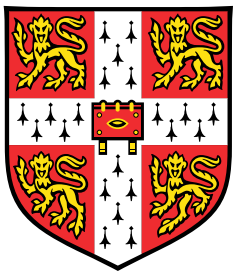
Improving Cascaded Systems in Spoken Language Processing



Yiting Lu

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Murray Edwards College September 2022

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Yiting Lu
September 2022

Abstract

Improving Cascaded Systems in Spoken Language Processing
Yiting Lu

Spoken language processing encompasses a broad range of speech production and perception tasks. One of the central challenges in building spoken language systems is the lack of end-to-end training corpora. For example in spoken language translation, there is little annotated data that directly transcribes speech into a foreign language. Therefore, in spoken language processing, a cascaded structure is widely adopted. This breaks down the complex task into simpler modules, so that individual modules can be trained with sufficient amount of data from the associated domains. However, this simplified cascaded structure suffers from several issues. The upstream and downstream modules are usually connected via an intermediate variable, which does not always encapsulate all the information needed for the downstream processing. For example, speech transcriptions cannot convey prosodic information, and any downstream tasks operating on transcripts will have no access to speech prosodies. The cascaded structure also forces early decisions to be made at the upstream modules, and early stage errors would potentially propagate through and degrade the downstream modules. Furthermore, individual modules in the cascaded system are often trained in their corresponding domains, which can be different from the target domain of the spoken language task. The mismatched training and evaluation domains would cause performance degradation at the inference stage. The focus of this thesis is therefore to investigate multimodular integration approaches addressing the issues facing the simple cascaded structure, and to improve spoken language processing tasks under limited end-to-end data.

The contributions of this thesis are three-fold. The first contribution is to describe the general concept of multimodular combination. The scoring criteria are modified to enable assessment of individual modules and complete systems, and approaches are explored to improve the vanilla cascaded structure. Three categories of spoken language systems are considered with an increasing level of module integration: cascaded, integrated and end-to-end systems. Cascaded systems train individual modules in their corresponding domains, and do not require any end-to-end corpora. Integrated systems propagate richer

information across modular connections, and require a small amount of end-to-end data to adapt to the target domain. End-to-end systems drop the notion of modules and require large amount of end-to-end data to reach convergence. More tightly integrated systems generally require larger amount of end-to-end training data. With the trade-off between modelling power and data efficiency, different approaches are discussed aiming to strike a balance between the two. The second contribution of this thesis is to propose a general framework of reranking for multimodular systems, addressing both the error propagation and information loss issues. Rerankers are commonly used for single module sequence generation tasks, such as speech recognition and machine translation. In this work, rerankers are applied to multimodular systems, where they directly access the hypothesis space of the intermediate variables at the modular connection. Taking into account multiple hypotheses of the modular connection leads to a richer information flow across modules, and consequently helps reduce error propagation. The third contribution of this thesis is to propose the embedding passing approach. The idea is to extract continuous feature representations of the upstream context, and use them as the modular connection. The embedding connection allows richer information propagation as well as gradient backpropagation across modules, thus enabling joint optimisation of the multimodular system.

Among the wide range of possible spoken language tasks, this thesis considers three example tasks with an increasing level of complexity: spoken disfluency detection (SDD), spoken language translation (SLT) and spoken grammatical error correction (SGEC). Spontaneous speech often comes with disfluencies, such as filled pauses, repetitions and false starts. As an important pre-processing step for many spoken language systems, SDD removes speech disfluencies and recovers a fluent transcription flow for downstream text processing tasks. SLT converts speech inputs into foreign text outputs, which is commonly adopted for automatic video subtitling as well as simultaneous interpreting. It is a challenging application that brings together automatic speech recognition (ASR) and neural machine translation (NMT), both of which are complex sequence-to-sequence tasks. With growing global demand for learning a second language, SGEC has become increasingly important to give feedback on the grammatical structure of spoken language. SGEC converts non-native disfluent speech into grammatically correct fluent text, and the main challenge is to operate under extremely limited end-to-end data. The SDD and SLT systems are respectively evaluated on the publicly available Switchboard and MuSTC datasets, and the SGEC system is evaluated on a proprietary LIN corpus. The experiments demonstrate that: the simple cascaded structure gives reasonable baselines for spoken language tasks; the proposed reranking and embedding passing approaches are both effective in propagating richer information and mitigating error propagation under limited end-to-end training corpora.

