vec2.0 and GPT2 (Radford et al., 2019, 2023). Whisper contains a series of ASR models that have been trained in a supervised manner on hundreds of thousands of hours of labelled speech, which can be broadly applied across a diverse range of ASR tasks. Unlike self-supervised foundation models, Whisper aims to provide a readily deployable “out-of-the-box" solution for ASR tasks in various environments, circumventing the need for supervised fine-tuning of a decoder for every deployment. This approach aims to mitigate the issue of decoder brittleness when fine-tuned for specific tasks.

Notwithstanding the comprehensive training data used in Whisper and GPT2, domain-specific words, such as proper nouns, are rare or absent in the general training data and may be susceptible to errors when used for a specific domain. On the other hand, fine-tuning these models on a limited amount of domain-specific data may potentially compromise their ability to generalise to other tasks. Therefore, it is essential to enhance the performance of these models on such domain-specific words without sacrificing their capacity for generalisation, and contextual biasing is one possible solution. The inclusion of words, such as restaurant names in an ontology for a task-oriented dialogue system, in the biasing list has been shown in this chapter to significantly improve recognition performance.

Based on the aforementioned concerns and the solution, the final part of this chapter endeavours to answer the question: “Can contextual biasing remain effective in Whisper and GPT2?" with the constraint that the parameters of both models are unchanged. Specifically, TCPGen is a good fit for this task because, as a neural biasing component, it achieves biasing via *distribution-level adaptation* which does not require any modifications to the

Whisper model. In this section, modifications to TCPGen to be compatible with Whisper, together with a dedicated training scheme on small-scale *task-specific* data will be introduced. Notably, the proposed training scheme enables the portability of TCPGen as a neural biasing component, allowing the "out-of-the-box" nature of Whisper to be maintained without the necessity of fine-tuning any of its components.

4.8.1 Background for Whisper and GPT2

Whisper (Radford et al., 2023) is a Transformer-based (Vaswani et al., 2017) encoder-decoder model trained purely on supervised data for multiple languages and multiple speech-related tasks. The training data contained paired multilingual speech and text obtained from the Internet. The amount of training data consisted of 680k hours of audio, of which 117k hours were non-English data. The released Whisper models have various sizes, ranging from 39M parameters (4 encoder blocks/4 decoder blocks) to 1550M parameters (32 encoder blocks/32 decoder blocks). The tokeniser used the same set of Byte-Pair Encoding (BPE) tokens as the GPT2 (Radford et al., 2019) model for English-only models and with extra tokens for multilingual models. Whisper provides competitive ASR results across multiple languages and domains, without further fine-tuning.

GPT2 (Radford et al., 2019) is a Transformer based LM trained to predict the next word piece via self-supervision, as explained in Section 2.4. The training data were gathered from a social media platform, where the resulting text was more than ten billion words. GPT2 and its predecessor GPT (Radford et al., 2018) have been used to improve the performance of an existing ASR model (Li et al., 2020b, Zheng et al., 2021).

4.8.2 Biasing Whisper with TCPGen

The integration of TCPGen into the Whisper model is illustrated in Fig. 4.14. TCPGen takes as input the Whisper final decoder block hidden state $\mathbf{h}^{\text{dec}}$ to derive the query to calculate attention as shown in Eqn. (4.21)

$$\mathbf{q}_i = \text{ReLU}(\mathbf{W}\mathbf{h}_i^{\text{dec}}) \tag{4.21}$$

where $\mathbf{W}$ is the parameter matrix. The key and value vectors directly used the decoder word piece embeddings. The computation of the generation probability also used the final hidden state of the decoder, as shown in Eqn. (4.22).

$$P_i^{\text{gen}} = \sigma(\mathbf{W}_1\mathbf{h}_i^{\text{dec}} + \mathbf{W}_2\mathbf{h}_i^{\text{ptr}}) \tag{4.22}$$

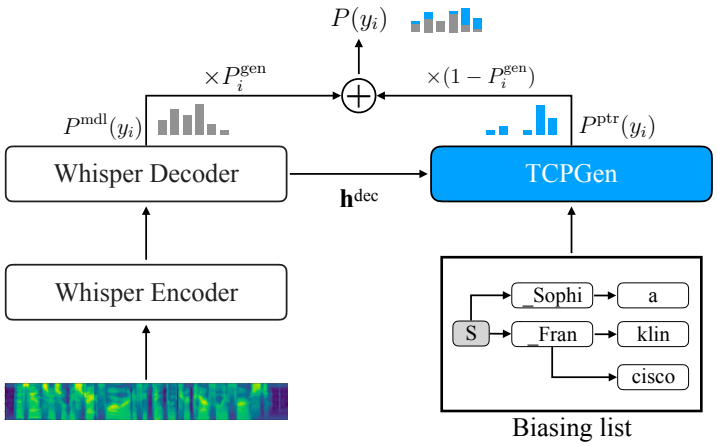


Fig. 4.14 Integration of TCPGen in Whisper with corresponding terms in Eqn. ((4.3)). The $\bigoplus$ symbol represents linear interpolation with weights P^{gen} predicted by TCPGen. Only the TCPGen part is updated during training. The biasing list example here contains `Sophia`, `Franklin` and `Francisco`, and underscore marks the word start

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are the parameter matrices. Then, through Eqn. (4.3), TCPGen leverages this distribution-level adaptation to bias the Whisper output, by only requiring the output from the final decoder block of Whisper. During training on task-specific data, the entire Whisper model is frozen and only the TCPGen parameters are updated. In this way, Whisper can still generalise to other data while TCPGen as a biasing component provides task-specific biasing information.

Moreover, as the word frequencies of the training data for Whisper are not available, a word-error-based approach was adopted to extract a good biasing list for training instead of using a frequency-based one. This was achieved by decoding the task-specific training set and gathering distinct word tokens with errors higher than the mean.

4.8.3 Rescoring with GPT2

GPT2, as an off-the-shelf large pretrained LM, was used to rescore the N-best hypotheses from the ASR model, as the Whisper large model has a different set of word piece units to the GPT2 model. To improve the effectiveness of using GPT2, an ILME method was also used following (Meng et al., 2021b). The N-best hypotheses were derived by using the beam search algorithm during Whisper decoding, together with the log probability scores associated with each hypothesis.

In addition to GPT2, the log probability of each hypothesis subtracts the internal LM score, which is estimated by forwarding the Whisper model with the encoder output set to all-zero vectors. The total score of each hypothesis based on which the re-ranking was

performed is computed by:

$$\log P_{\text{Whisper}}(Y|X) - \alpha \log P_{\text{ilm}}(Y) + \beta \log P_{\text{gpt2}}(Y) \tag{4.23}$$

where $P_{\text{Whisper}}(Y|X)$ is the probability of hypothesis Y given acoustic feature X predicted by Whisper, $P_{\text{ilm}}(Y)$ and $P_{\text{gpt2}}(Y)$ are the internal and external LM probabilities of Y, and α and β are hyper-parameters to be tuned.

4.9 Experiments with Whisper and GPT2

4.9.1 Experimental Setup

Experiments were performed on three datasets, including the LibriSpeech clean data, SLURP data and the dialogue state tracking challenge Track 2 (DSTC2). LirbiSpeech was used to represent a fairly generic dataset while the other two were more specific to certain tasks. The descriptions of each data set and the extraction of the biasing list during inference were provided below.

LirbiSpeech train-clean-100 set and the dev sets were used for training TCPGen and the two test sets were used for evaluation. The same full rare word list (Le et al., 2021a, Sun et al., 2021c), as described before containing 200k words, was used. During inference, one biasing list for each utterance was obtained by collecting words that appeared in the full rare word list and adding a certain number of distractors, which, if not specified, was 1000 for LibriSpeech and SLURP.

SLURP (Bastianelli et al., 2020) is a dataset containing single-turn user interactions with a home assistant, annotated with scenarios, actions and entities, where 58 hours of real speech data were used for training. As in Sun et al. (2023b), the full biasing list was obtained by extracting words from slot entities that appeared in the error-based biasing list for training. Out-of-training-set words in slot entities were also included, making the full rare word list of SLURP contain a total of 2.8k words. The biasing list for each utterance during inference was organised in the same way as for LibriSpeech.

DSTC2 was used the same way as the DSTC 2 and 3 in the Section 4.7.4, where the biasing list was extracted from the DSTC ontology. Adding out-of-training-set words into the full rare word list contains 380 words due to the small ontology and was used as a whole without the need for reference transcription.

Note that as the Whisper tokeniser is case sensitive, for each biasing word, a copy of it with the first character capitalised was also included in the biasing list, which doubled the

size of each biasing list. The feature extraction, pre-processing and tokenisation followed the Whisper pipeline (Radford et al., 2023).

Experiments were performed with both the Whisper "base.en" model trained on English data and the Whisper large trained on multilingual data[2]. The base model contained a 6-block Transformer encoder and decoder with 74M parameters, and the large model contained a 32-block Transformer encoder and decoder with 1550M parameters. TCPGen was trained separately on each dataset for 30 epochs using an Adam optimiser with a linear tri-state learning rate scheduler. The training was performed on a single A100 GPU, and, as an example, the training time of the "base.en" model with TCPGen on LibriSpeech clean-100 was 5 hours. The GPT2 base model was used to rescore 50-best lists without fine-tuning, with α and β values searched between 0 and 1 on validations sets respectively.

As before, WER and R-WER were used as the evaluation metric, and one decimal place was kept to be consistent with the Whisper paper (Radford et al., 2023). Moreover, to keep the performance comparable and compatible with other biasing setups (Le et al., 2021a, Sun et al., 2021c), unless indicated, text normalisation was not used for scoring, and a discussion on this is provided in Section 4.9.2.

4.9.2 Results

System	Test-clean		Test-other	
	WER (%)	R-WER (%)	WER (%)	R-WER (%)
Whisper base.en	5.2	16.3	10.5	31.4
Whisper base.en + TCPGen	4.8	12.5	10.1	26.5
Whisper base.en + GPT2	4.9	14.9	9.9	29.4
Whisper base.en + TCPGen + GPT2	4.5	11.7	9.7	25.0
Whisper large	4.0	10.4	6.7	20.0
Whisper large + TCPGen	3.4	8.3	6.3	16.3
Whisper large + GPT2	3.9	10.1	6.6	19.6
Whisper large + TCPGen + GPT2	**3.4**	**8.2**	**6.3**	**16.3**

Table 4.13 WER and R-WER on LibriSpeech test sets using Whisper and TCPGen, rescored with GPT2. Test-time biasing list selection followed the description in Sec. 4.9. LM weights tuned on dev sets of each data separately.

The main results are summarised in Table 4.13 for LibriSpeech and Table 4.14 for the other two data sets. Compared to the Whisper "base.en" model, using contextual biasing

[2]Code for Whisper models: https://github.com/openai/whisper

System	SLURP		DSTC2	
	WER (%)	R-WER (%)	WER (%)	R-WER (%)
Whisper base.en	27.8	72.5	21.1	71.3
Whisper base.en + TCPGen	25.8	52.3	18.2	41.8
Whisper base.en + GPT2	26.7	67.7	20.8	67.9
Whisper base.en + TCPGen + GPT2	24.8	47.0	16.9	38.7
Whisper large	16.7	61.6	17.3	69.1
Whisper large + TCPGen	15.1	37.1	14.7	33.5
Whisper large + GPT2	16.6	59.3	17.3	68.5
Whisper large + TCPGen + GPT2	**15.0**	**37.1**	**13.9**	**29.5**

Table 4.14 WER and R-WER on SLURP test set and DSTC2 test set using Whisper and TCPGen, rescored with GPT2. Test-time biasing list selection followed the description in Sec. 4.9. LM weights tuned on dev sets of each data separately.

achieved an average 20% relative R-WER reduction on test-clean and test-other sets, a 28% relative R-WER reduction on SLURP and a 42% relative R-WER reduction on DSTC2. Reductions on domain-specific data were much larger than that on the generic LibriSpeech data, since the test time biasing list of LibriSpeech contained mainly generic words, whereas the biasing lists for the other two test sets contained domain-specific words such as names of podcasts or restaurants. As biasing word tokens in LibriSpeech and SLURP occupied around 10% of the total number of word tokens in the test sets, the reduction in overall WER was greater than the overall error reduction reflected by R-WER, indicating that contextual biasing was also beneficial for unbiased words rather than degrading their performance. This effect was most obvious for the DSTC2 data. Contextual biasing achieved larger improvements with the large multilingual Whisper model, with 40% relative R-WER reduction on SLURP and over 50% R-WER reduction on DSTC2.

When GPT2 was applied to the Whisper “base.en" model, the relative R-WER reductions became slightly smaller for LibriSpeech as GPT2 served as another generic knowledge source, where the R-WER reduction was retained for other datasets. However, applying GPT2 to the large model resulted in a much smaller improvement than to the “base.en" model. As before, larger relative WER and R-WER reductions were observed with the large multilingual model when GPT2 was used for rescoring. TCPGen achieved an 18% relative R-WER reduction on LibriSpeech test sets, 37% on SLURP and 57% on DSTC compared to the Whisper large model with GPT2 rescoring.

4.9.3 Discussion

First, the variation of model performance against the sizes of the test-time biasing list was examined, where SLURP was used for this purpose as a non-generic test set, as shown in Fig. 4.15. Up to the full-sized biasing list of 5.6k biasing words, which achieved a 60.5% R-WER which did not require any knowledge about the reference, TCPGen still outperformed the baseline by a relative 18% reduction in R-WER.

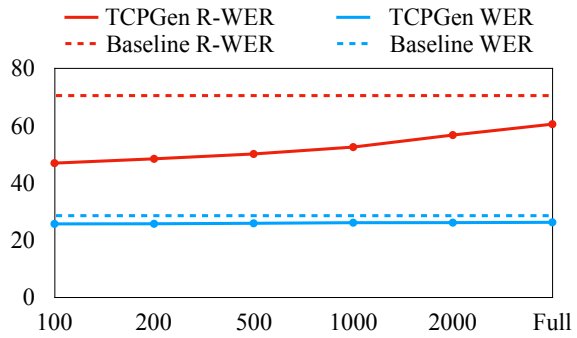


Fig. 4.15 WER and R-WER on SLURP using TCPGen and Whisper base.en model. The baseline is “base.en" without TCPGen. Full had 5.6k biasing words.

Meanwhile, the influence of text normalisation used in the Whisper paper as a post-processing stage should be mentioned for both the reference and hypothesis. Results with normalisation are provided in Table 4.15 for the LibriSpeech test-clean set. Text normalisation

System	test-clean (%)	SLURP (%)	DSTC (%)
Whisper norm.	2.5 (8.1)	14.4 (38.7)	19.9 (67.8)
Whisper + TCPGen norm.	**2.3** (**7.0**)	**13.9** (**29.2**)	**14.3** (**32.3**)

Table 4.15 WER and R-WER (in bracket) on LibriSpeech test-clean, SLURP and DSTC using Whisper large model and TCPGen, together with text normalisation. A beam size of 10 was used.

had a similar effect on test-other and SLURP, with a relative R-WER reduction of 25% on SLURP. It was much less influential to DSTC2 and the relative R-WER improvement remained at 50%. Nevertheless, text normalisation may obscure the improvements to a biasing word. For example, a word in its possessive contraction form (e.g. Steven’s) was recognised as its original form (e.g. Steven) while this could potentially be corrected by TCPGen by incorporating that in the biasing list. However, when text normalisation was applied, the reference word would be split into two words, e.g. Steven is. If Steven was in the biasing list, there would be an additional hit to reduce the R-WER, as ’s was obviously

not a biasing word and its error did not count. If Steven was not on the biasing list, such a correction by TCPGen would be omitted when calculating R-WER. Therefore, it was necessary to evaluate the performance of contextual biasing without text normalisation.

Next, the effect of using an error-based biasing list compared to using a frequency-based biasing list is shown in Table 4.16. As before, SLURP was used for this purpose. As the frequency-based biasing list had a much lower average WER than the error-based one, TCPGen was not needed as much by the Whisper model. As a result, using an error-based biasing list for training achieved 12% additional R-WER reduction compared to the frequency-based one.

Biasing list	WER (%)	R-WER (%)
Baseline	27.8	72.5
Frequency-based	26.6	59.8
Error-based	**25.8**	**52.3**

Table 4.16 Effect of the error-based biasing list for training TCPGen evaluated using SLURP data on the Whisper base model. The frequency biasing list contained words that appeared less than 30 times in the training set.

Finally, as TCPGen required training on a task-specific set, it is worthwhile examining the performance of words that did not belong to that training set. An OOV WER (Sun et al., 2022b) was used which was measured the same way as R-WER but for words not in the task-specific training set, with results shown in Table 4.17. Whisper with TCPGen biasing achieved 30% relative OOV WER reduction compared to using Whisper alone, which was much higher than that achieved on other biasing words. As a generic dataset, words not covered by the LibriSpeech training set had a high chance of being rare in the Whisper training set and hence TCPGen had a much larger improvement on them. The situation was reversed on the SLURP and DSTC as two domain-specific datasets, and words not covered by the training set may be common in the training set of Whisper, especially for DSTC. Therefore, the OOV WER reduction was smaller on those two datasets compared to the R-WER reduction.

4.9.4 Summary

This section examined the effectiveness of neural contextual biasing with TCPGen for Whisper in conjunction with GPT2. To allow TCPGen to be compatible with Whisper, structural changes were made to adapt TCPGen, along with a dedicated training scheme without the need to fine-tune Whisper. Experiments across three datasets revealed that using

System	LibriSpeech (%)	SLURP (%)	DSTC (%)
Whisper	35.0	67.1	15.1
Whisper + TCPGen	**24.5**	**45.3**	**11.7**

Table 4.17 Effect of TCPGen on words not in the task-specific training set using the Whisper large model, measured by OOV WER defined the same way as R-WER. OOV WER for test-clean and -other was calculated together.

TCPGen resulted in a large reduction of R-WER when a biasing list of 1000 words was utilised. In particular, 37% and 57% relative R-WER reductions were achieved on SLURP and DSTC2 respectively. The findings also suggested that contextual biasing is more effective when applied to domain-specific data and has the potential to enhance the performance of universal models without sacrificing their capacity for generalisation. Although only Whisper and GPT2 were investigated in this section, the biasing scheme can be easily integrated with other universal ASR and LMs, such as Google's USM (Zhang et al., 2023) if the parameters are available.

4.10 Summary and Conclusions

The key findings and contributions in this chapter are summarised as follows:

- Tree-constrained pointer generator (TCPGen) was introduced to integrate dynamic contextual knowledge into end-to-end trainable ASR systems. TCPGen constructed a neural shortcut that directly connected the ASR output distribution with the biasing list. It also computed a generation probability for dynamic interpolation.
- TCPGen, as a generic contextual biasing component, was integrated into both the attention-based encoder-decoder (AED) system and the RNN transducer (RNN-T) system.
- A dedicated minimum biasing word error (MBWE) training criterion was proposed for training systems with TCPGen, and a biasing-word-driven LM discounting (BLMD) decoding strategy was proposed for inference.
- Experiments on the LibriSpeech audiobook data, AMI meeting data and DSTC spoken dialogue data demonstrated the effectiveness of TCPGen for contextual biasing. TCPGen was found to be able to handle large biasing lists containing 5000 biasing words efficiently and was much more effective than deep biasing method.

- In order to show the effectiveness of neural contextual biasing on universal ASR models trained on hundreds of thousands of speech data, TCPGen was modified as a distribution-level adaptation component for the Whisper model, together with a set of dedicated training methods without the need to change any Whisper parameters.
- GPT2 was also used as a universal large LM to rescore the ASR hypothesis. Therefore, the effectiveness of TCPGen was evaluated under the combination of a universal speech model and a large LM, which formed a potential paradigm for the future.
- ASR experiments on LibriSpeech, DSTC and SLURP (a spoken language understanding corpus) showed consistent improvements using TCPGen as a neural biasing component. TCPGen was found to be more effective on domain-specific data such as DSTC and SLURP, than on more generic data such as LibriSpeech.

As described in this chapter, the TCPGen distribution is only computed by only looking at the possible set of next word pieces via their word piece embeddings. Those embeddings are not contextualised, i.e. the same word pieces leading into different words on the prefix-tree are not distinguished. On the other hand, the lookahead functionality that incorporates the information on the future branches is helpful to determine the TCPGen distribution and is also particularly useful for the prediction of the generation probability. In the next chapter, graph neural networks (GNNs), as a natural subsequent extension by exploiting the tree structure, will be introduced to TCPGen to obtain prefix-tree encodings that carry future branch information to further improve the performance of TCPGen in contextual ASR.

Chapter 5

TCPGen with GNN encodings and Audio-visual Contextual ASR

5.1 Introduction

TCPGen (Sun et al., 2021c), as an efficient neural biasing component that structures biasing lists into prefix trees and builds a pointer generator neural shortcut to modify the ASR output directly, has been introduced in Chapter 4. Further exploiting the prefix-tree structure in TCPGen, this chapter introduces the use of graph neural network (GNN) encodings in TCPGen to obtain more powerful node representations for the prefix-tree. Three different types of GNN for prefix-tree encoding in TCPGen, including the tree recursive neural network (Tree-RNN), graph convolutional network (GCN) (Kipf and Welling, 2017) including its variant GCNII (Chen et al., 2020b), and GraphSAGE (Hamilton et al., 2017a) with the max-pooling aggregator. While Tree-RNN is a representative GNN model with a single recursive layer, GCN and GraphSAGE are two popular and effective multi-layer GNN designs. Specifically, GCN encodes the tree by using a spectral representation, while GraphSAGE, as a spatial method, directly explores the graph topology (Zhou et al., 2020). To further enhance the performance of GNN tree encodings, this chapter proposes attentive and bilinear combination approaches to explore the complementarity between GCN and GraphSAGE. Additionally, this chapter introduces an effective parameter-tying scheme for both GCN and GraphSAGE, which aims to improve their performance with deeper structures.