Acknowledgements

I would like to acknowledge my supervisor, Mark Gales, for his guidance and support throughout my PhD. I would also like to express my gratitude to all the people from the lab, for the discussion and feedback I have relied on during this research. Finally, I would like to thank my family and friends for their support over the years. This PhD has been funded by the ALTA Institute, and supported by Cambridge Assessment.

I would like to dedicate this thesis to my loving parents.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Speech communication has always been the backbone of information exchange in human interactions. Today's computer development attempts to extend such human-human interaction to human-machine interface. Recent research has focused on using deep learning approaches for spoken language processing applications.

1.1 Spoken language processing

Spoken language processing [89] is a broad area encompassing speech production and perception. Speech synthesis generates human-like speech from text inputs [235]. Voice conversion [202, 146] transforms one's voice into another without changing the speech content. Tasks like spoken language translation [152, 195], spoken language understanding [46, 210] and spoken dialogue systems [131, 215] all extract contextual information from the speech sequences, and convert them into textual sequences. The scope of this work mainly concerns the speech-to-text applications.

Speech-to-text applications take speech signals as inputs, and produce text sequences depending on the application purposes. For example, spoken language translation systems convert speech in one language into text in a different language, and spoken dialogue systems produce text responses to speech inquiries from user prompts. One of the essential elements in developing spoken language systems is end-to-end data, with which the models can be trained to extract relevant acoustic context and further produce textual outputs upon requests. Unfortunately, such end-to-end data across modalities is a scare resource for most tasks. For example in spoken language translation, there is little annotated data that directly transcribes speech into a foreign language. On the other hand, there is usually an abundance of annotated data for speech recognition as well as various text processing tasks.

Due to the lack of end-to-end data, spoken language systems are usually broken down into multiple simpler modules, so that individual modules can be trained with sufficient amounts of data from their associated domains. The upstream automatic speech recognition (ASR) module converts speech into transcriptions, followed by some downstream natural language processing (NLP) modules. For example, spoken language translation (SLT) is composed of an ASR module and a machine translation (MT) module, and spoken language understanding (SLU) consists of an ASR module and a natural language understanding (NLU) module. The vanilla cascaded structure allows spoken language tasks to be trained under domains that are associated with individual modules, yet it faces several challenges [89, 195]:

- Mismatched written & spoken languages: Written and spoken languages usually have very different formats and styles. Written language contains punctuations that indicate clear sentence actions and boundaries, yet speech transcripts are unpunctuated and uncapitalised. Spoken language is often delivered in a conversational setting, and thus expected to be less formal. ASR modules are trained using spoken corpora, whereas text processing modules mainly deal with written text. Such mismatched language styles during training are likely to cause degradation in evaluation.

- Disfluencies: Spontaneous spoken language shows a large set of disfluencies such as repetitions, filler words and false starts [191]. ASR systems trained on read speech tend to degrade when encountered with disfluencies. Text processing systems are even more susceptible to spoken disfluencies, with disruptions coming from both disfluencies and errors in ASR transcripts.

- Erroneous early decisions: In a cascaded system, early decisions are required to be made at the upstream modules, and early stage errors would potentially propagate through and degrade the downstream modules. For example, committing to an erroneous ASR hypothesis could disrupt the downstream text processing modules.

- Loss of communicative prosody: Prosodic attributes of utterances provide crucial cues for spoken language understanding. In cascaded systems, the upstream ASR module and the downstream text processing modules are usually connected via speech transcripts, which do not encapsulate any prosodic information. Loss of prosodies would potentially give rise to ambiguity in the downstream text processing.

The focus of this thesis is therefore to explore approaches that mitigates the issues in cascaded multimodular systems, and the goal is to improve spoken language applications under a limited amount of end-to-end training data.