GNN encodings provide more powerful node representations in the prefix tree of TCPGen, allowing for “lookahead" functionality where each node contains not only its own word piece information but also information about its child branches. This improved node representation in TCPGen leads to more accurate generation probability predictions for biasing words,

enabling better contextual biasing by incorporating information about future word pieces during each ASR decoding step. TCPGen with GNN encodings, as a generic component for end-to-end ASR, has been integrated into both attention-based encoder-decoder (AED) and recurrent neural network transducer (RNN-T).

In addition to GNN encodings, this chapter also expands the scope of contextual biasing to a multi-modal scenario and introduces an audio-visual contextual ASR pipeline for the augmented multi-party interaction (AMI) meeting data. In this pipeline, contextual knowledge in biasing lists was obtained from the presentation slides alongside each meeting with an optical character recognition (OCR) tool. With contextual biasing, important named entities such as the presenter's name or the name of new concepts are more likely to be correctly recognised. Moreover, this demonstrates the great potential of applying contextual biasing, especially using TCPGen, in academic and educational scenarios.

This chapter is organised as follows. Section 5.2 introduces GNNs in general, together with the application of GNNs to speech and language tasks. Section 5.3 explains the application of the three GNNs used in TCPGen. Section 5.4 introduces the experimental setup and the audio-visual contextual ASR pipeline. Experimental results are provided in Section 5.5, and the chapter concludes in Section 5.6.

5.2 Background

5.2.1 Graph Neural Networks

Graphs are a form of data structure that represent a collection of entities (known as nodes) and their connections (known as edges). As a non-Euclidean data structure in the domain of machine learning, graph analysis is concerned with tasks such as node classification, link prediction, and clustering. The application of GNNs in TCPGen focuses on the node classification task. GNNs are deep learning techniques designed to operate on graph structures. Owing to their impressive performance (Zhou et al., 2020), GNNs have emerged as a popular approach for graph analysis in recent times.

The primary impetus for GNNs can be traced back to the extensive history of neural networks in the context of graph-based data structures. Previous work investigated the use of recursive neural networks for directed acyclic graphs (Sperduti and Starita, 1997, Frasconi et al., 1998). Subsequently, MLPs (Micheli, 2009) and RNNs (Scarselli et al., 2009) were introduced to handle cyclic graphs. Despite achieving success, these methods relied on building state transition systems on graphs and iteratively updating them until convergence, which limited their flexibility and representation capacity. The emergence of CNNs in deep

learning marked a turning point, as their local connections, shared weights, and multi-layered architecture proved instrumental in solving graph-based problems. Although traditional CNNs were originally designed for Euclidean structures such as 2D images or 1D sequences, they were later extended to graph structures which can also be seen as instances of non-Euclidean data structures. This extension of deep neural models to non-Euclidean domains, referred to as geometric deep learning, has become a burgeoning area of research (Bronstein et al., 2017).

Another significant motivation for GNNs stemmed from graph representation learning (Cui et al., 2017, Hamilton et al., 2017b, Cai et al., 2017), which involves learning low-dimensional vectors that represent graph nodes, edges, or subgraphs. Inspired by the success of word embeddings, DeepWalk (Perozzi et al., 2014) was the first graph embedding method that employed SkipGram (Mikolov et al., 2013) models on randomly generated walks. Subsequent research has adopted similar approaches, but often suffers from generalisability and computational inefficiency (Hamilton et al., 2017b).

Based on CNNs and graph embedding, variants of GNNs have been proposed to aggregate information from graph structure. Thus GNNs can model input and/or output consisting of elements and their dependency. This chapter divides GNNs by their propagation modules along the edge, which can be divided into convolution operations and recurrent operations. The Tree-RNN and tree-LSTM are two typical GNNs with recurrent propagation modules while GCN and GraphSAGE have convolution modules. The key difference between recurrent and convolution modules is whether the same layer is recurrently used throughout the path traversing the graph. The convolution operator is further divided into spatial approaches and spectral approaches. The GCN and GCNII belong to the spectral approach which computes node encodings using the normalised graph Laplacian, $\mathbf{L} = \mathbf{I}_N + \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ where $\mathbf{I}_N$ is the identity matrix of N nodes, $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is the adjacency matrix. The degree matrix is a diagonal matrix where each element i,i is the number of edges connected to node i, and the adjacency matrix is the matrix where each element i,j is 1 when there is a connection between node i and node j, and 0 if there is not. GraphSAGE belongs to the spatial approach where it directly exploits the local structure of the graph, i.e. it computes each node encodings by considering nodes connected to it.

5.2.2 GNN for Speech and Language Tasks

GNNs have been extensively employed in a multitude of speech and language tasks. In language-related applications, such as sentence-level text classification or word-level sequence labelling, GNNs are often used to capture the syntactic dependencies or semantic relations among words in a sentence. Furthermore, the encoding of subword-unit-based tree

structures using GNNs (Luong et al., 2013, Nguyen et al., 2019) has been explored for the purpose of generating more effective word representations. GNNs have also been applied to named-entity recognition (Li et al., 2017) and have been integrated into neural machine translation systems (Liu et al., 2014, Kawara et al., 2018).

GNNs have also found numerous applications in speech processing. One such application is text-to-speech synthesis, where GNNs have been used to model the syntactic and semantic relationships in the text. In GraphTTS (Sun et al., 2020a), the authors structured the sentence into a hierarchical tree by dividing the utterance into words and then further into characters. This allowed the system to capture prosodic relationships among different parts of the input. Additionally, GNNs have been applied to paralinguistic tasks, such as sentiment classification and hate speech detection (Chen et al., 2015a, Zhang et al., 2018, Sadr et al., 2019), where GNNs were used as the syntactic encoder. In Sun et al. (2022b), a Tree-RNN structure was used to encode a word piece prefix-tree in the TCPGen component for contextual biasing.

5.3 GNN Encodings for TCPGen

In standard TCPGen, node representations only contain information about the word piece associated with that node. However, anticipating future branches in the prefix tree can greatly enhance the accuracy of generation probability prediction. To incorporate lookahead functionality, GNNs are used to embed information about future branches into each node representation. The pipeline of integrating GNN encodings into TCPGen is illustrated in Figure 5.1.

The word piece prefix-tree is first encoded with a GNN to obtain encodings associated with each node. Then, the tree with GNN encodings is used by TCPGen, where the key and value for the TCPGen distribution are computed using the encoding of nodes in the set of valid word pieces, in place of word piece embeddings as shown in Eqn. (5.1).

$$\mathbf{k}_j = \mathbf{W}^{\mathrm{K}}\mathbf{h}^{\mathrm{gnn}}_{n_j} \qquad \mathbf{v}_j = \mathbf{W}^{\mathrm{V}}\mathbf{h}^{\mathrm{gnn}}_{n_j}, \tag{5.1}$$

where $\mathbf{W}^{\mathrm{V}}$ and $\mathbf{W}^{\mathrm{K}}$ are parameter matrices, and $\mathbf{h}^{\mathrm{gnn}}$ is the GNN node encoding obtained using different types of GNN. This section introduces three different types of GNN, namely the Tree-RNN, GCN (including its variant, GCNII) and GraphSAGE with max pooling, together with combinations of GCN and GraphSAGE as two complementary types of GNN. Details of these GNN structures applied in TCPGen together with modifications are described in the following sections.

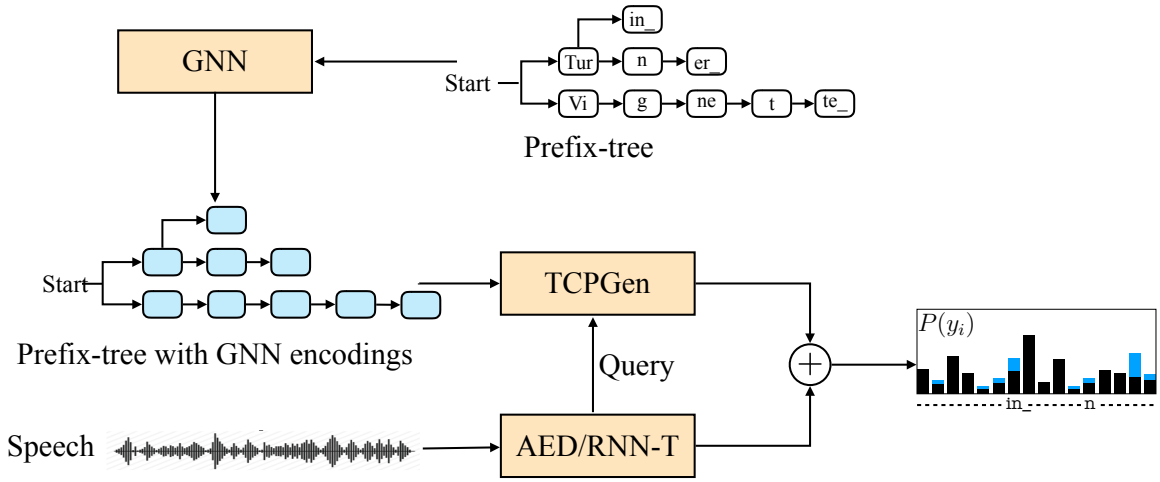


Fig. 5.1 Pipeline of encoding prefix-tree with GNN for TCPGen. The prefix-tree is first encoded by a GNN, and the GNN-encoded tree is used by TCPGen to generate the TCPGen distribution where key and value vectors are GNN-based node encodings.

5.3.1 Tree Recursive Neural Networks (Tree-RNN)

The Tree-RNN recursively encodes the tree from leaf nodes to the root using an RNN structure. Specifically, at node n_j which contains child nodes $n_1,...,n_k,...,n_K$, the vector representation of n_j can be written as Eqn. (5.2).

$$\mathbf{h}_{n_j}^{\text{trnn}} = \text{ReLU}(\mathbf{W}_1\mathbf{y}_j + \sum_{k=1:K} \mathbf{W}_2\mathbf{h}_{n_k}^{\text{trnn}}), \tag{5.2}$$

where $\mathbf{h}_{n_k}^{\text{trnn}}$ is the vector representation of node n_k, and $\mathbf{y}_j$ is the embedding vector of the word piece of node n_j. $\mathbf{W}_1$ and $\mathbf{W}_2$ are parameter matrices jointly optimised with the ASR system by allowing gradient back-propagation through $\mathbf{h}_{n_k}^{\text{trnn}}$. In this way, each node recursively encodes information from its child nodes, such that the information of the entire branch rooted from it can be incorporated in the node encoding $\mathbf{h}_{n_k}^{\text{trnn}}$.

Before the forward pass of the main ASR model, the encoding of each node is computed by Eqn. (5.2) recursively from leaf to root. Then, for the same example shown in Fig. 5.1, at the node of `Tur`, node encodings $\mathbf{h}_{\texttt{in\_}}^{\text{trnn}}$ and $\mathbf{h}_{\texttt{n}}^{\text{trnn}}$ are used to calculate the TCPGen distribution and $\mathbf{h}_i^{\text{ptr}}$. Therefore, if `Turner` is in the utterance, TCPGen is aware of this entire word as early as in the encoding of `Tur`. Such lookahead functionality achieves a more accurate prediction of the generation probability to determine when contextual biasing is needed.

Although Tree-RNN achieves lookahead functionality, it uses a rather simple RNN structure to encode the information of all succeeding nodes on that branch into a single vector

representation. Therefore, more powerful and flexible GNN encodings are explored in the following sections in order to improve performance.

5.3.2 Graph Convolutional Network (GCN)

As an alternative method to the Tree-RNN, the GCN is applied for tree encodings to achieve better node representations with controllable lookahead distance. The GCN is a multi-layer network where each layer computes the encoding of a node as a function of its neighbours based on the graph Laplacian matrix. Specifically, define $\mathbf{H}^{\text{gcn}}(l) = [\mathbf{h}^{\text{gcn}}_{n_1}(l), ..., \mathbf{h}^{\text{gcn}}_{n_N}(l)]$ as the node encoding matrix of layer l whose rows $\mathbf{h}^{\text{gcn}}_{n_j}(l)$ are node encodings corresponding to node n_j, the computation of each GCN layer is shown in Eqn. (5.3).

$$\mathbf{H}^{\text{gcn}}(l+1) = f(\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{H}^{\text{gcn}}(l)\mathbf{W}(l)) \tag{5.3}$$

where $\mathbf{W}(l)$ is the parameter matrix of layer l, $f(\cdot)$ is the activation function, $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with self-loops which allows the information of the current node to be included in the node representation, and $\hat{\mathbf{D}}$ is the degree matrix of $\hat{\mathbf{A}}$. This specific form of the normalised graph Laplacian (termed re-normalised graph Laplacian), $\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$, is used to address the vanishing/exploding gradient problem. Note that as only future branch information is needed in TCPGen, $\hat{\mathbf{A}}$ only contains edges that lead to child nodes, and hence $\hat{\mathbf{D}}$ is computed based on this modified $\hat{\mathbf{A}}$. TCPGen then takes the node encodings of the final layer, $\mathbf{H}^{\text{gcn}}(L)$, to compute key and value vectors in the same way as the Tree-RNN. As a practical consideration for deep networks in general, residual connections and layer normalisation are added for any GNNs with multiple layers.

Each layer of a GCN enables one message to be passed from immediate neighbouring nodes. For a GCN with L layers, the encoding of a node covers information from a node that is an L-hop ahead on branches rooted from it. Therefore, by controlling the number of layers L, the scope of the lookahead can be configured. However, recent research found that the performance of GCNs starts to degrade with more than three layers (Chen et al., 2020b, Gasteiger et al., 2019). Apart from the fact that deeper networks are more difficult to train, the representations of the nodes in GCN are inclined to converge to a certain value and hence become indistinguishable, which is referred to as the *over-smoothing* problem. One promising method to address this problem is to build a shortcut directly linking to the first GCN layer to ensure a certain fraction of the final representation comes from the current node itself. Thus, the GCNII variant is also investigated in this chapter with each layer computed

as in Eqn (5.4):

$$\mathbf{H}^{\text{gcnii}}(l+1) = f\Big([(1-\alpha)\hat{P}\mathbf{H}^{\text{gcnii}}(l) + \alpha\mathbf{H}^{\text{gcnii}}(0)]\mathbf{W}_{\beta}(l)\Big) \tag{5.4}$$

where $\mathbf{H}^{\text{gcnii}}(l)$ is the l-th layer output of GCNII, $\hat{P} = \hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}$, hyper-parameter α scales the shortcut to the first layer, and $\mathbf{W}_{\beta}(l)$ is the parameter matrix defined as:

$$\mathbf{W}_{\beta}(l) = (1-\beta_l)\mathbf{I}_N + \beta_l\mathbf{W}(l) \tag{5.5}$$

where β_l is a layer-dependent hyper-parameter which is $\ln(1/l+1)$ in this chapter.

Although GCNII has shown superior performance to a deeper GCN of more than 4 layers, the maximum length in the biasing list is usually less than 10. With this depth for tree structures, the over-smoothing problem is less of a concern compared to the best structure of GCNII with 64 layers, and the network complexity is more problematic. Therefore, a simple parameter-sharing scheme is also explored in this chapter for deep GNNs to reduce network complexity. Specifically, the parameter matrices in the first K layers are shared:

$$\mathbf{W}(1) = \mathbf{W}(2) = ... = \mathbf{W}(K) \tag{5.6}$$

In contrast to other complex graph structures, message-passing operations from child nodes to the root in a tree were similar across different layers. Therefore, having the same weight matrix representing this process effectively reduces the model complexity to a degree that is adequate for tree encoding. In particular, $K = L-1$ for GCN in this chapter so that there are effectively two layer parameters to be trained while maintaining the depth of the network. The first K layers act as universal message passing and the last layer performs final information aggregation from neighbouring nodes.

5.3.3 GraphSAGE with Max-Pooling

Node encodings for Tree-RNN and GCN are based on summation, whereas previous research (Zhang et al., 2015, Shen et al., 2018) has found that using max pooling also achieves competitive performance for subword, word or sentence representations. Hence, as an alternative GNN structure, GraphSAGE with a max pooling aggregator function is also studied in this chapter. GraphSAGE is a multi-layer GNN with each layer performing information aggregation over a sampled set of child nodes followed by an update to the representation of the current node. Although one of the innovations in GraphSAGE is fixed-size sampling, as the training time biasing list is already a sampled subset from the full

biasing list, the sampling of GraphSAGE is omitted in this chapter. The computation of each layer is

$$\mathbf{h}_{\mathcal{N}_i}(l+1) = \max(\{\sigma(\mathbf{W}_1(l)\mathbf{h}^{\text{sage}}_{n_k} + \mathbf{b}(l)), \forall n_k \in \mathcal{N}_j\}) \tag{5.7}$$

$$\mathbf{h}^{\text{sage}}_{n_j}(l+1) = \sigma(\mathbf{W}_2(l)[\mathbf{h}_{\mathcal{N}_j}(l+1); \mathbf{h}^{\text{sage}}_{n_j}(l)]) \tag{5.8}$$

where $\max(\cdot)$ denotes the element-wise max pooling operator, $\mathcal{N}_j$ is the set of child nodes of node n_j. Although slightly better than GCN, GraphSAGE also degrades when adding more layers. Therefore, the same parameter-sharing scheme as GCN is adopted for GraphSAGE, where both $\mathbf{W}_1$ and $\mathbf{W}_2$ are separately shared across layers respectively.

TCPGen with GNN encodings still achieves high efficiency in handling large biasing lists. In training, with a large biasing list of 1000 words, TCPGen with a Tree-RNN was three times slower than the standard AED or RNN-T model, with a negligible increase in space complexity. Among the three GNNs, GCN achieved the highest efficiency for training as its computation can be parallelised the most, whereas the recursive computation in Tree-RNN and the max-pooling in GraphSAGE hinder their training speed respectively. Moreover, by generating GNN encodings offline before the start of decoding once the biasing list is available, the time and space complexity during inference is close to the standard AED or RNN-T for biasing lists of thousands of words.

5.3.4 Combination of GCN and GraphSAGE

A combination of GCN and GraphSAGE is also explored in this chapter for tree encodings, as they are conceptually complementary. GCN adopts a spectral approach where the graph Laplacian is used to aggregate information, while GraphSAGE directly exploits the graph structure and performs max-pooling aggregation. To exploit the complementarity between the two GNNs, both additive and multiplicative combination methods were investigated.

Additive combination performs a weighted sum of node encodings from GCN and GraphSAGE as the final node encodings before being processed by TCPGen (see Eqn. (5.9)).

$$\mathbf{h}^{\text{comb}}_{n_j} = \alpha^{\text{gcn}}\mathbf{U}_1\mathbf{h}^{\text{gcn}}_{n_j} + \alpha^{\text{sage}}\mathbf{U}_2\mathbf{h}^{\text{sage}}_{n_j} \tag{5.9}$$

where $\mathbf{U}_1$ and $\mathbf{U}_2$ are parameter matrices which can rearrange the orders of the elements in each GNN encoding as they may not encode information in the same order (Sun et al., 2021a). Note that $\alpha^{\text{gcn}} + \alpha^{\text{sage}} = 1$ are two weights that are either fixed or predicted via attention. The attention calculation is performed on each node n separately, as shown in Eqn.

(5.10).

$$[\alpha_{i,n}^{\text{gcn}}, \alpha_{i,n}^{\text{sage}}] = \text{Softmax}(\mathbf{q}_i^T [\mathbf{h}_{n_j}^{\text{gcn}}, \mathbf{h}_{n_j}^{\text{sage}}]) \tag{5.10}$$

where $\mathbf{q}_i$ is the same query vector used to calculate the TCPGen distribution. In this way, different sets of weights are assigned to different nodes at different decoder steps.

Multiplicative combination is performed via a low-rank approximation of the bilinear pooling method. The combination is shown in Eqn. (5.11)

$$\hat{\mathbf{h}}_{n_j}^{\text{comb}} = P(\tanh(\mathbf{U}_1 \mathbf{h}_{n_j}^{\text{gcn}}) \odot \tanh(\mathbf{U}_2 \mathbf{h}_{n_j}^{\text{sage}})) \tag{5.11}$$

where $\mathbf{P}$, $\mathbf{U}_1$ and $\mathbf{U}_2$ are parameter matrices, and $\odot$ is the element-wise product between two vectors. Following Sun et al. (2021a), a shortcut connection from each individual GNN encoding was provided to form the final combined encoding for TCPGen, as shown in Eqn. (5.12).

$$\mathbf{h}_{n_j}^{\text{comb}} = \hat{\mathbf{h}}_{n_j}^{\text{comb}} + \mathbf{V}_1 \mathbf{h}_{n_j}^{\text{gcn}} + \mathbf{V}_2 \mathbf{h}_{n_j}^{\text{gcn}} \tag{5.12}$$

where $\mathbf{V}_1$ and $\mathbf{V}_2$ are two parameter matrices.

5.4 Experimental Setup

5.4.1 Data

Experiments were performed on the LibriSpeech and AMI datasets whose descriptions can be found in Section 4.6.1. The experimental setup on LibriSpeech followed exactly the same as in Section 4.6.1, where the biasing list was formulated for each utterance by selecting biasing words (words belonging to the full rare word list) in the reference transcription and adding a certain number of distractors.

As before, to demonstrate the effectiveness of contextual biasing on data from another domain with limited training resources, a subset comprising 10% of the utterances from the AMI training set corresponding to 8 hours of audio was used to fine-tune the models previously trained on the LibriSpeech 960-hour data. There were 14 meetings from the dev set and 8 meetings from the eval set that were accompanied by slides, and they were selected to formulate the new test set for the audio-visual contextual ASR pipeline.

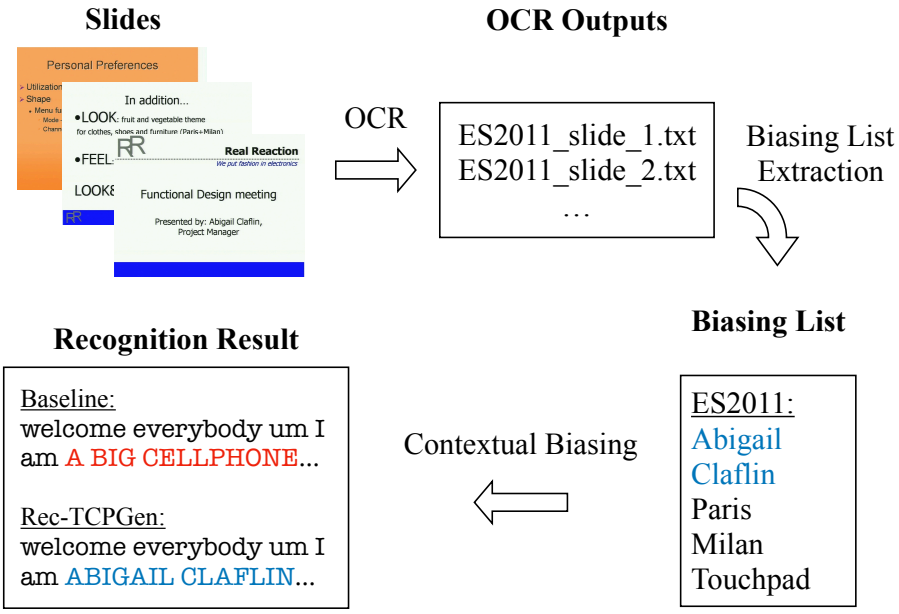


Fig. 5.2 Illustration of the visual-grounded contextual ASR pipeline for the meeting series ES2011 containing meetings ES2011a to ES2011d.

5.4.2 Audio-visual Contextual ASR Pipeline

Figure 5.2 illustrates the visual-grounded contextual ASR pipeline for AMI that utilises OCR output for slides. The Tesseract 4 OCR engine, equipped with LSTM models[1], was first applied to the slides of each meeting series (e.g., ES2011[a-d]). Subsequently, distinct word tokens were extracted from the OCR output text files, and words in the full rare word list, which also occurred fewer than 100 times in the AMI training set, were selected to form the biasing list for that particular meeting series. These meeting-specific biasing lists were then used for the recognition of all utterances in that meeting series. The size of the biasing lists varied between 175 to 576, and the total number of word tokens covered by these lists was 1,751 out of 112,110 word tokens (1.5%). As shown in the example in Fig. 5.2, these words mainly consisted of highly valuable content words whose accurate recognition was crucial for comprehending the utterance. Therefore, although the biasing lists had a minor impact on the overall word error rate (WER), they were essential for improving the recognition performance of critical words.

[1]Implementation at https://github.com/tesseract-ocr/tesseract

5.4.3 Model and Training Specifications

The model and training specifications in this chapter are similar to the previous chapter. The 80-dim FBANK features at a 10 ms frame rate concatenated with 3-dim pitch features were used as the model input. SpecAugment (Park et al., 2019) with the setting $(W, F, m_F, T, p, m_T) = (40, 27, 2, 40, 1.0, 2)$ was used without any other data augmentation or speaker adaptation.

A unigram word piece model based on 600 unique word pieces was created using the LibriSpeech data and was applied directly to the AMI data. Both the AED and RNN-T models employed a Conformer (Gulati et al., 2020) encoder, which comprised 16 conformer blocks comprising 4 attention heads of size 512. The AED used a single-layer LSTM decoder of size 1024 and a location-sensitive attention mechanism featuring 4 heads of size 1024. The RNN-T, on the other hand, employed a 1024-dim predictor and a joint network consisting of a single fully-connected layer of size 1024. GNN encodings for LibriSpeech train-clean-100 experiments used 256-dim GNN encodings, whereas the LibriSpeech full-scale experiments used 1024-dim for GNN encoders.

During training, biasing lists with 1000 distractors were used for the experiments conducted on the LibriSpeech dataset, while 100 distractors were used for the AMI data. To create these lists, biasing words were selected from the reference transcription, and additional distractors were added. To prevent the AED model from becoming overly confident about TCPGen outputs, a dropout-inspired technique was employed during training, as described in Le et al. (2021a). Specifically, biasing words that were present in the reference transcription had a 30% probability of being removed from the biasing list. The Conformer was optimised using the Noam optimiser Vaswani et al. (2017).

LM shallow fusion and biasing-driven LM discounting (BLMD) introduced in Chapter 4 were implemented using a two-layer LSTM-LM with 2048 hidden units trained on the 800 million-word text training corpus of LibriSpeech as the target domain LM for the LibriSpeech experiments. Each source domain LM, trained on the text of the audio training data, used a single-layer LSTM with 1024 hidden units. It is worth noting that each LM had the same word pieces as the corresponding ASR system.

5.4.4 Evaluation Metrics

Evaluation was performed based on WER, R-WER and OOV WER as described in Section 4.6.5 in detail. Moreover, the slides rare word error rate (R_s-WER) was reported for the AMI experiments calculated in the same way as R-WER, but for the rare words in slides. Insertions of slides biasing words were included in R_s-WER.

As rare words are scarce in the dataset, significance tests were performed to ensure that the improvements found by using GNN encodings were statistically significant. Specifically, independence was assumed at the book level for LibriSpeech and at the speaker level for AMI. The alternative hypothesis was defined as the GNN system performing better than the standard TCPGen (i.e. a one-tailed sign test).

5.5 Results

5.5.1 LibriSpeech Train-Clean-100 Results

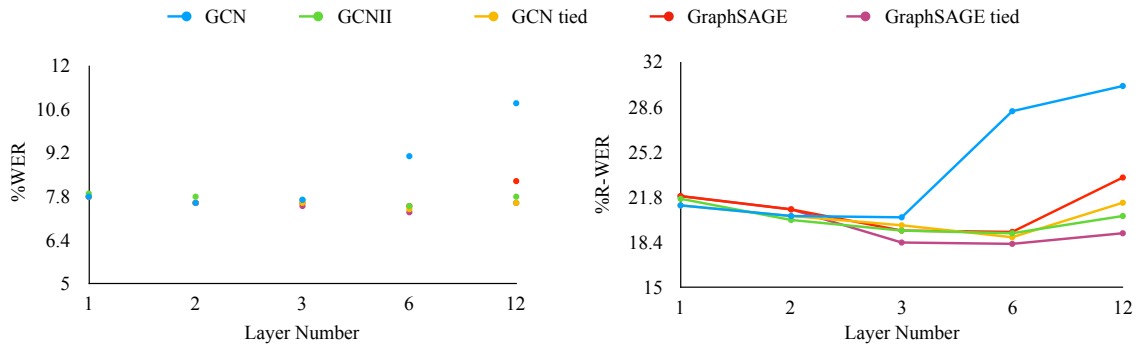


Fig. 5.3 Plot of WER (%) and R-WER (%) against the number of GNN layers for RNN-T on LibriSpeech test-clean data. Systems were trained on train-clean-100 for 120 epochs. Biasing lists with 1000 distractors were used. “Tied" referred to the parameter-tying scheme.

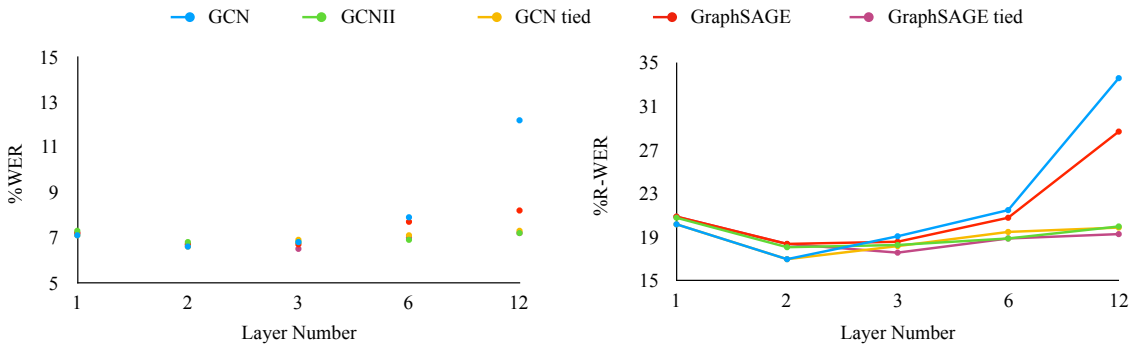


Fig. 5.4 Plot of WER (%) and R-WER (%) against the number of GNN layers for AED on LibriSpeech test-clean data. Systems were trained on train-clean-100 for 120 epochs. Biasing lists with 1000 distractors were used. “Tied" referred to the parameter-tying scheme.

Experiments were first performed on the train-clean-100 set to find the best-performing GNN setups. The first investigation was on the number of GNN layers which determines

how much lookahead is needed. The plots of WER and R-WER against the number of layers for RNN-T are shown in 5.3, and those for AED are shown in 5.4. Note that parameter tying only had effects when there were more than two layers.

For RNN-T, both GCN and GraphSAGE had a clear trend that the performance tended to degrade when the number of layers was increased beyond three and degraded significantly when it reached twelve layers. GraphSAGE suffered less from this problem than GCN as the max pooling operator enabled the gradient for salient nodes to remain at its original value instead of being over-smoothed Chen et al. (2020b). This degradation was clearly mitigated by either using the GCNII structure or the parameter-tying scheme proposed in this chapter. The best performance was achieved by 6-layer systems using WER as the selection criterion, where GCN and GraphSAGE with tied parameters achieved better performance than GCNII.

Similar observations were found with AED, where GCN and GraphSAGE with tied parameters achieved slightly better performance than GCNII. However, the best performance was achieved using 2-layer GCN and 3-layer GraphSAGE models. This difference is mainly caused by the label-synchronous nature of AED, where the model required knowledge for predicting only the next token, and information further into the future is much less useful for next-token prediction in contrast to RNN-T.

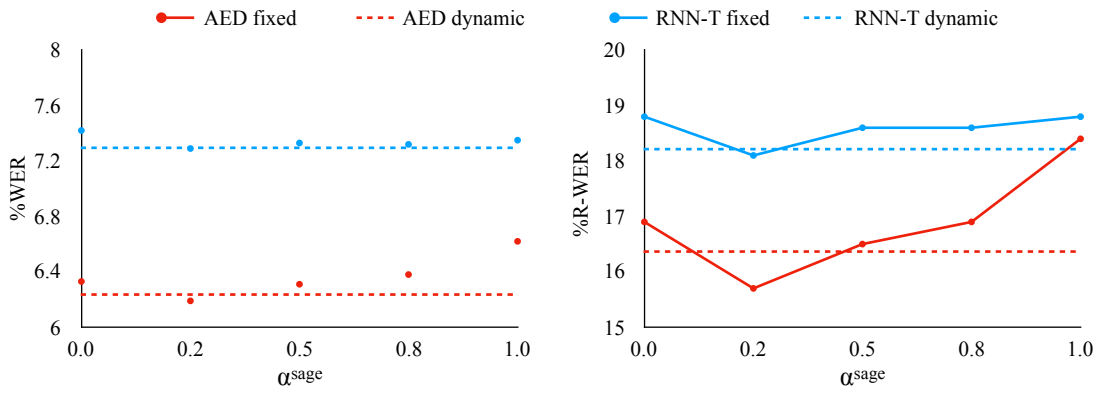


Fig. 5.5 Variation of WER and R-WER against model combination weight α_{sage}, and dynamic refers to using attention mechanism for weight calculation.

The best-performing GCN and GraphSAGE with tied parameters were used for model combination. For additive combination, systems with different fixed combination weights gave the results shown in Fig. 5.5, together with the system adopting attentive combination weights. As a result, dynamic combination performed better than most fixed-weight combinations, whereas the best performance was still obtained by the fixed-weight combination for both AED and RNN-T, with $\alpha_{\text{sage}} = 0.2$. By examining the predicted dynamic weights, it was found that unless at the root node of the tree, the dynamic weight almost completely

ignored GraphSAGE encodings and hence suffered from mode collapse, and hence GraphSAGE encodings were not properly trained. In fact, since GCN is usually better at handling near-future information, the model learnt to only rely on GCN.

5.5.2 LibriSpeech 960-hour Results

System	GNN Enc.	BLMD	test-clean (%)		test-other (%)	
			WER	R-WER	WER	R-WER
Conformer AED	N/A	×	3.71	13.2	9.36	29.5
+TCPGen	No	×	3.34	8.4	8.43	21.3
+TCPGen	Tree-RNN	×	3.13	6.7	7.94	17.8
+TCPGen	GCN tied	×	2.81	6.7	7.34	17.1
+TCPGen	GCNII	×	2.88	6.8	7.46	17.5
+TCPGen	GraphSAGE tied	×	2.91	6.6	7.42	17.0
+TCPGen	Additive (fixed)	×	2.59	5.8	7.00	15.5
+TCPGen	Additive (dynamic)	×	2.72	6.3	7.17	16.4
+TCPGen	Bilinear	×	2.62	6.1	7.08	16.1
Conformer AED	N/A	✓	3.33	12.3	8.04	27.6
+TCPGen	No	✓	2.79	6.9	7.40	19.5
+TCPGen	GCN tied	✓	2.48	5.2	6.33	14.2
+TCPGen	Additive (fixed)	✓	**2.26**	**4.2**	**5.87**	**11.5**

Table 5.1 WER and R-WER on LibriSpeech test-clean and test-other sets using Conformer AED and TCPGen trained on LibriSpeech 960-hour data with various GNN encodings. Note that both GCN (2-layer) and GraphSAGE (3-layer) adopted parameter tying.

The main results for the LibriSpeech full-set experiments were given in Table 5.1 for AED and in Table 5.2 for RNN-T respectively. Compared to the standard TCPGen, all three types of GNNs achieved significantly better WER and R-WER (at p values smaller than 0.01) on both test sets for both AED and RNN-T. In particular, using multi-layer GNN, such as GCN and GraphSAGE, achieved clearly better performance than Tree-RNN on AED, whereas the performance difference among those three GNN types was less obvious on RNN-T. The best-performing GNN structure on both AED and RNN-T was GCN with tied parameters. For AED, GCN achieved a 16% relative WER reduction with a 20% R-WER reduction on the test-clean set and a 13% relative WER reduction with a 19% relative R-WER reduction on the test-other set (comparing row 4 to row 2 in Table 5.1). For RNN-T, GCN achieved a 9% relative WER reduction with a 21% relative R-WER reduction on the test-clean set and a 7% WER reduction with a 17% relative R-WER reduction on the test-other set (comparing

System	GNN Enc.	BLMD	test-clean (%) WER	test-clean (%) R-WER	test-other (%) WER	test-other (%) R-WER
Conformer RNN-T	N/A	×	4.02	14.1	10.12	33.1
+TCPGen	No	×	3.40	8.9	8.79	22.2
+TCPGen	Tree-RNN	×	3.14	7.6	8.23	18.8
+TCPGen	GCN tied	×	3.11	7.0	8.14	18.4
+TCPGen	GCNII	×	3.16	7.7	8.36	18.9
+TCPGen	GraphSAGE tied	×	3.10	6.9	8.18	18.6
+TCPGen	Additive (fixed)	×	2.99	6.5	8.10	16.7
+TCPGen	Additive (dynamic)	×	3.02	6.6	8.14	17.2
+TCPGen	Bilinear	×	2.97	6.2	8.02	16.6
Conformer RNN-T	N/A	✓	3.55	12.5	8.90	30.4
+TCPGen	No	✓	3.02	8.0	7.49	18.6
+TCPGen	GraphSAGE tied	✓	2.71	6.3	7.34	17.7
+TCPGen	Bilinear	✓	**2.56**	**5.3**	**6.89**	**14.1**

Table 5.2 WER and R-WER on LibriSpeech test-clean and test-other sets using Conformer RNN-T and TCPGen trained on LibriSpeech 960-hour data with various GNN encodings. Note that both GCN (6-layer) and GraphSAGE (6-layer) adopted parameter tying.

row 4 to row 2 in Table 5.2). With similar levels of R-WER reduction, AED achieved a higher reduction in WER. As analysed in Section 4.7.5, TCPGen produced a much more confident prediction of P^{gen} with AED than RNN-T, where the main reductions in overall WER were attributed to the reduction in R-WER. The improvements using GNN indicated that the GNN encoding improved the prediction of P^{gen}, which was more beneficial for the overall WER in AED.

Both additive and bilinear combinations of GNN encodings achieved superior performance to individual GNN encodings with both AED and RNN-T models. For the AED model, the best performance was achieved by the additive combination with the best set of fixed weights previously found on train-clean-100 experiments, while the bilinear combination achieved very similar performance to the additive one. This led to a total of 31% relative R-WER reduction compared to the standard TCPGen (comparing row 7 to row 2 in Table 5.1), and a total of 56% relative R-WER reduction compared to the baseline Conformer AED model. The performance improvement with the GNN combination was smaller on RNN-T, with the best results achieved by the bilinear pooling combination. This resulted in a 30% relative R-WER reduction compared to the standard TCPGen (comparing row 9 to row 2 in Table 5.2), and an overall 56% relative R-WER reduction compared to the baseline Conformer RNN-T model.

Selected systems were evaluated with BLMD, where TCPGen with GNN encodings achieved a further performance improvement. The best-performing system for AED was TCPGen with the additive combination of GNN encodings, which achieved an overall 66% relative R-WER reduction on the test-clean set and 58% reduction on the test-other compared to the baseline. For RNN-T, an overall 57% relative R-WER reduction was achieved on test-clean, and 54% was achieved on test-other.

Systems	AED (%)	RNN-T (%)
Baseline	71.8	80.0
+ TCPGen	31.9	42.2
+ TCPGen + GCN	27.9	41.5
+ TCPGen + GCN + GraphSAGE	**23.5**	**34.9**

Table 5.3 Macro averaged OOV-WER across test-clean and test-other sets for AED and RNN-T using TCPGen with GNN encodings, under the same setup as Table 5.1 and 5.2 with 1000-word biasing lists. Baseline referred to the standard AED or RNN-T systems. Note that BLMD was applied for all systems in this table.

Moreover, the OOV-WERs were shown in Table 5.3, which had the same reduction pattern as R-WER for both AED and RNN-T. The best AED and RNN-T systems with combined GNN encodings for TCPGen reduced the OOV-WER by over 60%. Notably, BLMD was particularly beneficial for GNN encodings in AED systems, reducing the OOV-WER to 1/3 of the OOV-WER of the baseline standard Conformer AED. This confirmed that even though GNN required more parameters to encode the prefix tree, TCPGen still generalised well to unseen branches (i.e. OOV words) on the tree.

5.5.3 AMI Audio-Visual Contextaul ASR experiments

The performance of various GNN encodings for TCPGen was further investigated on the audio-visual contextual ASR pipeline, and results were shown in Table 5.4. In general, reductions in R-WER had a much smaller influence on the overall WER than with LibriSpeech, as rare words only occupied a much smaller portion. The findings were consistent with LibriSpeech, where the best-performing combination methods for AED and RNN-T were additive and bilinear respectively. However, the relative R-WER improvement was smaller compared to that in the LibriSpeech experiments, as the best-performing system here already had an R-WER that was very close to the overall WER whereas the R-WER in LibriSpeech was still twice as high as the overall WER. Compared to the baseline standard systems, TCPGen in AED achieved over 35% relative R-WER reduction using the best combined

System	GNN Enc.	BLMD	AED (%) WER	AED (%) R_s-WER	RNN-T (%) WER	RNN-T (%) R_s-WER
AMI Baseline	N/A	×	23.6	56.3	26.5	58.0
Baseline	N/A	×	22.2	51.2	25.7	52.6
+TCPGen	No	×	22.0	40.5	25.5	44.7
+TCPGen	Tree-RNN	×	21.9	36.7	25.4	40.7
+TCPGen	GCN tied	×	21.9	35.3	25.4	40.7
+TCPGen	GraphSAGE tied	×	21.9	36.4	25.2	39.6
+TCPGen	Additive Comb.	×	21.8	33.1	25.1	36.2
+TCPGen	Bilinear Comb.	×	21.8	33.5	25.1	36.2
Baseline	N/A	✓	21.1	45.5	24.3	46.5
+TCPGen	No	✓	20.9	34.2	24.1	37.7
+TCPGen	GCN tied	✓	20.8	32.2	23.7	33.8
+TCPGen	Additive Comb.	✓	**20.7**	**29.7**	23.6	33.2
+TCPGen	Bilinear Comb.	✓	20.7	31.1	**23.6**	**32.9**

Table 5.4 WER and R_s-WER on AMI test set using the audio-visual contextual ASR pipeline with 10% of AMI training set. Baseline refers to the standard AED and RNN-T systems, and AMI baseline refers to standard systems trained from scratch on the full AMI training set.

GNN encodings, while TCPGen in RNN-T achieved over 30% relative R-WER reduction with the best GNN encodings both with and without BLMD.

5.6 Summary

Key findings and contributions in this chapter are summarised as follows:

- The tree-constrained pointer generator (TCPGen) component was further augmented with Graph Neural Network (GNN) for prefix-tree encodings. Instead of using the word piece information associated with each node, GNN allowed the lookahead functionality so that information on branches rooted from a specific node was also incorporated in the node encoding. This enabled TCPGen to better predict whether the next word was in the biasing list through a more accurate generation probability, P^{gen}.
- Three types of GNN models were investigated for TCPGen, including the tree recursive neural network (Tree-RNN), the graph convolution network (GCN) together with its variant, GCNII, and GraphSAGE. While Tree-RNN represents the recurrent operation, GCN and GraphSAGE represent spectral and spatial operations respectively.

Moreover, as two complementary GNN structures, GCN and GraphSAGE were further combined at the final encoding level by additive combination and bilinear combination approaches.

- Experiments were conducted on the LibriSpeech audiobook and the augmented multi-party interaction (AMI) meeting corpora. In particular, the audio-visual contextual ASR pipeline was proposed for AMI, where contextual knowledge was extracted from the presentation slides associated with each meeting. Results on both corpora showed significant and consistent improvements using GNN encodings compared to the standard TCPGen. The audio-visual contextual ASR pipeline also demonstrated the potential of applying contextual biasing and TCPGen to many academic scenarios where presentation slides or papers are available.

So far, TCPGen has been applied to various ASR tasks. However, the long-tailed word problem also exists in end-to-end spoken language understanding (SLU) systems that also adopt a static set of model parameters. In practice, SLU tasks are usually accompanied by domain-specific knowledge bases, such as restaurant or music names. Such knowledge bases can be readily converted into biasing lists, and by predicting slot types, more focused biasing lists could be extracted. Moreover, the knowledge bases usually cover rare but important named entities, which are perfect targets to perform contextual biasing. Therefore, in the next chapter, TCPGen equipped with GNN encodings is also applied to end-to-end SLU systems to achieve improved SLU performance on rare but important named entities.

Chapter 6

TCPGen for End-to-end SLU

Spoken language understanding (SLU) plays a key role in spoken dialogue systems, which includes user intent detection and slot-filling. SLU is often implemented as a pipeline system that first transcribes speech into text with an ASR system, and then performs intent detection or slot-filling with a natural language understanding (NLU) component operating only on texts. The pipeline systems ignore the prosody and pronunciation information that is embedded in the speech but not available in the text transcription and can have more NLU errors propagated from the ASR errors, in particular, named entity-related errors. End-to-end SLU systems (Serdyuk et al., 2018, Haghani et al., 2018, Radfar et al., 2020) can potentially resolve these issues, by combining the ASR and NLU components into a single audio-grounded model. Such systems can be improved by leveraging powerful acoustic and language representations pre-trained with a large amount of data (Baevski et al., 2020, Liu et al., 2019a, Hinton et al., 2014, Seo et al., 2022, Raju et al., 2022, Rao et al., 2021, Wang et al., 2021).

While end-to-end SLU systems mitigate error propagation, the slot-filling task still relies on the correct recognition of the named entities in the speech, especially when the named entity contains rare words. Contextual biasing, as an effective method used in ASR to handle the long-tailed word problem, can also be applied to end-to-end SLU systems.

This chapter introduces the application of TCPGen in end-to-end SLU systems. TCPGen, together with GNN encodings, can be integrated into both sequence tagging-based SLU systems and sequence generation-based SLU systems: both of them are introduced in this chapter. TCPGen particularly benefits the model performance on slot-filling by integrating knowledge from a structured knowledge base associated with the slot-filling data, especially on rare and unseen entities in the training set. Moreover, TCPGen achieves zero-shot learning on unseen entities in both types of SLU system.

The rest of this chapter is organised as follows. Section 6.1 provides background for SLU tasks and knowledge integration in SLU. Section 6.2 introduces how TCPGen can be integrated into the tagging-based SLU system, with experimental results provided in Section 6.3. Section 6.4 introduces the knowledge-aware audio-grounded (KA2G) framework by integrating TCPGen into a sequence generation-based SLU system, with experimental results shown in Section 6.5. A chapter summary is provided in Section 6.6

6.1 Background

6.1.1 End-to-End SLU

In recent years, end-to-end SLU systems have tried to leverage external knowledge to achieve improved performance. Most research focused on using implicit knowledge from pre-trained representations, such as RoBERTa (Liu et al., 2019a) and wav2vec2.0 (Baevski et al., 2020). External knowledge from those models was integrated via knowledge distillation (Hinton et al., 2014) or network integration (Seo et al., 2022, Raju et al., 2022, Rao et al., 2021, Wang et al., 2021). Specifically, in (Rao et al., 2021) and (Seo et al., 2022), neural representations from the ASR model and text representations from a pre-trained LM were combined in the interface for the SLU tasks.

For slot-filling, recent research has seen increased interest in reformulating the task beyond standard sequence labelling and classification paradigms (Shah et al., 2019, Budzianowski and Vulić, 2019, Mehri and Eskenazi, 2021). For example, Namazifar et al. (2020) and Liu et al. (2022) recast slot filling as a QA task and a reading comprehension task, respectively, with both studies focusing on applications with limited data. More recently, Fuisz et al. (2022) performed a comprehensive analysis of the QA approach for slot filling and provided both efficient and effective fine-tuning methods for domain-specific slot-filling models.

Formulating slot filling as a text generation task has also recently become an active research area. (Mehri and Eskenazi, 2021) proposed a generative slot-filling framework that leverages pre-trained language models (PLMs) fine-tuned on specific tasks and domains to improve task-/domain-specific generation. Another research stream focused on framing dialogue state tracking (DST) for multi-turn task-oriented dialogue (ToD) as a text generation task. In particular, the T5DST model (Lin et al., 2021b) utilised different slot descriptions as the prompt for generation for cross-domain DST.

6.1.2 Knowledge Integration in End-to-End SLU

The research conducted by Chen et al. (2020a) involved encoding domain-slot relationships from the dialogue ontology using a graph neural network (GNN) to provide guidance for the system. Subsequently, Lin et al. (2021a) extended the use of GNNs to capture interdependencies in slot values across different domains. In the context of slot filling for task-oriented dialogue systems with speech input, Wang et al. (2022) employed a Transformer encoder to encode external knowledge into hidden representations, whereas Sun et al. (2023b) established a neural shortcut from the external knowledge base directly to the slot filling output. Both methods focused on zero-shot learning setups; however, they relied on the conventional sequence labelling formulation of the slot filling task.

6.2 TCPGen for Tagging-Based SLU

6.2.1 Motivation

Contextual biasing is a common technique employed in end-to-end ASR systems to improve the recognition accuracy of rare words. In SLU and spoken dialogue systems, a structured knowledge base (KB) can be used to collect possible named entities for each slot type. The biasing list can be extracted from the KB by selecting rare and unseen entities in relevant slots, thus improving the performance of end-to-end SLU systems. To generate the biasing list, a class language model (CLM) can be used to predict a shortlist of possible slot types for the upcoming words, which is referred to as the slot shortlist (SS). The possible entities for each slot type in the SS can then be retrieved from the KB and organised into the biasing list.

This chapter focuses on the TCPGen component as a contextual biasing mechanism for SLU. In the tagging-based system, TCPGen's shortcut to the ASR output can be expanded to include slot-filling output tags by transforming the distribution over wordpieces into a distribution over slot types. This approach is referred to as slot probability biasing (SPB). When combined with SS, TCPGen and SPB can enhance the performance of the tagging-based SLU system on rare and unseen entities. Furthermore, incorporating possible named entities of those types in the KB allows for zero-shot learning of unseen slot types.

6.2.2 Tagging-Based SLU System

The SLU system in this chapter is shown in Fig. 6.1, similar to Rao et al. (2021), Raju et al. (2022). The decoder hidden state sequence from AED was first sent through a bi-directional long short-term memory module. Then, to leverage external knowledge from a pre-trained

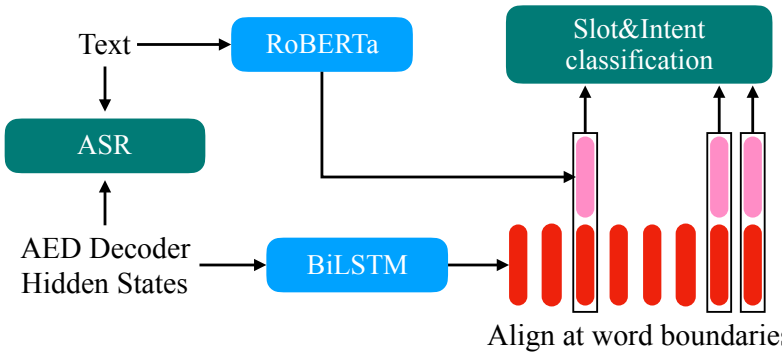


Fig. 6.1 End-to-end SLU system. The word-level alignment aligns the two sequences of representations at word boundaries.

representation, a sequence of RoBERTa output vectors was extracted and both sequences were aligned and concatenated at word boundaries. The concatenated vector sequence was used to perform slot classification at the boundary of each word. Intent classification only used the output at the final step. The AED, RoBERTa and BiSLTM modules were jointly optimised with the ASR, slot and intent classification tasks.

6.2.3 Slot Shortlist Prediction with a Class LM

TCPGen with slot shortlists (SS) used a more focused biasing list for better recognition. Specifically, a word-level causal CLM which predicts the class of the next word given word history was applied to predict a shortlist of the top N most probable slot types at the beginning of each word. The biasing list can be extracted by collecting entities belonging to those slots. However, when using GCN encodings during inference, as the SS kept varying during decoding, it was inefficient to encode the prefix tree repeatedly for every update of the biasing list. To address this, prefix trees were encoded offline for each slot type separately before decoding. The computation of the TCPGen distribution first computed the joint probability distribution over valid wordpieces and slot types and marginalised w.r.t. the slot types. This procedure is illustrated in Fig. 6.2.

As shown in Fig. 6.2, the number j on each node denotes a wordpiece and the encodings of each node, $\mathbf{e}_{m,j}$, were obtained by encoding the tree corresponding to slot m using a GCN. These node encodings were concatenated to form the keys and the same scaled dot-product attention in TCPGen was performed to calculate a distribution over these node encodings, as shown in Eqn. (6.1).

$$P^{\text{ptr}}(s, y_i) = \text{Softmax}\Big(\frac{\mathbf{q}_i^T[\cdots, \mathbf{e}_{m,j}, \cdots]}{\sqrt{d}}\Big) \tag{6.1}$$

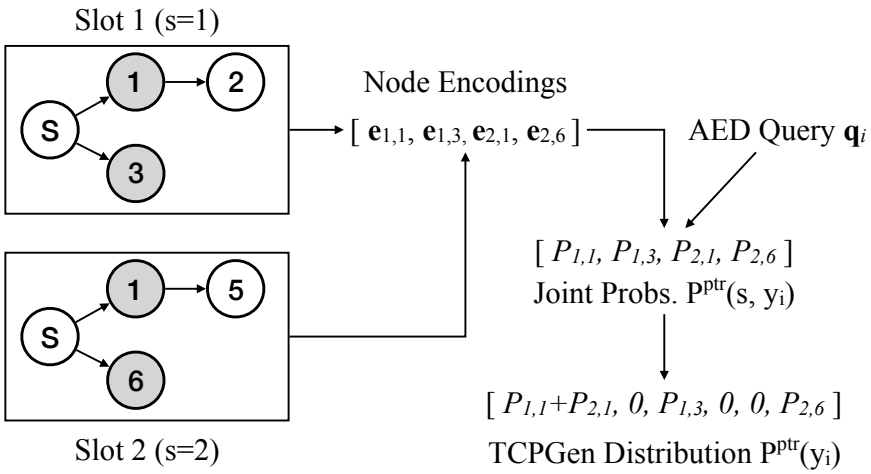


Fig. 6.2 Illustration of TCPGen with slot shortlist (SS). The example SS contains 2 slot types with their prefix trees each containing 2 entities. Nodes with grey fillings are the valid subset of wordpieces. Number 1 to 6 represents wordpieces. The current decoding step is i, $\mathbf{e}_{m,j}$ denotes wordpiece j on tree m, and $P_{m,j} = P^{\text{ptr}}(s = m, y_i = j)$

where $\mathbf{q}_i$ and d are the same as Eqn. (4.1). Finally, probabilities of nodes corresponding to the same wordpiece were summed to obtain the final TCPGen distribution as shown in Eqn. (6.2)

$$P^{\text{ptr}}(y_i) = \sum_{s \in \mathcal{S}} P^{\text{ptr}}(s, y_i) \tag{6.2}$$

where $\mathcal{S}$ denotes the set of slot types. This method is applied during inference. Moreover, as for TCPGen, TCPGen with SS can be generalised to work with phrases. When handling a biasing list of entities of more than one word, instead of obtaining a new shortlist at each word boundary, the shortlist is only updated at word boundaries where there are no valid paths on current prefix trees.

6.2.4 Slot Probability Biasing

A distribution over slot types can also be estimated from the joint distribution in Eqn. (6.1) by summing all of the node probabilities on each slot tree, as shown in Eqn. (6.3)

$$P^{\text{ptr}}(s) = \sum_{y_i \in \mathcal{Y}_s} P^{\text{ptr}}(s, y_i) \tag{6.3}$$

where $\mathcal{Y}_s$ is the set of valid wordpieces at step i on the tree corresponding to slot s. This probability was then interpolated with the original SLU model output slot probabilities, weighted by the generation probability P_i^{gen} indicating how likely the next wordpiece token

should be taken from the prefix trees, as shown in Eqn. (6.4).

$$P(s) = P^{\text{mdl}}(s) \times (1 - \alpha P_i^{\text{gen}}) + P^{\text{ptr}}(s) \times \alpha P_i^{\text{gen}} \tag{6.4}$$

where α is a hyper-parameter between 0 and 1 to restrict the influence of TCPGen as entities found in the biasing list are not always slot values. $P^{\text{mdl}}(s)$ is the original model output slot probability. This method was particularly beneficial to entities that are unseen in the training set, and it realised zero-shot learning of unseen slots by providing a list of possible entities.

6.3 Experiments for Tagging-based SLU

6.3.1 Experimental Setup

6.3.1.1 Data

Experiments were performed on the SLURP data (Bastianelli et al., 2020) in this chapter. SLURP is a collection of 72K audio recordings of single-turn user interactions with a home assistant, annotated with scenarios, actions and entities. Experiments were first performed using the official training, validation and test split, with synthesised audio used during training following (Arora et al., 2022a), to show the effectiveness of TCPGen on the standard SLU task. Additionally, a new split of the data was used by holding out utterances containing entities in five randomly selected types (`podcast_name`, `artist_name`, `audiobook_name`, `business_name`, `radio_name`) to evaluate zero-shot learning of unseen slot types. These utterances were mixed with an equal number of randomly selected utterances with seen slots or without slots to form the test set, and the rest of the utterances were used for training and validation. Moreover, the LibriSpeech 960-hour read English corpus was used to pre-train the ASR part of the system before training on SLURP.

6.3.1.2 Biasing List Extraction

The biasing list selection on LibriSpeech data followed the same procedure as used in Chapter 4. For SLURP, lists of slot entities in the data were categorised into their corresponding slots to form the KB. In this chapter, rare words were defined as words in the KB and appeared less than 30 times in the SLURP training set, which also includes unseen words. There were altogether 3k rare words. The *rare word biasing list* was organised by including rare words that appeared in the list of each slot, and the *rare entity biasing list* included entities containing rare words for each list. Note that unbounded slots such as date, time or frequency were not included in this biasing list as it was difficult and tedious to enumerate all possible

values. The size of rare word biasing lists ranged from 4 to 712 words, and the size of rare entity biasing lists ranged from 1 to 847 entities. For experiments on the new data split, all entities in the held-out 5 slots were used in the *rare entity biasing list* as none of them appeared in the training set.

6.3.1.3 Models and Evaluation Metrics

For both datasets, input features used 80-dimensional (-d) log-Mel FBANK features at a 10 ms frame rate concatenated with 3-d pitch features. SpecAugment Park et al. (2019) with the setting $(W, F, m_F, T, p, m_T) = (40, 27, 2, 40, 1.0, 2)$ was used for data augmentation.

The ASR model used was an AED model with an encoder with 16 conformer blocks (Gulati et al., 2020) with 512-d hidden state and 4-head attention, 1024-d single-head location-sensitive attention and a 1024-d single-layer unidirectional LSTM decoder. The BiLSTM module contained a 1024-d single-layer bi-directional LSTM. The RoBERTa base model with 768-d output representations was used. The CLM for SS prediction was a single-layer 2048d LSTM LM. AED models together with TCPGen were first trained on LibriSpeech 960-hour data for 20 epochs. The model parameters were then used to initialise relevant parts in the SLU system. During training, n slot types were selected by finding slots that appeared in the utterance annotation and adding distracting slots to match the n used during inference. Rare words belonging to those slot types were gathered to form the biasing list for training.

Models were evaluated using WER and rare word error rates (R-WER) defined in Chapter 4. R-WER is the rate of deletion, substitution and insertion of a rare word. For SLU, the SLU-F1 described in Section 3.5 was used to measure the slot-filling performance. The baseline is a pipeline system which takes the 1-best hypothesis from the AED model as the input to a tagging-based NLU system with RoBERTa (Liu et al., 2019a) as a backbone.

6.3.2 Results and Discussion

The performance of the end-to-end SLU system was evaluated on the official split of the SLURP dataset, and the results for WER, R-WER, and SLU-F1 are reported in Table 6.1. In the baseline model, the same RoBERTa model structure was used for the NLU component as in the end-to-end SLU system. The end-to-end SLU system outperformed the baseline model in terms of SLU-F1 as well as WER and R-WER. The improvement in performance can be attributed to the slot-filling task being included as a multi-task learning objective, which also facilitated the training of the ASR component.

Further reductions in WER, R-WER and SLU-F1 were achieved using TCPGen. When the full rare word list was used for biasing, a 13% relative R-WER reduction was achieved,

System	Biasing list	WER	R-WER	SLU-F1
Std. AED + NLU	N/A	12.7%	43.4%	77.8%
Std. SLU	N/A	12.6%	43.0%	78.4%
+TCPGen	full	12.4%	37.6%	78.9%
+TCPGen	top 1	12.1%	37.0%	79.1%
+TCPGen	top 2	**11.9%**	36.2%	**79.2%**
+TCPGen	top 5	12.0%	**35.6%**	79.1%
+TCPGen	top 10	12.1%	36.6%	79.0%
+TCPGen	top 2 entity	12.0%	36.8%	**79.2%**

Table 6.1 Results on the SLURP test set with the official split using TCPGen with different sizes of slot shortlist (SS) in SLU. Std. AED + NLU is the pipeline system. Full refers to using a single large biasing list containing all rare words, and top n refers to using biasing lists corresponding to the top n slot types. SPB was not used in this table. Entity refers to using the rare entity biasing list during inference. Improvements were statistically significant at $p \leq 0.01$.

which resulted in an increase of 0.5% in SLU-F1. This increase was mainly due to the correct recognition of rare words. Then, more focused biasing lists were obtained using the predicted slot shortlist. The best number of slot types to include in the SS was found by balancing the trade-off between the size and the coverage of biasing lists. Including more slot types increased the chance to cover a specific rare word, but also increased the size of the biasing list which can degrade performance. In Table 6.1, the best WER and SLU-F1 were achieved using the top 2 slot types, which gave a 16% relative R-WER reduction and an increase of 1.4% in SLU-F1 compared to the baseline. Although the top 5 achieved a lower R-WER, as the size of the biasing list increased, the degradation of recognition accuracy on common words reduced the overall performance. Moreover, using rare entity biasing lists achieved a very similar performance to rare word biasing lists.

The slot probability biasing (SPB) method enabled the system to further exploit rare and unseen words that were correctly recognised using TCPGen but were still misclassified by the SLU output, and gave an additional performance boost as shown in Table 6.2. By increasing α up to 0.5, the overall SLU-F1 increased by 0.3 compared to the best TCPGen SLU system without SPB, which led to an increase of 1.7% in the overall SLU-F1 score compared to the baseline. The main contributor to this improvement was the recall rate. However, further increasing α degraded the SLU-F1, as the recall rate stopped increasing while precision degraded. Moreover, the intent accuracy was also compared here to complete

System	α	SLU-F1 (unseen)	Intent Acc.
Std. AED + NLU	N/A	77.8% (50.5%)	87.9%
Std. SLU	N/A	78.4% (51.1%)	88.6%
+TCPGen	0.0	79.2% (54.8%)	88.7%
+TCPGen	0.5	**79.5%** (**57.5%**)	**88.9%**

Table 6.2 Results on the SLURP test set with the official split using SS and SPB in SLU, with the effects of different α values (see Eqn. (6.4)). Std. AED + NLU is the pipeline system. The top 2 slot types were used for biasing lists. Unseen referred to the SLU-F1 score measured on entities containing out-of-training-set words.

the SLU task set, and although TCPGen was not intended to directly help intent accuracy, the classification accuracy was also improved by 1.0% compared to the baseline.

The enhancement in performance of the TCPGen and SPB techniques, as evidenced by the SLU-F1 metric, can be primarily attributed to their performance in handling previously unseen entities. In order to elucidate the superiority of SPB when incorporated into TCPGen, particularly for unseen entities, a distinct SLU-F1 score was computed for those unseen entities, which constituted 5% of the entities in the entire test set. The optimal α value, which yielded the highest overall performance, was employed, and the results are presented in Table 6.2. By integrating SPB into TCPGen, there was a notable improvement of 7.0% in the F1 score for unseen entities. In contrast to the overall SLU-F1 score, the SLU-F1 score for these specific entities kept increasing as α increased, as this offered a boost to the recall rate associated with these entities.

System	α	SLU-F1 (unseen slots)
Std. AED + NLU	N/A	29.7% (0.0%)
Std. SLU	N/A	30.1% (0.0%)
+TCPGen	0.0	29.6% (0.0%)
+TCPGen	0.5	42.1% (36.6%)
+TCPGen	1.0	**52.0%** (**50.2%**)

Table 6.3 Results on the proposed held-out set with unseen slots. Std. AED + NLU is the pipeline system. TCPGen SLU in this table used biasing lists of all selected unseen slots. Unseen slots referred to the SLU-F1 on the unseen slot types only.

Finally, experiments were performed on the new split of SLURP for zero-shot learning, which had 5 unseen slots in the test sets, and the results are summarised in Table 6.3. For

systems without SPB, even if the entity was correctly recognised, the model could never classify it to the slot type that was not covered by the training set. Thus, the SLU-F1 was dominated by the performance on the training set slot types for those systems. Besides, as the biasing list here only contained entities from unseen slot types, TCPGen alone was not helpful for the training set slot types, and only obtained a similar overall performance as the baseline and the standard end-to-end SLU system. On the other hand, SPB drastically improved SLU-F1 scores by enabling the slot-filling output to handle entities of unseen slot types. As before, using the best value found on the validation set, $\alpha = 0.5$, achieved a 42% relative overall SLU-F1 increase, which also achieved an F1 score of 36.6% on unseen slots. Furthermore, if the proportion of unseen slots is known to be large, such as in the cross-domain application scenario, the value of α could be set higher. Eventually, with $\alpha = 1.0$, TCPGen with SPB achieved an F1 score of 50.2% on unseen slots.

6.3.3 Summary

This chapter has proposed the use of TCPGen in an end-to-end SLU system. TCPGen leverages a KB containing entities that were likely to appear in each slot type for contextual biasing. Besides, slot shortlists (SS) predicted by a class LM were used to obtain a more focused biasing list. Moreover, slot probability biasing (SPB) was proposed that estimates slot probabilities from TCPGen to bias the model slot prediction. Experiments on the SLURP data showed progressive improvements using TCPGen and SPB, with a large performance improvement in entities with unseen words. Intent detection accuracy was also improved with TCPGen. Furthermore, TCPGen with SPB achieved zero-shot learning, with an SLU-F1 score of 50% on unseen slot types following the new data split with held-out unseen slots in this chapter, compared to a zero F1 score for systems without SPB.

However, the clear drawback of SPB is the hand-tuned interpolation factor. With automatically predicted value of the shortcut, the performance was much worse as the SLU output is fairly confident about its prediction and hence it often ignores any additional information provided to it. The performance with the best hand-tuned interpolation factor is therefore brittle and inflexible. Moreover, it is not generalisable to other SLU systems, especially the nowadays popular sequence generation formulation of SLU. In the next few sections, a new framework that integrates knowledge using TCPGen into a sequence generation-based SLU system is described, which also achieves zero-shot learning of unseen entities while achieving superior performance on rare and unseen entities.

6.4 KA2G Framework for Slot Filling

6.4.1 Motivation

As manual fine-grained annotation for slot labels is expensive, time-consuming, and usually requires domain expertise (Casanueva et al., 2022), increasing demands have been put on the performance of slot-filling systems under *few-shot* or even *zero-shot* learning setups (Hou et al., 2020, Liu et al., 2020b, Henderson and Vulić, 2021). Following the now-prevalent use of large Transformer-based PLMs (Radford et al., 2019, Devlin et al., 2019, Raffel et al., 2019) for transfer learning across many NLP tasks, the PLMs have also been widely adopted in ToD for slot-filling tasks with limited labelled data (Chen et al., 2019a, Henderson and Vulić, 2021). The reformulation of slot-filling as a sequence generation task further exploits PLMs in low-data scenarios (Fuisz et al., 2022, Du et al., 2021, Campagna et al., 2020, Lin et al., 2021b, Namazifar et al., 2020, Liu et al., 2022, Hosseini-Asl et al., 2020, Madotto et al., 2020). However, all these approaches were purely based on text, where the influence of ASR on the slot-filling performance on infrequent named entities (e.g., atypical personal, restaurant or hotel names) is rarely analysed in the context of limited annotated data.

This chapter focuses on a particularly challenging setup which is typically met in production: limited annotated data with a multitude of rare entities. To this end, a Knowledge-Aware Audio-Grounded (KA2G) generative slot-filling framework is proposed and tailored towards improving the robustness and performance of slot-filling with speech input.

KA2G integrates information from both audio and text as input to a slot-value generator (SVG) which then generates fillers for each slot in the text modality. That is, the final generation is also grounded in the audio modality. This mitigates the issues arising from noisy ASR transcriptions. KA2G particularly boosts the performance of rare and unseen entities by learning to exploit available external knowledge (e.g., predefined lists of possible values for slots) via two TCPGen components (see Section 4.3). In short, the first TCPGen is applied on the ASR side to reduce the ASR errors on the high-valued biasing entities based on the context. The second TCPGen is applied on the SVG side to bias the generator's output using sub-trees which contain branches on the prefix-trees that are traversed during the ASR beam search. The entire KA2G model is jointly optimised in an end-to-end fashion from the input-speech-'end' to the generated slot-value-'end'.

6.4.2 Model Structure

The KA2G framework illustrated in Fig. 6.3 comprises the following three key parts:

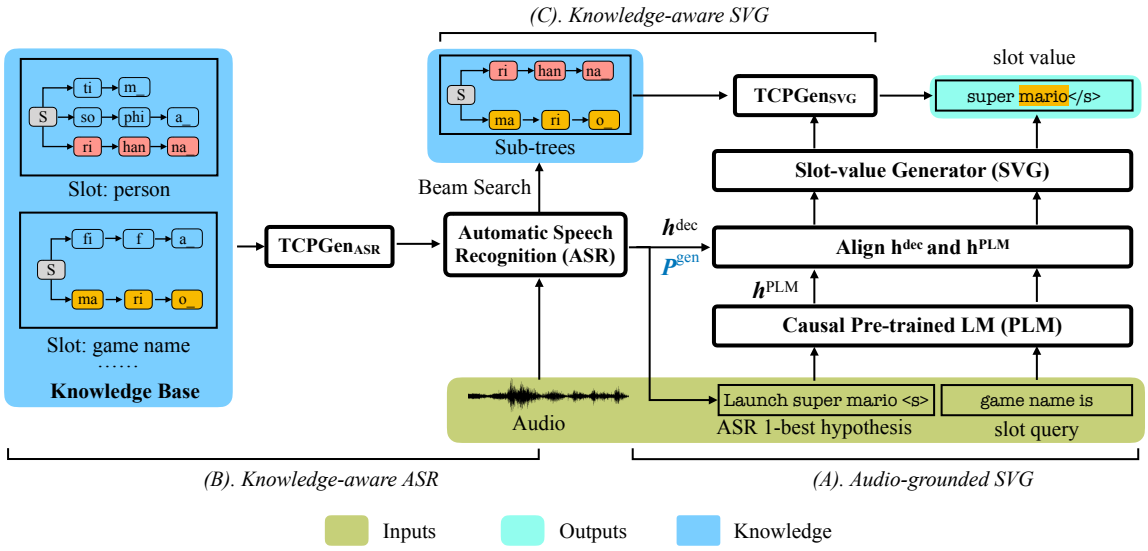


Fig. 6.3 The KA2G framework for slot filling. Its three key components are indicated by the labels (A), (B) and (C) and described in Section 6.4. The example in the figure shows that the first pointer generator network ($\text{TCPGen}_{\text{ASR}}$) traverses branches of `mario` and `rihanna`, which are then included in sub-trees. This branch is then further used by another generator network ($\text{TCPGen}_{\text{SVG}}$) when generating slot values.

(A) The audio-grounded SVG module that combines output representations from the ASR module and the text-only PLM.

(B) The knowledge-aware ASR component that integrates external knowledge into KA2G via the first TCPGen ($\text{TCPGen}_{\text{ASR}}$).

(C) The knowledge-aware SVG further that integrates knowledge explored during the ASR beam search via the second TCPGen ($\text{TCPGen}_{\text{SVG}}$).

Each component is described in the following Sections 6.4.2.1 to 6.4.2.3 respectively.

6.4.2.1 Audio-Grounded SVG

The audio-grounded SVG module, one of the key components of KA2G, comprises (i) an ASR model, (ii) a causal PLM, (iii) an alignment module, and (iv) the SVG; see the right side of Fig. 6.3. The SVG is implemented as a single-layer unidirectional LSTM which takes the concatenated vectors from the ASR module and the PLM as the representation of the context to make predictions for the value with a given slot query. The LSTM architecture is chosen for SVG here for simplicity and increased stability in low-resource setups, and to avoid over-parameterisation since both ASR and PLM have complex model structures with millions of parameters. The ASR model is an attention-based encoder-decoder (AED) where the decoder hidden states, $\mathbf{h}^{\text{dec}}$, are sent to the SVG.

As the label space for the PLM is overly sparse for the ASR model, the ASR model uses a much smaller set of subword tokens than the PLM, and hence the sequence of ASR hidden states and the sequence of PLM output vectors $\mathbf{h}^{\text{PLM}}$ will be asynchronous. To resolve this *(mis)alignment issue*, the output of the SVG is first set to have the same subword tokens as the ASR model, which helps to make the best use of the acoustic information. The PLM outputs are then taken per each (full) word instead of every subword at the ending of a word and aligned with $\mathbf{h}^{\text{dec}}$ at each ending subword before concatenation. For non-terminal subwords, zero-vector padding is used as placeholders for the PLM outputs. In this way, the alignment of $\mathbf{h}^{\text{dec}}$ and $\mathbf{h}^{\text{PLM}}$ are aligned at word ends, and using the same subwords for ASR and PLM is not necessary. Moreover, for a slot query which prompts the generation (e.g. `the person is` or `the game name is`, see Fig. 6.3) which does not have any corresponding input audio, the embedding of the preceding wordpiece is used instead of $\mathbf{h}^{\text{dec}}$. An example of this alignment is shown in Fig. 6.4.

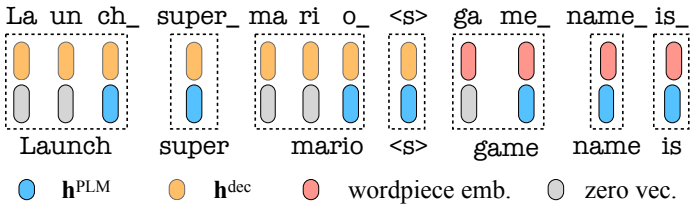


Fig. 6.4 An example of aligning $\mathbf{h}^{\text{PLM}}$ and $\mathbf{h}^{\text{dec}}$.

The SVG is trained in an end-to-end manner by jointly optimising the ASR and SVG criteria:

$$\mathcal{L}_{\text{joint}} = \mathcal{L}_{\text{ASR}} + \mathcal{L}_{\text{SVG}}, \tag{6.5}$$

where the respective sub-losses are defined as

$$\mathcal{L}_{\text{ASR}} = \log P(\mathbf{y}_{1:n}|\mathbf{x}_{1:T}), \tag{6.6}$$

$$\begin{aligned}\mathcal{L}_{\text{SVG}} &= \log P(\mathbf{s}_{1:m}|\mathbf{q}_{1:k}, \mathbf{x}_{1:T}) \\ &= \log \left(\mathbb{E}_{\mathbf{y}_{1:n}}[P(\mathbf{s}_{1:m}|\mathbf{y}_{1:n}, \mathbf{q}_{1:k}, \mathbf{x}_{1:T})]\right) \\ &\approx \log P(\mathbf{s}_{1:m}|\mathbf{q}_{1:k}, \mathbf{h}^{\text{dec}}, \mathbf{h}^{\text{PLM}})\end{aligned} \tag{6.7}$$

Here, $\mathbf{y}_{1:n}$ is the subword ASR sequence and $\mathbf{x}_{1:T}$ is the sequence of acoustic features. The second loss, $\mathcal{L}_{\text{SVG}}$, which is expressed as the expectation over all possible ASR hypotheses, is approximated by the decoder hidden states of the ASR module. The slot query $\mathbf{q}_{1:k}$ containing the slot type information is also encoded by the PLM. As the SVG operates on a different set of subwords, these PLM encodings are fed into the SVG at word ends, similar to

the aforementioned encoding of the context, to generate the value sequence $\mathbf{s}_{1:m}$ (e.g., `super mario`). In order to allow the model to also handle predictions for slots not present in the utterance, N_n (randomly sampled) slots that are not mentioned in the context are incorporated in training as negative examples, where N_n is a hyper-parameter. The model should learn to generate `None` values for the 'non-present' slots.

During inference, $\mathbf{h}^{\text{PLM}}$ is obtained by feeding the 1-best ASR hypothesis to the PLM. The same context is prompted with all possible slot types, and those that do not output a `None` value are saved. For multi-turn ToD, the dialogue history is encoded by the PLM before the start of the current context.

6.4.2.2 Knowledge-Aware ASR

The knowledge-aware ASR part is essentially the AED ASR model with TCPGen described in Section 4.3, in conjunction with slot shortlist prediction from Section 6.2. External knowledge is organised as a dictionary of slots along with the possible values per slot to form a knowledge base (KB). The KB conditions ASR via contextual biasing using $\text{TCPGen}_{\text{ASR}}$ (as shown in Fig. 6.3). In slot filling, possible named entities for each slot type can be collected to form a structured KB, and the biasing list can be extracted from the KB. TCPGen, as described in previous chapters, is used for contextual biasing in the knowledge-aware ASR module. An example of a prefix tree is shown on the left side of Fig. 6.3. During ASR decoding, a set of valid wordpieces is obtained by searching the prefix tree with the decoded preceding wordpieces. Then, scaled dot-product attention is performed to obtain the $\text{TCPGen}_{\text{ASR}}$ distribution $P^{\text{ptr}}(y_i)$ as described in Eqn. (4.1). The key and value vectors are the graph convolutional network (GCN) node encodings (Kipf and Welling, 2017), as described in Section 5.3.2 of corresponding subwords on the prefix tree. The generation probability is calculated using the decoder hidden state and the weighted combination of node encodings from the attention. The final interpolation is computed as Eqn. (4.3).

Possible entities for each slot are structured as prefix trees separately; see Fig. 6.3. To have a more focused biasing list, instead of using all slots, the slot shortlist method described in Section 6.2 is also adopted here. Moreover, as the generation probability P^{gen} provides the indication of how much contextual biasing is needed to decode each subword token, it is concatenated with $\mathbf{h}^{\text{dec}}$ and sent to the SVG to further indicate where in the context the knowledge has been used. $\text{TCPGen}_{\text{ASR}}$ is jointly optimised with the SVG module.

6.4.2.3 Knowledge-Aware SVG

Information about alternative hypotheses is an essential resource that can be obtained from the ASR system, especially for rare named entities. To exploit the knowledge available in the additional ASR hypotheses, the knowledge-aware SVG module is proposed: it extracts branches on each prefix-tree during ASR beam-search decoding and forms sub-trees to be used by $\text{TCPGen}_{\text{SVG}}$ (as shown in Fig. 6.3) to integrate knowledge into SVG.

In particular, as each prefix-tree used in the ASR beam search decoding is searched, a valid path that leads from the root node S to a leaf node will be saved, which corresponds to a valid named entity belonging to that slot type. After decoding, the lists of entities corresponding to the valid paths found for each slot are gathered and organised into prefix-trees that are essentially sub-trees of the original trees. Following this, sub-trees are encoded using the same GCN as in 6.4.2.2 and are searched when generating slot values in the same way as with $\text{TCPGen}_{\text{ASR}}$. For $\text{TCPGen}_{\text{SVG}}$, the query comes from the SVG hidden state at each decoding step, and keys and values are taken from the node encodings on the sub-trees. In this way, possible entities that are not covered by the 1-best hypothesis but are explored in the ASR beam search can be effectively used via the copy mechanism in SVG.

6.5 Experiments to Evaluate KA2G

6.5.1 Experimental Setup

6.5.1.1 Training and Evaluation Data

Experiments were performed on two structurally different speech-based English ToD datasets. The first dataset was SLURP which was explained in detail in Section. 6.3. Experiments were run in two data setups. 1) The official training, validation and test split was used and following Arora et al. (2022a) where synthesised audio was also used for training. 2) Zero-shot setups following the data split in Section 6.3 were adopted here: in training, all utterances containing entities of five randomly selected 'unseen' slots were held out and were used for testing.

Experiments were also performed on an industry dataset, termed CONCIERGE. This was a multi-turn dialogue dataset obtained from a commercial system provided by PolyAI Ltd, and it represented a standard and challenging few-shot learning setup typically met in production. The dataset contained a collection of 8 kHz noisy phone-call conversations of real customer interactions with a concierge voice bot covering the `restaurant`, `shop` and `bar` domains. The audio was upsampled to 16 kHz to match the ASR model. Example

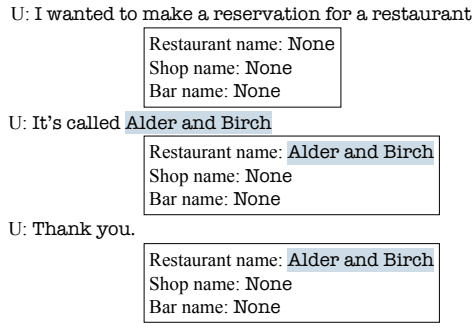


Fig. 6.5 An example dialogue from the CONCIERGE dataset along with the state tracking labels. The slot filling label for the restaurant name in the last turn is `None`

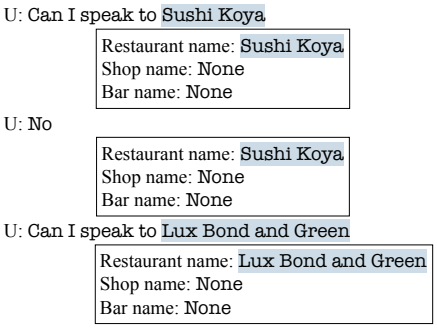


Fig. 6.6 Another example dialogue from the CONCIERGE dataset along with the state tracking labels. The user changed the goal in the third turn.

dialogues from CONCIERGE can be found in Fig. 6.5 and 6.6. Some dataset statistics are given in Table 6.4. Each entity in the test set had only 1 to 5 occurrences in the training portion of the dataset.

Split	# Turns	# Dialogues	Time (hours)
Train	1934	829	3.17
Valid	212	97	0.36
Test	428	225	0.86

Table 6.4 Statistics of the CONCIERGE dataset.

6.5.1.2 Model Specifications

The ASR model used an encoder with 16 Conformer blocks (Gulati et al., 2020) with a 512-dim hidden state and 4-head attention, a 1024-dim single-head location-sensitive attention and a 1024-dim single-layer unidirectional LSTM decoder. The input signals were 80-dim FBANK and a suffix-based unigram WordPiece model (Whittaker and Woodland, 2000, Kudo, 2018) with 600 distinct WordPieces was used for the output.

GPT-2 (Radford et al., 2019) with 768-dim output representations was used as the causal PLM in all experiments. Both TCPGen components adopted one 256-dim single-head attention layer and a two-layer 256-dimensional GCN was used with the input node encodings being the WordPiece embeddings of the ASR decoder, based on the same setup as Section 6.3. The dimensions of the SVG and projection layers in TCPGen were all determined by

the mentioned dimensionalities. The CLM used to predict a shortlist of slots for SLURP experiments was a single-layer 2048-dim LSTM.

The proposed KA2G framework was compared to a pipeline system, which used the same ASR model to get the 1-best hypothesis as input text to the GPT-2 model to perform slot-value generation. The GPT-2 model was finetuned in the same way as KA2G.

6.5.1.3 Training and Inference Specifications

The ASR AED models, together with the ASR-TCPGen component, were pretrained for 20 epochs on LibriSpeech's 960-hour English audiobook data following Section 6.3. When training SVG (see 6.4.2.1), $N_n = 10$ negative 'non-present' slots were randomly chosen and added for each utterance in SLURP, and $N_n = 3$ for CONCIERGE. Experiments were trained on a single A100 GPU.

Both TCPGen components were trained in the same way as in Section 6.3, where a full biasing list was first defined for each dataset, and the biasing list for each utterance was organised by selecting biasing entities in the reference transcription and adding a certain number of distractors following Le et al. (2021a). For SLURP, lists of slot values/entities in the data were first categorised into their corresponding slots to form the KB. The full biasing list was defined by selecting entities in the KB with fewer than 30 examples in the training set, which also included unseen entities. There are altogether 5,000 biasing entities. For $\text{TCPGen}_{\text{ASR}}$, the number of distractors was randomly picked between 100 and 200, whereas for $\text{TCPGen}_{\text{SVG}}$, 20 distractors from the same slot type were used, which was closer to the size used in inference. The random dropping of the reference biasing entities was also applied to $\text{TCPGen}_{\text{SVG}}$. The same full biasing list selection criterion, as well as the training procedure for TCPGen, was applied to the CONCIERGE data.

During inference, a beam size of 30 was used for ASR decoding following prior work. For SLURP, entities in the KB with fewer than 30 examples in training were used as biasing entities, and the CLM predicts the top 2 slot types at each word boundary. For the zero-shot learning setup on SLURP, the biasing list with 2k entities included all entities in the unseen slots since they were all unseen entities, and were used as a whole during inference, as the CLM could not predict unseen slot types. For few-shot learning on CONCIERGE, all entities in the test set were included in the biasing list since they all appeared less than 5 times, which formed a biasing list of 105 entities. Since this was a much smaller biasing list, the entire list was used without CLM prediction. Entities that appeared less than 5 times are referred to as *few-shot entities*. For the CONCIERGE data, all biasing entities were few-shot entities. Unless stated otherwise, greedy decoding was used for SVG.

6.5.1.4 Evaluation Metric

ASR models were evaluated using the standard WER measure. For slot filling, the SLU-F1 (Bastianelli et al., 2020) and the micro Entity-F1 scores were used to measure performance, offering insights into both word-level and character-level F1 scores. Moreover, for multi-turn dialogues, JGA was reported: it counted a turn as correct only if all slots in the dialogue state were correctly filled in that turn.

6.5.2 Results and Discussion on SLURP

System	WER (%) ↓	Overall SLU-F1 (%) ↑
Pipeline	12.7	79.2 (73.0)
Audio-grounded SVG	12.5	79.6 (73.9)
Pipeline + Contextual ASR	12.4	79.7 (73.7)
Full KA2G	**12.1**	**80.6 (75.1)**

Table 6.5 WER, SLU-F1, and Entity-F1 (in parentheses) scores on SLURP. F1 scores were measured for all entities (*Overall*). *Pipeline* represented the pipeline system using the same AED-based ASR model to get the 1-best hypothesis, followed by GPT-2 for the slot-value generation. *Contextual ASR* used $\text{TCPGen}_{\text{ASR}}$ in the pipeline system.

System	$0 < \text{f} < 30$ (%)	$0 < \text{f} < 5$ (%)	$\text{f} = 0$ (%)
Pipeline	72.4 (69.5)	69.2 (69.1)	21.1 (7.2)
Audio-grounded SVG	74.3 (71.8)	72.2 (71.9)	20.9 (6.5)
Pipeline + Contextual ASR	73.5 (70.2)	70.8 (70.5)	24.5 (9.8)
Full KA2G	**75.7 (73.4)**	**73.8 (73.7)**	**32.3 (18.6)**

Table 6.6 WER, SLU-F1, and Entity-F1 (in parentheses) scores on SLURP. F1 scores were measured for biasing entities (occurrence frequency $f < 30$), few-shot entities ($f < 5$) and unseen entities ($f = 0$). *Pipeline* represented the pipeline system using the same AED-based ASR model to get the 1-best hypothesis, followed by GPT-2 for the slot-value generation. *Contextual ASR* used $\text{TCPGen}_{\text{ASR}}$ in the pipeline system.

The results on SLURP were summarised in Table 6.5 and Table 6.6. The scores revealed that, compared to the pipeline system, using the audio-grounded SVG model (Row 2 in the table) achieved better performance overall, with much better scores on biasing entities (*i.e.*, their occurrence frequency was $0 < f < 30$) and few-shot entities ($0 < f < 5$). Similar

improvements from audio-grounding were observed when comparing the full KA2G framework to the pipeline system with a contextual ASR using TCPGen. Overall, KA2G achieved a 1.4% absolute SLU-F1 increase compared to the baseline pipeline system, with a 4.6% SLU-F1 improvement on few-shot entities and an 11.2% SLU-F1 improvement on unseen entities. This indicated that audio grounding helped the system to better deal with few-shot entities as long as at least some examples were exposed to it. Furthermore, similar to Le et al. (2022), the model was also able to generate entities that were incorrectly recognised by ASR: for the audio-grounded system, 10% of the correctly filled entities were not correctly recognised by the ASR system, whereas that ratio was only 3% for the pipeline system.

Case 1:

Reference transcription:	launch super mario
ASR transcription:	lunch super mario
Slot-value (Pipeline):	meal_type: lunch
Slot-value (Audio-grounding):	game_type: super mario
Slot-value (Ground truth):	game_type: super mario

Case 2:

Reference transcription:	.....song from the album abbas
ASR transcription:	.....song from the album abs
Slot-value (Pipeline):	music_album: abs
Slot-value (Audio-grounding):	music_album: abbas
Slot-value (Ground truth):	music_album: abbas

Fig. 6.7 Two examples of utterances from SLURP where audio-grounding helps slot filling.

Two example cases where the pipeline system and audio-grounded SVG have the same ASR output are shown in Fig. 6.7. In Case 1, while the semantic meaning of "launch" and "lunch" was completely different, their pronunciations were similar. Therefore, "lunch" as an ASR error confused the pipeline system to produce the wrong slot type and value, which was not fixable in the SVG part since the audio information was lost in the pipeline system. However, with audio-grounding, the pronunciation similarities of "launch" and "lunch" could be considered by the SVG with the strong GPT-2 based PLM, which therefore successfully directed the model to look at the correct entity.

In Case 2, although "abs" was in the ASR output, the ASR was uncertain about this prediction and "abbas" also had a high ASR score. Therefore, while both systems were able to fill the correct slot type, the audio-grounded SVG was able to replace the entity with the one seen in the training set that had similar pronunciation. This also explained why the pipeline-based baseline performed slightly better than the audio-grounded SVG baseline on unseen entities (see the last column of the first two rows in Table 6.6).

When $\text{TCPGen}_{\text{ASR}}$ was used (Row 3 in the table), the main performance boost of the system was due to a better WER. The improvements in WER and F1 scores were mainly on

biasing and unseen entities as those were included in the biasing list for TCPGen. Finally, the full KA2G framework achieved higher overall F1 scores compared to the pipeline system, and particularly prominent gains were observed on few-shot entities and unseen entities. Moreover, benefiting from multi-task training, KA2G achieved a lower overall WER: this reflected the fact that the slot-filling task could positively impact the ASR performance. Further, note that the WER on words in the biasing entities was similar to that of the pipeline system: this indicated that the gain with KA2G on low-frequency entities did not only come from improved ASR.

6.5.2.1 Comparison to Baseline from Prior Work

The baseline pipeline system with the overall SLU-F1 score of 79.2 was taken from the best system in Table 6.1. Further, Arora et al. (2022b) adopted a sequence labelling approach, where an overall SLU-F1 score of 78.0 was reported on SLURP (*cf.*, 80.6 reported with KA2G in Table 6.5). Note that the work of Arora et al. (2022b) used the more powerful WavLM pretrained speech representations (Chen et al., 2022), resulting in a much lower WER of 9% with an Entity-F1 of 75.7. Concerning generation-based slot filling systems, Wang et al. (2021) achieved a score of 78.9 using a sequence generator with wav2vec 2.0 representations.

6.5.2.2 Few-Shot Versus Zero-Shot

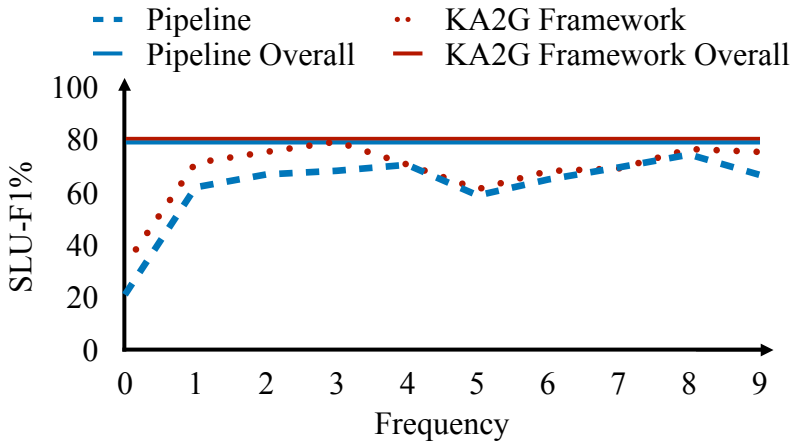


Fig. 6.8 SLU-F1 (%) over different training set occurrence frequencies for entities in SLURP. *Overall* SLU-F1 scores were also provided as horizontal lines.

The preliminary comparison of results between few-shot and unseen entities from Table 6.5 already revealed that even when only provided a handful of examples the model

was able to achieve a large leap in performance. To provide more insights, a finer-grained frequency bin analysis was conducted on SLURP as shown in Fig. 6.8. The scores revealed that there was a very large increase in SLU-F1 after only providing a single sample for an entity (i.e., moving from zero-shot to one-shot), increasing scores from ∼30 to ∼70: This corroborated the major benefits of few-shot learning over zero-shot learning (Lauscher et al., 2020). Fig. 6.8 again validated that KA2G provided improvements over the baseline system in such low-resource scenarios.

6.5.2.3 Ablation Study

System	**Overall** (%)	$0 < \text{f} < 5$ (%)	$\text{f} = 0$ (%)
KA2G framework	**80.6 (75.1)**	**73.8 (73.7)**	**32.3 (18.6)**
without $\text{TCPGen}_{\text{SVG}}$	80.6 (74.7)	73.6 (73.5)	27.5 (8.1)
without $\text{TCPGen}_{\text{ASR}}$	79.7 (74.3)	72.5 (72.3)	27.4 (13.5)
without $\text{TCPGen}_{\text{SVG}}$ and P^{gen} input	80.2 (74.5)	73.1 (73.0)	24.4 (7.5)
without $\text{TCPGen}_{\text{SVG}}$ and $\text{TCPGen}_{\text{ASR}}$	79.6 (73.9)	72.2 (71.9)	20.9 (6.5)
KA2G framework + SVG beam search	**80.6 (75.2)**	**74.2 (74.1)**	**32.4 (18.8)**
without $\text{TCPGen}_{\text{SVG}}$ and $\text{TCPGen}_{\text{ASR}}$	79.6 (73.9)	72.6 (72.2)	21.1 (7.2)

Table 6.7 An ablation study on SLURP based on SLU-F1 (Entity-F1 in parentheses). The first row referred to the complete KA2G framework, and each subsequent row represented removing the corresponding components.

The results of the KA2G ablation experiments are given in Table 6.7, where the last row in the table corresponded to the system which uses the audio-grounded SVG (i.e., the second row in Table 6.5). By removing the $\text{TCPGen}_{\text{SVG}}$, the most significant change was the decrease in performance on unseen entities, especially in terms of Entity-F1. When $\text{TCPGen}_{\text{SVG}}$ was included, the SVG was fully guided by the biasing entities, and hence the model was more likely to predict complete entities. This observation was also found by comparing the system without $\text{TCPGen}_{\text{ASR}}$ to the system without $\text{TCPGen}_{\text{SVG}}$. Moreover, since the WERs for unseen entities were much higher for the 1-best hypothesis, extracting information from alternative hypotheses was much more useful for unseen entities.

While $\text{TCPGen}_{\text{ASR}}$ contributed to the performance improvement, the use of the copy probability P^{gen} also contributed to the improvement, especially for rare and unseen entities, as it provided the indication of where knowledge had been used. This was particularly useful when the entity was correctly recognised but SVG was not able to generate it due to not seeing enough examples.

The beam search can be performed for both ASR and SVG. For ASR, it was found that a higher beam size only yielded very marginal improvements in WER, while taking significantly more time for decoding. From the perspective of the biasing lists for $\text{TCPGen}_{\text{SVG}}$, in the one-best hypothesis, only 50% of entities could be found. With a beam size of 30, 70% of entities were covered in the biasing lists for $\text{TCPGen}_{\text{SVG}}$, which was the main source of improvement for $\text{TCPGen}_{\text{SVG}}$. However, this was only improved by 2% using a beam size of 100, whereas the average size of the biasing lists increased from 10 to 15 which introduced more noise into the biasing lists. Therefore, a beam size of 30 was used for the rest of the experiments. On the SVG side, beam search only provided a marginal improvement, but the time for decoding was 4–5 times longer as the lengths of alternative slot values were in general longer than `None`.

6.5.2.4 Impact of Training Data Size

The impact of the SLURP training data size on the performance of few-shot entities was also studied. To this end, the 2.6k utterances which contain few-shot entities were retained while the rest of the training set was subsampled to a certain proportion. These subsets were then used to train the pipeline system and the full KA2G framework. The same training subset was used to train the ASR model (with TCPGen). The results are shown on the left part of Fig. 6.9. An additional experiment, where the ASR module was first trained on the full training set to maintain the same ASR performance is shown on the right part of Fig. 6.9.

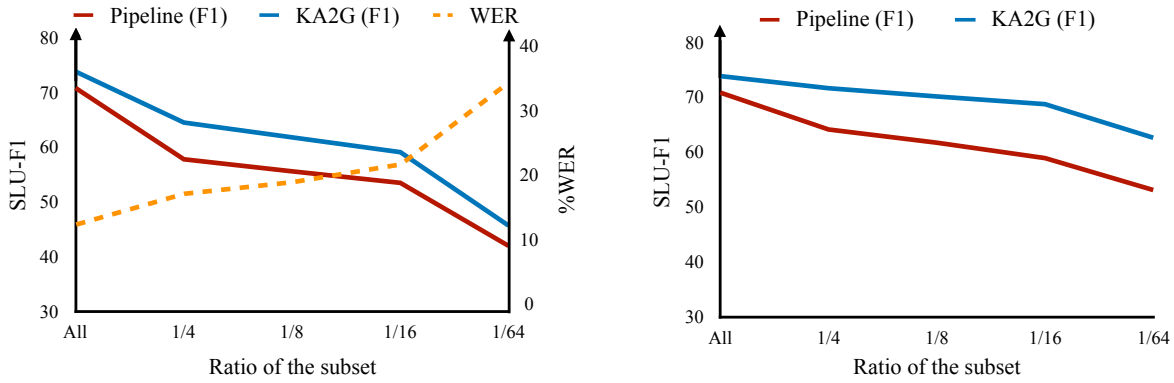


Fig. 6.9 SLU-F1 on few-shot entities when subsampling the part of the training set not containing few-shot entities. (a). The ASR component in the pipeline and KA2G systems were trained on the same subset. (b). The ASR was first trained on the full SLURP training set but SLU was only trained on the selected subset, where WER for all points is $\sim$ 13%.

Note that reducing other training data did have a strong impact on SLU-F1 scores of few-shot entities despite the fact that the utterances covering those entities were fully retained. The

full KA2G again consistently outperformed the baseline system across different training data sizes. The performance degradation with the degree of subsampling was even smaller with the ASR module trained on the full data, indicating the SVG module was more competent in few-shot learning than the ASR module.

6.5.2.5 SLURP: Zero-Shot Setup

The results of the zero-shot setup are summarised in Table 6.8. The WER for the zero-shot learning test set was 18.0% for ASR without TCPGen, which decreased to 16.9% for the KA2G. Compared to the pipeline system, KA2G achieved worthwhile improvements both in SLU-F1 and Entity-F1 scores. Therefore, KA2G provided an effective way of leveraging knowledge for zero-shot slot filling by bridging the SVG and ASR alternative hypotheses via a neural shortcut.

System	SLU-F1 (%)	Entity-F1 (%)
Pipeline	10.1	3.7
KA2G framework	**23.7**	**12.9**
w/o TCPGen	9.3	3.0

Table 6.8 SLU-F1 and Entity-F1 scores for unseen slots under the zero-shot learning setup on SLURP.

Note that the audio-grounded SVG module in KA2G alone was inherently able to handle zero-shot slot types as the slot types were fed into the model via natural language queries. Although the zero-shot performance of KA2G was slightly worse than the performance in Table 6.3, the way KA2G handled zero-shot entities in a full neural way did not require any manual settings of hyper-parameter. KA2G with TCPGen can be regarded as a more robust zero-shot learning framework.

6.5.3 Results and Discussion on CONCIERGE

6.5.3.1 Single-Turn Evaluation

The CONCIERGE dataset was used to validate the proposed KA2G framework in a real-world use case, and the results were discussed as follows. The WER on this test set was $\sim$35% due to limited audio data for training, which made the slot-filling task with speech input on CONCIERGE even more challenging. Single-turn evaluation was first performed, with the same metrics as on SLURP. The results were provided in Table 6.9.

System	SLU-F1 (%)	Entity-F1 (%)
Pipeline	37.2	26.9
KA2G framework	**56.6**	**42.5**
w/o TCPGen	41.9	29.3
Pipeline zero-shot	1.6	1.6
KA2G zero-shot	12.4	2.6

Table 6.9 SLU-F1 and Entity-F1 scores on CONCIERGE. Zero-shot results were obtained by removing dialogues containing test set entities from training.

For the challenging CONCIERGE dataset, external knowledge now played an even more important role, which led to much higher performance improvements with the KA2G framework. As with SLURP, having a few examples for entities yielded much better performance than zero-shot learning: put simply, providing a handful of examples in the training set resulted in huge performance improvements. This again corroborates the fact that although zero-shot learning was an attractive research problem, few-shot learning was often more pragmatic for industrial applications, also when using generative systems such as KA2G.

6.5.3.2 Multi-Turn Evaluation

For multi-turn experiments, entity mapping was applied in order to group different expressions of the same entity together. Further, the ASR 1-best hypothesis from the history of user inputs was included as input to the PLM. The JGA scores were summarised in Table 6.10. It was observed that large improvements were obtained with the full KA2G, which were mostly due to the use of the TCPGen component.

System	JGA (%)
Pipeline	21.0
KA2G framework	**41.5**
w/o TCPGen	24.3

Table 6.10 JGA on the CONCIERGE dataset with multi-turn dialogue state tracking.

6.5.4 Conclusion

Sections 6.4 and 6.5 introduced the knowledge-aware audio-grounded (KA2G) generative slot-filling framework for speech-based ToD. The framework is especially suited for low-

resource slot-filling tasks and for handling rare and unseen entities/values. KA2G comprises an audio-grounded SVG, together with two TCPGen components. The first TCPGen integrates knowledge from an external knowledge base containing possible entities for all slots into ASR, while the second TCPGen exploits entities found in alternative ASR hypotheses. Comprehensive evaluations have been performed on two different datasets with speech input: i) single-turn SLURP data and ii) multi-turn CONCIERGE data obtained from a commercial ToD system. It empirically validated the usefulness of KA2G on both datasets, with clear performance gains achieved over current state-of-the-art systems. It was also verified that KA2G is especially useful in few-shot and zero-shot setups.

6.6 Summary

The key findings and contributions described in this chapter are summarised as follows:

- The tree-constrained pointer generator (TCPGen), as a neural contextual biasing component, was extended to be applied to spoken language understanding (SLU) tasks, especially for slot filling.
- A more focused biasing list was obtained by leveraging the structured knowledge base (KB) associated with the slot filling task. Slot shortlists (SS) could be predicted by a class LM (CLM) and possible named entities associated with each slot type were gathered together to form the focused biasing list.
- For tagging-based SLU systems, a slot probability biasing (SPB) mechanism was integrated at the SLU output which transformed the TCPGen distribution into a distribution over slot types.
- TCPGen with SS and SPB for tagging-based SLU systems achieved significantly better performance on the SLURP dataset measured by SLU-F1, especially on rare and unseen entities. Moreover, SPB achieved zero-shot slot filling for unseen slot types with a manually adjusted interpolation factor for biasing.
- For sequence generation-based SLU, a knowledge-aware audio-grounded (KA2G) generative slot-filling framework was proposed, which contained an ASR module and a slot-value generation (SVG). The SVG module took vector outputs from the ASR module as inputs and generated the slot values in natural language according to natural language queries for a specific slot type. This inherently handled unseen slot types.

- A stacked TCPGen was proposed in KA2G where one TCPGen was integrated into the ASR module and another, sharing the same tree encoding parameters, was integrated into the SVG module. The TCPGen at SVG integrated knowledge explored in the ASR beam search procedure and achieved much better performance on rare and unseen entities and slot types.

In general, TCPGen has the potential to serve as an explicit knowledge integration component in any language understanding task. In particular, the stacked TCPGen mechanism is generally applicable to any language understanding task involving the speech modality, such as spoken question answering, audio captioning and summarization, etc. In the era where large LMs (LLMs), such as ChatGPT and GPT-4 are dominating techniques which have been found to be deficient in factual and dynamically changing external knowledge, TCPGen-type knowledge integration has the potential to serve as an alternative and complementary knowledge integration component in addition to in-context learning. Moreover, explicit knowledge integration also enjoys the benefit of improved interpretability, controllability and reliability than any other implicit integration approach for language understanding.

Chapter 7

Conclusions and Future Work

This thesis focuses on contextual knowledge integration in end-to-end automatic speech recognition (ASR) and spoken language understanding (SLU) tasks. The thesis began with the introduction to deep learning in Chapter 2, with various building blocks explained, followed by the introduction to end-to-end ASR and SLU systems in Chapter 3. Then, the novel tree-constrained pointer generator (TCPGen) component was introduced in Chapter 4 together with dedicated training and decoding algorithms for ASR tasks. In Chapter 5, TCPGen was further equipped with graph neural network (GNN) prefix-tree encodings to achieve better performance in ASR. TCPGen was also integrated into SLU systems in Chapter 6. Based on all the aforementioned contributions, this chapter provides a summary of key conclusions drawn from this thesis and offers promising prospects for future work.

7.1 Conclusions

End-to-end ASR and SLU systems often suffer from the “long-tailed words" problem. Contextual biasing aims at addressing this problem by integrating dynamic contextual knowledge carrying information about those rare words that are likely to appear in a given context into the static end-to-end systems. This thesis discusses how such dynamic contextual knowledge is effectively integrated via the TCPGen component.

Chapter 4 introduced TCPGen as a neural biasing component. In contrast to other pre-existing methods, TCPGen combines the advantage of shallow fusion and deep biasing by building a neural shortcut that directly modifies the model output distribution while being end-to-end trainable and aware of the hidden states from various parts of the ASR system. TCPGen leverages a prefix tree to structure the biasing list, which achieves high efficiency in handling biasing lists containing thousands of words or phrases. As a generic biasing

component, TCPGen has been successfully integrated into the attention-based encoder-decoder (AED) model and the recurrent neural network transducer (RNN-T) model.

The minimum biasing word error (MBWE) training algorithm for training was also proposed together with the biasing-word-driven language model discounting (BLMD) method for inference. BMWE directly optimises the expected word error rate, with a particular focus on the errors in the biasing words. BLMD takes care of the TCPGen distribution separately from the original model distribution with a different set of hyper-parameters for discounting.

In Chapter 4, experiments were performed on the LibriSpeech audiobook corpus and the augmented multi-party interaction (AMI) meeting corpus following the validated simulation of contextual biasing on open datasets (Le et al., 2021a). TCPGen was also evaluated on the DSTC spoken dialogue corpus as a realistic task formulation by extracting biasing list from the ontology of a spoken dialogue system. TCPGen achieved consistent and significant performance improvements in WER and, in particular, rare word error rate (R-WER) across all datasets. MBWE and BLMD each further improved the relative performance improvements of TCPGen for all datasets, and by applying them together, TCPGen achieved around 50% R-WER reduction on the LibriSpeech test-clean and test-other sets using a biasing list of 1000 words.

TCPGen was further examined with universal ASR models trained on hundreds of thousands of hours in conjunction with large language models (LLM) in Chapter 4 by taking Whisper and GPT2 as examples. TCPGen has been configured as a distribution-level adaptation component together with a dedicated training scheme for Whisper that does not require any parameter fine-tuning of the Whisper model. In this way, TCPGen can be treated as a portable and flexible neural component that can be easily applied to any future universal speech model. Experiments on LibriSpeech, SLURP and DSTC data showed consistent performance gains over the original unbiased Whisper model, and the performance improvement is more significant with domain-specific data including SLURP and DSTC.

In Chapter 5, TCPGen was further improved by encoding the prefix tree with GNNs. GNNs exploit the tree structure used for biasing lists in TCPGen and provide the lookahead functionality where the information on the branches rooted from a specific node is also encoded into that node encoding. This enables a better contextualised TCPGen distribution and generation probability to be predicted, which foresees a potential biasing word as early as the first wordpiece of that word.

Three different types of GNN structures were explored in Chapter 5, including the tree recursive neural network (Tree-RNN), the graph convolutional network (GCN) and graphSAGE with a max-pooling aggregator. While Tree-RNN encodes the tree using a single layer of recursive operations, the other two methods encode the tree using multiple layers

of convolution operations. GCN adopts the spectral approach by using the graph Laplacian, whereas GraphSAGE follows the spatial approach by directly exploiting the local graph connections. Furthermore, a combination of the two complementary GNN structures, GCN and GraphSAGE, was also introduced in Chapter 5.

In addition to the aforementioned contextual ASR setups, an innovative audio-visual contextual ASR pipeline was also proposed in Chapter 5 using the AMI meeting corpus. The contextual knowledge in this pipeline is extracted from presentation slides associated with each meeting via optical character recognition (OCR) tools, and then is organised into biasing lists to be integrated using TCPGen. This setup demonstrated the potential benefits of applying contextual biasing, especially TCPGen, in various academic and educational scenarios where papers, posters or presentations are available.

Experiments on the previous contextual ASR tasks, as well as the audio-visual contextual ASR pipeline, show consistent and significant performance boosts using GNN encodings compared to the vanilla TCPGen component. The combination of two GNN encodings achieved the best performance across all evaluated tasks.

In Chapter 6, TCPGen was further introduced into end-to-end SLU systems. It is both natural and necessary to include contextual biasing for the SLU system as it often involves rare but crucial named entities. A structured knowledge base (KB) for each SLU task is usually easily available by gathering all possible entities for each type of slot, and a focused biasing list can be extracted using the structured KB.

Particularly focusing on the slot-filling task, TCPGen was integrated into both the tagging-based SLU system and the sequence generation-based SLU system. For both systems, the slot shortlist (SS) method was proposed which predicts a shortlist of possible slot types given the decoded history tokens using a class LM (CLM), and the shortlist of slot types used to organise the focused biasing list from the KB. For the tagging-based SLU system, the slot probability biasing (SPB) method was also proposed which transforms the TCPGen distribution into a distribution over slot types to be interpolated with the original slot-filling output in the same way as the pointer generator. SS and SPB together improve the SLU system on the SLURP data, especially on rare and unseen entities. SPB also enables the tagging-based system to handle unseen slot types and hence achieves zero-shot learning.

The integration of TCPGen into the sequence generation-based SLU system led to a knowledge-aware audio-grounded (KA2G) generative slot-filling framework, which comprises one slot value generation (SVG) module and two stacked TCPGen modules sharing the same tree encodings on the ASR and SVG sides respectively. The SVG module is inherently able to handle any unseen slot types as the slot is sent to the SVG via natural language queries and the slot value is also generated in natural language. The TCPGen on the ASR side is

used in the same way as the TCPGen in the tagging-based system, while the TCPGen on the SVG side can be regarded as taking sub-trees explored by the beam search procedure during decoding as the most likely knowledge to bias the generation. Experiments performed on the SLURP dataset and an industry data set termed CONCIERGE demonstrate the superiority of KA2G over a strong pipeline system as well as end-to-end generation-based systems without TCPGen. KA2G is particularly beneficial for few-shot learning where only a handful of data samples are provided, which is a desired characteristic for industry case.

7.2 Future Work

One of the biggest challenges that have confronted almost all deep learning research is the emergence and prevalence of large LMs, such as ChatGPT. Despite remaining debatable whether this technology and route of future development will lead to a true form of artificial intelligence, it is clearly a powerful tool that is revolutionising human-computer interaction, and the expected performance for conversational AI systems.

There are two weaknesses of LLMs that are relevant to my future work that have been observed, and where my future work will be focused. One of the drawbacks is the ability to take into consideration the dynamically changing factual knowledge. LLMs tend to generate false facts confidently that will misguide the user, e.g. if you ask it to generate the author list of a paper. Such dynamic knowledge could arguably be injected via “in-context learning", where knowledge, such as the biasing list, is converted into an entire long string used as the context of the user’s query. The context and the query can be concatenated together before sending it to the LLM. Although it mitigates this problem, it is less explicit, less controllable and less reliable compared to methods such as the TCPGen in the slot-value generator for SLU in Section 6.4.

Another challenge for LLMs is their perception across different modalities. LLMs, as is the case for standard LMs, are usually trained on text only and can only deal with text. Recent advancements in GPT4 and Flamingo (Alayrac et al., 2022) have developed a visual LM capability that generates text based on visual input, combining visual and text modalities and yielding state-of-the-art performance. On the other hand, models that combine both visual and audio modalities are still missing, especially with an explicit knowledge integration component.

Based on the aforementioned challenges, there are two directions that I plan to investigate in future, as a continuation of my PhD work:

- Following the concept of the KA2G framework and the application of TCPGen in Whisper, it is worthwhile to investigate contextual biasing using TCPGen for dialogue

response generation with LLMs. As with KA2G, response generation itself is a sequence generation task that is based on domain-specific knowledge. Such knowledge is local to certain applications which makes it challenging for LLMs to accurately respond, hence necessitating dedicated knowledge integration components, such as TCPGen, to be used for LLMs. Similar to the Whisper biasing method, TCPGen can be integrated into open-source LLMs (e.g. LLaMa (Touvron et al., 2023)) as a distribution-level adaptation that can bias the generation procedure following TCPGen in the SVG module of KA2G.

- Another direction to investigate is multi-modal contextual knowledge integration using TCPGen for audio-visual ASR and understanding. This direction will take the audio-visual contextual ASR pipeline as the starting point, and explore various types of visual modality information that are likely to appear in academic and educational application scenarios in addition to the text via OCR only. This includes figures, tables and equations in presentation slides or papers, as well as any general visual objects.

A longer-term direction is to achieve knowledge integration in a prompt-based audio-visual universal LLM. In contrast to ASR, SLU and other specific spoken language technologies, this prompt-based audio-visual LLM formulates all audio-visual tasks into the same prompt-response format. With this formulation, contextual knowledge can be included via TCPGen in the same way as in the KA2G framework.

References

Al-Rfou, R., Choe, D., Constant, N., Guo, M., and Jones, L. (2019). Character-level language modeling with deeper self-attention. In *Proc. Association for the Advancement of Artificial Intelligence (AAAI)*, Honolulu, Hawaii, USA.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., Ring, R., Rutherford, E., Cabi, S., Han, T., Gong, Z., Samangooei, S., Monteiro, M., Menick, J., Borgeaud, S., Brock, A., Nematzadeh, A., Sharifzadeh, S., Binkowski, M., Barreira, R., Vinyals, O., Zisserman, A., and Simonyan, K. (2022). Flamingo: A visual language model for few-shot learning. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, New Orleans, Louisiana, USA.

Alon, U., Pundak, G., and Sainath, T. (2019). Contextual speech recognition with difficult negative training examples. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Brighton, UK.

Arora, S., Dalmia, S., Denisov, P., Chang, X., Ueda, Y., Peng, Y., Zhang, Y., Kumar, S., Ganesan, K., Yan, B., Vu, N. T., Black, A. W., and Watanabe, S. (2022a). ESPnet-SLU: Advancing spoken language understanding through ESPnet. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore.

Arora, S., Dalmia, S., Yan, B., Metze, F., Black, A. W., and Watanabe, S. (2022b). Token-level sequence labeling for spoken language understanding using compositional end-to-end models. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP) findings*.

Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv:1607.06450*.

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Vancouver, British Columbia, Canada.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. In *Proc. The International Conference on Learning Representations (ICLR)*, Banff, Alberta, Canada.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., and Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Shanghai, China.

Bahl, L., Brown, P., Souza, P., and Mercer, R. (1986). Maximum mutual information estimation of hidden Markov parameters for speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Tokyo, Japan.

Bastianelli, E., Vanzo, A., Swietojanski, P., and Rieser, V. (2020). SLURP: A spoken language understanding resource package. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Virtual.

Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. In *Inequalities III: Proceedings of the Third Symposium on Inequalities*, pages 1–8. Academic Press.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*. Vol. 3, No. 6, pp. 1137–1155.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*. Vol. 5, No. 2, pp. 157–166.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*. Vol 13, No. 10, pp. 281–305.

Bishop, C. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bourlard, H. and Morgan, N. (1993). *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*. Vol. 34, No. 4, pp. 18–42.

Brown, P. (1987). *The acoustic modelling problem in automatic speech recognition. PhD Thesis*. Carnegie Mellon University.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Proc. Conference and Workshop on Advances in Neural Information Processing Systems (NeurIPS)*.

Bruguier, A., Le, D., Prabhavalkar, R., Li, D., Liu, Z., Wang, B., Chang, E., Peng, F., Kalinli, O., and Seltzer, M. L. (2022). Neural-FST class language model for end-to-end speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore.

Budzianowski, P. and Vulić, I. (2019). Hello, it's GPT-2 - how can I help you? Towards the use of pretrained language models for task-oriented dialogue systems. In *Proc. NGT*, Hong Kong, China.

Cai, H., Zheng, V. W., and Chang, K. C.-C. (2017). A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 30, No. 9, pp. 1616–1637.

Campagna, G., Foryciarz, A., Moradshahi, M., and Lam, M. (2020). Zero-shot transfer learning with synthesized data for multi-domain dialogue state tracking. In *Proc. The Association for Computational Linguistics (ACL)*, Virtual.

Casanueva, I., Vulić, I., Spithourakis, G., and Budzianowski, P. (2022). NLU++: A multi-label, slot-rich, generalisable dataset for natural language understanding in task-oriented dialogue. In *Proc. North American Chapter of the Association for Computational Linguistics (NAACL) findings*, Seattle, Washington, USA.

Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Shanghai, China.

Chen, L., Lv, B., Wang, C., Zhu, S., Tan, B., and Yu, K. (2020a). Schema-guided multi-domain dialogue state tracking with graph attention neural networks. In *Proc. Association for the Advancement of Artificial Intelligence (AAAI)*, New York, USA.

Chen, M., Wei, Z., Huang, Z., Ding, B., and Li, Y. (2020b). Simple and deep graph convolutional networks. In *Proc. The International Conference on Machine Learning (ICML)*, Vienna, Austria.

Chen, Q., Zhuo, Z., and Wang, W. (2019a). BERT for joint intent classification and slot filling. *arXiv:1902.10909*.

Chen, S., Wang, C., Chen, Z., Wu, Y., Liu, S., Chen, Z., Li, J., Kanda, N., Yoshioka, T., Xiao, X., Wu, J., Zhou, L., Ren, S., Qian, Y., Qian, Y., Zeng, M., Yu, X., and Wei, F. (2022). Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*. Vol. 16, No. 6, pp. 1505–1518.

Chen, X. (2017). Scalable recurrent neural network language models for speech recognition. *PhD thesis*.

Chen, X., Liu, X., Qian, Y., Gales, M. J. F., and Woodland, P. C. (2016). CUED-RNNLM: an open-source toolkit for efficient training and evaluation of recurrent neural network language models. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Shanghai, China.

Chen, X., Qiu, X., Zhu, C., Wu, S., and Huang, X. (2015a). Sentence modeling with gated recursive neural network. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Lisbon, Portugal.

Chen, X., Tan, T., Liu, X., Lanchantin, P., Wan, M., Gales, M. J. F., and Woodland, P. C. (2015b). Recurrent neural network language model adaptation for multi-genre broadcast speech recognition. In *Proc. Interspeech*, Dresden, Germany.

Chen, X., Wang, Y., Liu, X., Gales, M. J. F., and Woodland, P. C. (2014). Efficient GPU-based training of recurrent neural network language models using spliced sentence bunch. In *Proc. Interspeech*, Singapore.

Chen, Z., Jain, M., Wang, Y., Seltzer, M. L., and Fuegen, C. (2019b). End-to-end contextual speech recognition using class language models and a token passing decoder. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Brighton, UK.

Chen, Z., Jain, M., Wang, Y., Seltzer, M. L., and Fuegen, C. (2019c). Joint grapheme and phoneme embeddings for contextual end-to-end asr. In *Proc. Interspeech*, Graz, Austria.

Chiu, C.-C. and Raffel, C. (2017). Monotonic chunkwise attention. In *Proc. The International Conference on Learning Representations (ICLR)*, Toulon, France.

Chorowski, J. and Jaitly, N. (2016). Towards better decoding and language model integration in sequence to sequence models. In *Proc. Interspeech*, San Francisco, California, USA.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Montreal, Canada.

Cui, P., Wang, X., Pei, J., and Zhu, W. (2017). A survey on network embedding. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 31, No. 5, pp. 833–852.

Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *Proc. The International Conference on Machine Learning (ICML)*, Sydney, Australia.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. Vol. 28, No. 4, pp. 357–366.

Desot, T., Portet, F., and Vacher, M. (2022). End-to-end spoken language understanding: Performance analyses of a voice command task in a low resource setting. *Computer Speech & Language*. Vol. 75, No. C, pp. 101369.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. The Association for Computational Linguistics (ACL)*, Florence, Italy.

Du, X., He, L., Li, Q., Yu, D., Pasupat, P., and Zhang, Y. (2021). QA-driven zero-shot slot filling with weak supervision pretraining. In *Proc. The Association for Computational Linguistics (ACL)*, Bangkok, Thailand.

Frasconi, P., Gori, M., and Sperduti, A. (1998). A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*. Vol. 9, No. 5, pp. 768–786.

Fuisz, G., Vulić, I., Gibbons, S., Casanueva, I., and Budzianowski, P. (2022). Improved and efficient conversational slot labeling through question answering. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Abu Dhabi, UAE.

Gage, P. (1994). A new algorithm for data compression. *C Users Journal*. Vol. 12, No. 2, pp. 23–28.

Ganesan, K., Bamdev, P., B, J., Venugopal, A., and Tushar, A. (2021). N-best ASR transformer: Enhancing SLU performance using multiple ASR hypotheses. In *Proc. ICNLP*, Beijing, China.

Garg, A., Gupta, A., Gowda, D., Singh, S., and Kim, C. (2020). Hierarchical multi-stage word-to-grapheme named entity corrector for automatic speech recognition. In *Proc. Interspeech*, pages 1793–1797, Shanghai, China.

Gasteiger, J., Bojchevski, A., and Günnemann, S. (2019). Predict then propagate: Graph neural networks meet personalized pagerank. In *Proc. The International Conference on Learning Representations (ICLR)*, Edinburgh, UK.

Ghahremani, P., BabaAli, B., Povey, D., Riedhammer, K., Trmal, J., and Khudanpur, S. (2014). A pitch extraction algorithm tuned for automatic speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Florence, Italy.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Graves, A. (2011). Practical variational inference for neural networks. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Granada, Spain.

Graves, A. (2012). Sequence transduction with recurrent neural networks. In *Proc. The International Conference on Machine Learning (ICML) Representation Learning Workshop*, Edinburgh, Scotland.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv:1308.0850*.

Graves, A., Fernández, S., Gomez, F. J., and Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proc. The International Conference on Machine Learning (ICML)*, Pittsburgh, Pennsylvania, USA.

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., and Pang, R. (2020). Conformer: Convolution-augmented transformer for speech recognition. In *Proc. Interspeech*, Shanghai, China.

Gulcehre, C., Firat, O., Xu, K., Cho, K., Barrault, L., Lin, H.-C., Bougares, F., Schwenk, H., and Bengio, Y. (2015). On using monolingual corpora in neural machine translation. *arXiv:1503.03535*.

Guo, J., Tiwari, G., Droppo, J., Segbroeck, M. V., Huang, C.-W., Stolcke, A., and Maas, R. (2020). Efficient minimum word error rate training of RNN-Transducer for end-to-end speech recognition. In *Proc. Interspeech*, Shanghai, China.

Gutmann, M. and Hyvärinen, A. (2010). Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proc. International Conference on Artificial Intelligence and Statistics (AISTATS)*.

Haghani, P., Narayanan, A., Bacchiani, M., Chuang, G., Gaur, N., Moreno, P., Prabhavalkar, R., Qu, Z., and Waters, A. (2018). From audio to semantics: Approaches to end-to-end spoken language understanding. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Athens, Greece.

Hall, K. B., Cho, E., Allauzen, C., Beaufays, F., Coccaro, N., Nakajima, K., Riley, M., Roark, B., Rybach, D., and Zhang, L. (2015). Composition-based on-the-fly rescoring for salient n-gram biasing. In *Proc. Interspeech*, Dresden, Germany.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Long Beach, California, USA.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017b). Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*. Vol. 40, No. 3, pp. 52–74.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. In *Proc. IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, Boston, Massachusetts, USA.

He, Y., Sainath, T., Prabhavalkar, R., McGraw, I., Alvarez, R., Zhao, D., Rybach, D., Kannan, A., Wu, Y., Pang, R., Liang, Q., Bhatia, D., Shangguan, Y., Li, B., Pundak, G., Sim, K., Bagby, T., Chang, S.-y., Rao, K., and Gruenstein, A. (2019). Streaming end-to-end speech recognition for mobile devices. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Brighton, UK.

Heck, M., van Niekerk, C., Lubis, N., Geishauser, C., Lin, H.-C., Moresi, M., and Gasic, M. (2020). TripPy: A triple copy strategy for value independent neural dialog state tracking. In *Proc. Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, Virtual.

Henderson, M. and Vulić, I. (2021). ConVEx: Data-efficient and few-shot slot labeling. In *Proc. North American Chapter of the Association for Computational Linguistics (NAACL)*, Seattle, Washington, USA.

Hermansky, H. (1990). Perceptual linear predictive (PLP) analysis of speech. *Journal of the Acoustical Society of America*. Vol. 87, No. 4, pp. 1738–1752.

Hinton, G., Vinyals, O., and Dean, J. (2014). Distilling the knowledge in a neural network. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS) Deep Learning workshop*, Montreal, Canada.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*. Vol. 9, No. 8, pp.1735–1780.

Hori, T., Watanabe, S., Zhang, Y., and Chan, W. (2017). Advances in joint CTC-attention based end-to-end speech recognition with a deep CNN encoder and RNNLM. In *Proc. Interspeech*, Stockholm, Sweden.

Hosseini-Asl, E., McCann, B., Wu, C.-S., Yavuz, S., and Socher, R. (2020). A simple language model for task-oriented dialogue. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Virtual.

Hou, Y., Che, W., Lai, Y., Zhou, Z., Liu, Y., Liu, H., and Liu, T. (2020). Few-shot slot tagging with collapsed dependency transfer and label-enhanced task-adaptive projection network. In *Proc. The Association for Computational Linguistics (ACL)*, Seattle, Washington, USA.

Hsu, W.-N., Bolte, B., Tsai, Y.-H., Lakhotia, K., Salakhutdinov, R., and Mohamed, A. (2021). HuBERT: Self-supervised speech representation learning by masked prediction of hidden units. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Vol. 29, pp. 3451–3460.

Huang, R., Abdel-Hamid, O., Li, X., and Evermann, G. (2020). Class LM and word mapping for contextual biasing in end-to-end ASR. In *Proc. Interspeech*, Shanghai, China.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*. Vol. 18, No. 1, pp.6869–6898.

Huber, C., Hussain, J., Stüker, S., and Waibel, A. H. (2021). Instant one-shot word-learning for context-specific neural sequence-to-sequence speech recognition. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Cartagena, Colombia.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. The International Conference on Machine Learning (ICML)*, Lille, France.

Irie, K., Zeyer, A., Schlüter, R., and Ney, H. (2019). Language modeling with deep transformers. In *Proc. Interspeech*, Graz, Austria.

Jaech, A. and Ostendorf, M. (2018). Low-rank RNN adaptation for context-aware language modeling. In *Proc. The Association for Computational Linguistics (ACL)*, Melbourne, Australia.

Jaitly, N. and Hinton, E. (2013). Vocal tract length perturbation (VTLP) improves speech recognition. In *Proc. The International Conference on Machine Learning (ICML)*, Atlanta, USA.

Jelinek, F. (1998). *Statistical Methods for Speech Recognition*. MIT Press, Cambridge, MA, USA.

Jim, K., Giles, C., and Horne, B. (1996). An analysis of noise in recurrent neural networks: Convergence and generalization. *IEEE Transactions on Neural Networks*. Vol. 7, No. 6, pp. 1424–1438.

Juang, B., Hou, W., and Lee, C. (1997). Minimum classification error rate methods for speech recognition. *IEEE Transactions on Speech and Audio Processing*. Vol. 5, No. 3, pp. 257–265.

Jurafsky, D. and Martin, J. H. (2009). *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall.

Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 33, No. 1, pp. 117–128.

Kang, Y. and Zhou, Y. (2020). Fast and robust unsupervised contextual biasing for speech recognition. In *U.S. Patent Application No. 16/993,797*.

Kanthak, S. and Ney, H. (2002). Context-dependent acoustic modelling using graphemes for large vocabulary speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Orlando, Florida, USA.

Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*. Vol. 35, No. 3, pp. 400–401.

Kawara, Y., Chu, C., and Arase, Y. (2018). Recursive neural network based preordering for English-to-Japanese machine translation. In *Proc. The Association for Computational Linguistics (ACL) workshop*, Melbourne, Australia.

Kim, J., Lee, Y., and Kim, E. (2020). Accelerating RNN-Transducer inference via adaptive expansion search. *IEEE Signal Processing Letters*. Vol. 27, pp. 2019–2023.

Kim, S., Hori, T., and Watanabe, S. (2016). Joint CTC-attention based end-to-end speech recognition using multi-task learning. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Shanghai, China.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proc. The International Conference on Learning Representations (ICLR)*, San Diego, California, USA.

Kipf, T. N. and Welling, M. (2017). Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. The International Conference on Learning Representations (ICLR)*, Toulon, France.

Kneser, R. and Ney, H. (1995). Improved backing-off for n-gram language modelling. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Detroit, Michigan, USA.

Ko, T., Peddinti, V., Povey, D., and Khudanpur, S. (2015). Audio augmentation for speech recognition. In *Proc. Interspeech*, Dresden, Germany.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Lake Tahoe, Nevada, USA.

Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proc. The Association for Computational Linguistics (ACL)*, Melbourne, Australia.

Lauscher, A., Ravishankar, V., Vulić, I., and Glavaš, G. (2020). From zero to hero: On the limitations of zero-shot language transfer with multilingual Transformers. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Virtual.

Le, D., Jain, M., Keren, G., Kim, S., Shi, Y., Mahadeokar, J., Chan, J., Shangguan, Y., Fuegen, C., Kalinli, O., Saraf, Y., and Seltzer, M. (2021a). Contextualized streaming end-to-end speech recognition with trie-based deep biasing and shallow fusion. In *Proc. Interspeech*, Brno, Czech Republic.

Le, D., Keren, G., Chan, J., Mahadeokar, J., Fuegen, C., and Seltzer, M. (2021b). Deep shallow fusion for RNN-T personalization. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Shenzhen, China.

Le, D., Shrivastava, A., Tomasello, P., Kim, S., Livshits, A., Kalinli, O., and Seltzer, M. (2022). Deliberation model for on-device spoken language understanding. In *Proc. Interspeech*, Incheon, Korea.

Lecun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*. Vol. 1, No. 4, pp. 541–551.

LeCun, Y., Bottou, L., Orr, G., and Müller, K. (2012). *Efficient backprop*, pages 9–48. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Springer Verlag.

Lee, K.-F. (1988). On large-vocabulary speaker-independent continuous speech recognition. *Speech Communication*. Vol. 7, No. 4, pp. 375–379.

Li, J., Zhao, R., Meng, Z., Liu, Y., Wei, W., Parthasarathy, S., Mazalov, V., Wang, Z., He, L., Zhao, S., and Gong, Y. (2020a). Developing RNN-T models surpassing high-performance hybrid models with customization capability. In *Proc. Interspeech*, Shanghai, China.

Li, K., Liu, Z., He, T., Huang, H., Peng, F., Povey, D., and Khudanpur, S. (2020b). An empirical study of transformer-based neural language model adaptation. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Barcelona, Spain.

Li, P.-H., Dong, R.-P., Wang, Y.-S., Chou, J.-C., and Ma, W.-Y. (2017). Leveraging linguistic structures for named entity recognition with bidirectional recursive neural networks. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Li, Q. (2022). Attention-based encoder-decoder models for speech processing. *PhD Thesis*.

Li, Q., Zhang, C., and Woodland, P. C. (2023). Combining hybrid DNN-HMM ASR systems with attention-based models using lattice rescoring. *Speech Communication*. Vol. 147, No. C, pp. 12–21.

Li, W., Peng, R., Wang, Y., and Yan, Z. (2020c). Knowledge graph based natural language generation with adapted pointer-generator networks. *Neurocomputing*. Vol. 382, No. C, pp. 174–187.

Lichouri, M., Lounnas, K., Djeradi, R., and Djeradi, A. (2022). Performance of end-to-end vs pipeline spoken language understanding models on multilingual synthetic voice. In *Proc. ICAASE*, Constantine, Algeria.

Lin, W., Tseng, B.-H., and Byrne, B. (2021a). Knowledge-aware graph-enhanced GPT-2 for dialogue state tracking. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Mexico City, Mexico.

Lin, Z., Liu, B., Moon, S., Crook, P., Zhou, Z., Wang, Z., Yu, Z., Madotto, A., Cho, E., and Subba, R. (2021b). Leveraging slot descriptions for zero-shot cross-domain dialogue state tracking. In *Proc. North American Chapter of the Association for Computational Linguistics (NAACL)*, Mexico City, Mexico.

Liu, D.-R., Liu, C., Zhang, F., Synnaeve, G., Saraf, Y., and Zweig, G. (2020a). Contextualizing ASR lattice rescoring with hybrid pointer network language model. In *Proc. Interspeech*, Shanghai, China.

Liu, J., Yu, M., Chen, Y., and Xu, J. (2022). Cross-domain slot filling as machine reading comprehension: A new perspective. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Vol. 30, pp. 673–685.

Liu, S., Yang, N., Li, M., and Zhou, M. (2014). A recursive recurrent neural network for statistical machine translation. In *Proc. The Association for Computational Linguistics (ACL)*, Baltimore, Maryland, USA.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019a). RoBERTa: A robustly optimized bert pretraining approach. *arxiv:1907.11692*.

Liu, Z., Ng, A., Guang, S. L. S., Aw, A., and Chen, N. F. (2019b). Topic-aware pointer-generator networks for summarizing spoken conversations. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Singapore.

Liu, Z., Winata, G. I., Xu, P., and Fung, P. (2020b). Coach: A coarse-to-fine approach for cross-domain slot filling. In *Proc. The Association for Computational Linguistics (ACL)*, Seattle, Washington, USA.

Luong, T., Socher, R., and Manning, C. (2013). Better word representations with recursive neural networks for morphology. In *Proc. CoNLL*, Sofia, Bulgaria.

MacKay, D. J. C. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Madotto, A., Liu, Z., Lin, Z., and Fung, P. (2020). Language models as few-shot learner for task-oriented dialogue systems. *arXiv:2008.06239*.

McDermott, E., Sak, H., and Variani, E. (2019). A density ratio approach to language model fusion in end-to-end automatic speech recognition. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Singapore.

Mehri, S. and Eskenazi, M. (2021). GenSF: Simultaneous adaptation of generative pre-trained models and slot filling. In *Proc. Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, Singapore.

Meng, Z., Kanda, N., Gaur, Y., Parthasarathy, S., Sun, E., Lu, L., Chen, X., Li, J., and Gong, Y. (2021a). Internal language model training for domain-adaptive end-to-end speech recognition. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Toronto, Canada.

Meng, Z., Parthasarathy, S., Sun, E., Gaur, Y., Kanda, N., Lu, L., Chen, X., Zhao, R., Li, J., and Gong, Y. (2021b). Internal language model estimation for domain-adaptive end-to-end speech recognition. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Shenzhen, China.

Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*. Vol. 20, No. 3, pp. 498–511.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *Proc. The International Conference on Learning Representations (ICLR) Workshop*, Scottsdale, Arizona, USA.

Mikolov, T., Karafiat, M., Burget, L., Cernocky, J. H., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proc. Interspeech*, Makuhari, Japan.

Mikolov, T. and Zweig, G. (2012). Context dependent recurrent neural network language model. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Miami, Florida, USA.

Mohamed, A., Hinton, G., and Penn, G. (2012). Understanding how deep belief networks perform acoustic modelling. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Kyoto, Japan.

Namazifar, M., Papangelis, A., Tur, G., and Hakkani-Tur, D. (2020). Language model is all you need: Natural language understanding as question answering. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Barcelona, Spain.

Nguyen, M.-T., Ngo, G. H., and Chen, N. F. (2019). Hierarchical character embeddings: Learning phonological and semantic representations in languages of logographic origin using recursive neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Vol. 28, pp. 461–473.

Oparin, I., Sundermeyer, M., Ney, H., and Gauvain, J. (2012). Performance analysis of neural networks in combination with n-gram language models. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Kyoto, Japan.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L. E., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., and Lowe, R. J. (2022). Training language models to follow instructions with human feedback. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, New Orleans, Louisiana, USA.

Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., and Le, Q. V. (2019). Specaugment: A simple data augmentation method for automatic speech recognition. In *Proc. Interspeech*, Graz, Austria.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch:

An imperative style, high-performance deep learning library. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada.

Peng, Y., Arora, S., Higuchi, Y., Ueda, Y., Kumar, S., Ganesan, K., Dalmia, S., Chang, X., and Watanabe, S. (2022). A study on the integration of pre-trained SSL, ASR, LM and SLU models for spoken language understanding. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Doha, Qatar.

Perozzi, B., Al-Rfou, R., and Skiena, S. (2014). Deepwalk: Online learning of social representations. In *Proc. Special Interest Group on Knowledge Discovery in Data (SIGKDD)*, New York, USA.

Povey, D., Peddinti, V., Galvez, D., Ghahrmani, P., Manohar, V., Na, X., Wang, Y., and Khudanpur, S. (2016). Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *Proc. Interspeech*, San Francisco, California, USA.

Povey, D. and Woodland, P. (2002). Minimum phone error and I-smoothing for improved discriminative training. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Orlando, Florida, USA. IEEE.

Prabhavalkar, R., Rao, K., Sainath, T. N., Li, B., Johnson, L., and Jaitly, N. (2017). A comparison of sequence-to-sequence models for speech recognition. In *Proc. Interspeech*, Stockholm, Sweden.

Prabhavalkar, R., Sainath, T. N., Wu, Y., Nguyen, P., Chen, Z., Chiu, C.-C., and Kannan, A. (2018). Minimum word error rate training for attention-based sequence-to-sequence models. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Calgary, Alberta, Canada.

Pundak, G., Sainath, T. N., Prabhavalkar, R., Kannan, A., and Zhao, D. (2018). Deep context: End-to-end contextual speech recognition. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Athens, Greece.

Qian, Y. and Woodland, P. C. (2016). Very deep convolutional neural networks for robust speech recognition. In *Proc. IEEE Spoken Language Technology Workshop (SLT)*, San Diago, California, USA.

Radfar, M. H., Mouchtaris, A., and Kunzmann, S. (2020). End-to-end neural transformer based spoken language understanding. In *Proc. Interspeech*, Shanghai, China.

Radford, A., Kim, J., Xu, T., Brockman, G., McLeavey, C., and Sutskever, I. (2023). Robust speech recognition via large-scale weak supervision. In *Proc. The International Conference on Machine Learning (ICML)*, Honolulu, Hawai'i, USA.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. In OpenAI Blog. https://openai.com/research/language-unsupervised.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. In OpenAI Blog. https://openai.com/research/gpt-2-1-5b-release.

Raffel, C., Luong, M.-T., Liu, P. J., Weiss, R. J., and Eck, D. (2017). Online and linear-time attention by enforcing monotonic alignments. In *Proc. The International Conference on Machine Learning (ICML)*, Sydney, Australia.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*. Vol. 21, No. 140, pp. 1–67.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Austin, Texas, USA.

Raju, A., Rao, M., Tiwari, G., Dheram, P., Anderson, B., Zhang, Z., Lee, C., Bui, B., and Rastrow, A. (2022). On joint training with interfaces for spoken language understanding. In *Proc. Interspeech*, Incheon, Korea.

Ramachandran, P., Zoph, B., and Le, Q. V. (2018). Searching for activation functions. In *Proc. The International Conference on Learning Representations (ICLR)*, Vancouver, BC, Canada.

Rao, M., Dheram, P., Tiwari, G., Raju, A., Droppo, J., Rastrow, A., and Stolcke, A. (2021). Do as i mean, not as i say: Sequence loss training for spoken language understanding. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Toronto, Canada.

Rosenblatt, F. (1962). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*. Vol 323, No. 6088, pp. 533–536.

Sadr, H., Pedram, M. M., and Teshnehlab, M. (2019). A robust sentiment analysis method based on sequential combination of convolutional and recursive neural networks. *Neural Processing Letters*. Vol. 50, No. 3, pp. 2745–2761.

Sainath, T., He, Y., Li, B., Narayanan, A., Pang, R., Bruguier, A., Chang, S.-y., Li, W., Alvarez, R., Chen, Z., Chiu, C.-C., Garcia, D., Gruenstein, A., Hu, K., Jin, M., Kannan, A., Liang, Q., McGraw, I., Peyser, C., and Zhao, D. (2020). A streaming on-device end-to-end model surpassing server-side conventional model quality and latency. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Barcelona, Spain.

Sainath, T., Mohamed, A., Kingsbury, B., and Ramabhadran, B. (2013). Deep convolutional neural networks for LVCSR. In *Proc IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Vancouver, Canada.

Sainath, T., Prabhavalkar, R., Kumar, S., Lee, S., Kannan, A., Rybach, D., Schogol, V., Nguyen, P., Li, B., Wu, Y., Chen, Z., and Chiu, C. (2018). No need for a lexicon? In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Calgary, Alberta, Canada.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*. Vol. 20, No. 1, pp.61–80.

See, A., Liu, P. J., and Manning, C. D. (2017). Get to the point: Summarization with pointer-generator networks. In *Proc. The Association for Computational Linguistics (ACL)*, Vancouver, Canada.

Seo, S., Kwak, D., and Lee, B. (2022). Integration of pre-trained networks with continuous token interface for end-to-end spoken language understanding. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore.

Serdyuk, D., Wang, Y., Fuegen, C., Kumar, A., Liu, B., and Bengio, Y. (2018). Towards end-to-end spoken language understanding. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Calgary, Alberta, Canada.

Shah, D., Gupta, R., Fayazi, A., and Hakkani-Tur, D. (2019). Robust zero-shot cross-domain slot filling with example values. In *Proc. The Association for Computational Linguistics (ACL)*, Florence, Italy.

Shan, C., Weng, C., Wang, G., Su, D., Luo, M., Yu, D., and Xie, L. (2019). Component fusion: Learning replaceable language model component for end-to-end speech recognition system. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Brighton, UK.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*. Vol. 27, No. 3, pp.379–423.

Shen, D., Wang, G., Wang, W., Min, M. R., Su, Q., Zhang, Y., Li, C., Henao, R., and Carin, L. (2018). Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms. In *Proc. The Association for Computational Linguistics (ACL)*, Melbourne, Australia.

Sietsma, J. and Dow, R. J. (1991). Creating artificial neural networks that generalize. *Neural Networks*. Vol. 4, No. 1, pp.67–79.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *Proc. The International Conference on Learning Representations (ICLR)*, San Diego, California, USA.

Soltau, H., Liao, H., and Sak, H. (2017). Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition. In *Proc. Interspeech*, Stockholm, Sweden.

Sperduti, A. and Starita, A. (1997). Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*. Vol. 8, No. 3, pp.714–735.

Sriram, A., Jun, H., Satheesh, S., and Coates, A. (2017). Cold fusion: Training seq2seq models together with language models. In *Proc. Interspeech*, Stockholm, Sweden.

Sun, A., Wang, J., Cheng, N., Peng, H., Zeng, Z., and Xiao, J. (2020a). GraphTTS: Graph-to-sequence modelling in neural text-to-speech. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Barcelona, Spain.

Sun, G., Zhang, C., Vulić, I., Budzianowski, P., and Woodland, P. C. (2023a). Knowledge-aware audio-grounded generative slot filling with limited annotated data. In *Preparation*.

Sun, G., Zhang, C., and Woodland, P. C. (2020b). Cross-utterance language models with acoustic error sampling. *arXiv:2009.01008*.

Sun, G., Zhang, C., and Woodland, P. C. (2021a). Combination of deep speaker embeddings for diarisation. *Special issue on Advances in Deep Learning Based Speech Processing, Neural Networks*. Vol. 141, No. 1, pp. 372–384.

Sun, G., Zhang, C., and Woodland, P. C. (2021b). Transformer language models with LSTM-based cross-utterance information representation. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Toronto, Canada.

Sun, G., Zhang, C., and Woodland, P. C. (2021c). Tree-constrained pointer generator for end-to-end contextual speech recognition. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Cartagena, Colombia.

Sun, G., Zhang, C., and Woodland, P. C. (2022a). Minimising biasing word errors for contextual asr with the tree-constrained pointer generator. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*. Vol. 31, pp. 345–354.

Sun, G., Zhang, C., and Woodland, P. C. (2022b). Tree-constrained pointer generator with graph neural network encodings for contextual speech recognition. In *Proc. Interspeech*, Incheon, Korea.

Sun, G., Zhang, C., and Woodland, P. C. (2023b). End-to-end spoken language understanding with tree-constrained pointer generator. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

Sundermeyer, M., Schluter, R., and Ney, H. (2012). LSTM neural networks for language modelling. In *Proc. Interspeech*, Portland, Oregon, USA.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Montreal, Canada.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proc. IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, Vegas, Nevada, USA.

Tjandra, A., Sakti, S., and Nakamura, S. (2017). Local monotonic attention mechanism for end-to-end speech and language processing. In *Proc. International Conference on Natural Language Processing (ICNLP)*, Taipei, Taiwan.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). LLaMA: Open and efficient foundation language models. *arXiv:2302.13971*.

Tur, G. and Mori, R. D. (2011). *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*. Wiley.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv:1807.03748*.

Variani, E., Rybach, D., Allauzen, C., and Riley, M. (2020). Hybrid autoregressive Transducer (HAT). In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Barcelona, Spain.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Long Beach, California, USA.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *Journal of the Acoustical Society of America*. Vol. 37, No. 3, pp. 328–339.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP) Workshop*, Brussels, Belgium.

Wang, C., Li, M., and Smola, A. J. (2019). Language models with transformers. *arXiv:1904.09408*.

Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., and Tang, X. (2017). Residual attention network for image classification. In *Proc. IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, Honolulu, Hawaii, USA.

Wang, P., Su, Y., Zhou, X., Ye, X., Wei, L., Liu, M., You, Y., and Jiang:, F. (2022). Speech2Slot: A limited generation framework with boundary detection for slot filling from speech. In *Proc. Interspeech*, Incheon, Korea.

Wang, Y., Boumadane, A., and Heba, A. (2021). A fine-tuned wav2vec2.0/HuBERT benchmark for speech emotion recognition, speaker verification and spoken language understanding. *arXiv:2111.02735*.

wen Yang, S., Chi, P.-H., Chuang, Y.-S., Lai, C.-I. J., Lakhotia, K., Lin, Y. Y., Liu, A. T., Shi, J., Chang, X., Lin, G.-T., Huang, T.-H., Tseng, W.-C., tik Lee, K., Liu, D.-R., Huang, Z., Dong, S., Li, S.-W., Watanabe, S., Mohamed, A., and yi Lee, H. (2021). SUPERB: Speech Processing Universal PERformance Benchmark. In *Proc. Interspeech*, Brno, Czech Republic.

Weng, C., Cui, J., Wang, G., Wang, J., Yu, C., Su, D., and Yu, D. (2018). Improving attention based sequence-to-sequence models for end-to-end English conversational speech recognition. In *Proc. Interspeech*, Hyderabad, India.

Weng, C., Yu, C., Cui, J., and Zhang, C. (2020). Minimum Bayes risk training of RNN-Transducer for end-to-end speech recognition. In *Proc. Interspeech*, Shanghai, China.

Whittaker, E. and Woodland, P. C. (2000). Particle-based language modelling. In *Proc. Interspeech*, Beijing, China.

Williams, I., Kannan, A., Aleksic, P., Rybach, D., and Sainath, T. (2018). Contextual speech recognition in end-to-end neural network systems using beam search. In *Proc. Interspeech*, Hyderabad, India.

Woodland, P. and Povey, D. (2002). Large scale discriminative training of hidden Markov models for speech recognition. *Computer Speech & Language*. Vol. 16, No. 1, pp. 25–47.

Woodland, P. C. (1989). Weight limiting, weight quantisation and generalisation in multi-layer perceptrons. In *Proc. IEEE International Conference on Artificial Neural Networks (ICANN)*, London, UK.

Wynands, N.-P., Michel, W., Rosendahl, J., Schlüter, R., and Ney, H. (2022). Efficient sequence training of attention models using approximative recombination. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore.

Yang, X., Li, Q., and Woodland, P. C. (2022). Knowledge distillation for neural Transducers from large self-supervised pre-trained models. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2019). XLNet: Generalized autoregressive pretraining for language understanding. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada.

Zeiler, M. D. (2012). ADADELTA: An adaptive learning rate method. *arXiv:1212.5701*.

Zhang, C., Li, B., Lu, Z., Sainath, T., and Chang, S.-Y. (2022). Improving the fusion of acoustic and text representations in RNN-T. In *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Singapore.

Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. (2019). Self-attention generative adversarial networks. In *Proc. The International Conference on Machine Learning (ICML)*, Long Beach, California, USA.

Zhang, T., Huang, M., and Zhao, L. (2018). Learning structured representation for text classification via reinforcement learning. In *Proc. Association for the Advancement of Artificial Intelligence (AAAI)*, New Orleans, Louisiana, USA.

Zhang, X., Zhao, J., and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Proc. Conference and Workshop on Neural Information Processing Systems (NeurIPS)*, Cambridge, MA, USA. MIT Press.

Zhang, Y., Han, W., Qin, J., Wang, Y., Bapna, A., Chen, Z., Chen, N., Li, B., Axelrod, V., Wang, G., Meng, Z., Hu, K., Rosenberg, A., Prabhavalkar, R., Park, D. S., Haghani, P., Riesa, J., Perng, G., Soltau, H., Strohman, T., Ramabhadran, B., Sainath, T., Moreno, P., Chiu, C.-C., Schalkwyk, J., Beaufays, F., and Wu, Y. (2023). Google USM: Scaling automatic speech recognition beyond 100 languages. *arXiv:2303.01037*.

Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Laurent, C., Bengio, Y., and Courville, A. C. (2016). Towards end-to-end speech recognition with deep convolutional neural networks. In *Proc. Interspeech*, San Francisco, California, USA.

Zhao, D., Sainath, T. N., Rybach, D., Rondon, P., Bhatia, D., Li, B., and Pang, R. (2019). Shallow-fusion end-to-end contextual biasing. In *Proc. Interspeech*, Graz, Austria.

Zheng, X., Zhang, C., and Woodland, P. C. (2021). Adapting GPT, GPT-2 and BERT language models for speech recognition. In *Proc. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Cartagena, Colombia.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*. Vol. 1, pp. 57–81.