1.2 Thesis organisation

This thesis is organised as follows. Chapter 2 introduces building blocks of deep neural networks, training techniques, as well as sequence modelling approaches. Chapter 3 applies the deep learning techniques to formulating individual modules of spoken language processing tasks, including automatic speech recognition (ASR), disfluency detection (DD), neural machine translation (NMT) and grammatical error correction (GEC). Chapter 4 further discusses challenges in combining multiple modules into a single system, and different approaches are proposed to help improve multimodular systems. Three categories of spoken language systems are considered, including loosely cascaded, tightly integrated, as well as end-to-end systems. Chapter 5 investigates the spoken disfluency detection (SDD) task. An embedding passing approach is adopted to encourage tighter integration between the ASR and DD modules. Chapter 6 investigates the spoken language translation (SLT) task. Error mitigation is applied to improve the cascaded multimodular system. Reranking and embedding passing approaches are also adopted to encourage tighter modular integration between the ASR and NMT modules. Chapter 7 investigates the spoken grammatical error correction (SGEC) task. Error mitigation, reranking and embedding passing approaches are adopted to improve multimodular combination, and further analyses are conducted on feedback quality. Finally, Chapter 8 summaries the work and proposes future research directions.

1.3 Published works

Part of the author's original work have been published during the course of the research:

- Y Lu, MJF Gales, K Knill, P Manakul, L Wang, Y Wang. Impact of ASR performance on spoken grammatical error detection. Annual Conference of the International Speech Communication Association (INTER-SPEECH), 2019.
- Y Lu, MJF Gales, K Knill, P Manakul, Y Wang. Disfluency Detection for Spoken Learner English. Speech and Language Technologies in Education (SLaTE), 2019.
- Y Lu, MJF Gales, Y Wang, Spoken language 'grammatical error correction'. Annual Conference of the International Speech Communication Association (INTER-SPEECH), 2020.
- Y Lu, Y Wang, MJF Gales. Efficient Use of End-to-End Data in Spoken Language Processing. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021.

- Y Lu, S Bannò, MJF Gales. On Assessing and Developing Spoken 'Grammatical Error Correction' Systems. Workshop on Innovative Use of NLP for Building Educational Applications (BEA), 2022.

Chapter 2

Deep Learning Fundamentals

Deep learning refers to a broad range of machine learning approaches that are used to model complex structured data. This chapter mainly introduces fundamental deep learning techniques for sequence modelling, which are widely adopted in spoken language applications. It starts with an overview of the basic units of neural networks including feed-forward networks, recurrent networks, attention mechanisms as well as Transformer blocks. More complex sequence tagging and sequence-to-sequence models are further introduced, which are widely used in speech and language processing. The training of neural networks are then discussed, covering commonly used objective functions for classification, regression as well as sequence tasks, and the gradient descent based optimisation process is also reviewed. Lastly, a group of large-scale pre-training approaches are reviewed, which has achieved great successes on various text processing tasks.

2.1 Neural networks

Deep learning is a class of techniques that allows computational models to learn structured representations of data with multiple levels of abstraction [120]. The most fundamental concept is called neural network, a parametric model that maps an input $\boldsymbol{x}$ to an output $\boldsymbol{y}$:

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) \tag{2.1}$$

where the function f denotes a non-linear mapping, and the parameters $\boldsymbol{\theta}$ are learned to achieve the best function approximation. Regression and classification are two commonly seen tasks modelled using neural networks. For regression tasks, the input $\boldsymbol{x}$ is mapped to a continuous or numerical output. For classification tasks, the input $\boldsymbol{x}$ is mapped to a vector set of categorical output, representing which of the classes the input belongs to. Neural networks

are called networks because f typically represents a composition of different operations [63]. The exact form of f is designed to reflect a set of assumptions about the nature of the mapping between $\boldsymbol{x}$ and $\boldsymbol{y}$. The choice of function f defines the parameters $\boldsymbol{\theta}$ that need to be optimised. It is one of the most important inductive biases introduced in a machine learning algorithm.

The following sections will review four forms of network architectures that are used in this thesis. Deep neural networks (2.1.1) realise conversions between unstructured vector inputs and outputs. Recurrent neural networks (2.1.2) process sequential data, modelling conversions between vector sequences. Attention mechanisms (2.1.3) enhance some parts of the input data over others, converting variable-length inputs into fixed-length outputs. Transformer blocks (2.1.4) extract abstract representations of the input data with much less inductive bias compared to other architectures.

2.1.1 Feed-forward networks

Feed-forward networks (FFNs), also known as multilayer perceptrons (MLPs), is composed of multiple fully-connected feed-forward layers [13, 80]. As illustrated in Figure 2.1, the input of each layer $f^{(k)}$ is the output of its predecessor, and information flows from the input to the output without any feedback connections. Equation 2.1 can thus be expanded as:

$$\boldsymbol{h}^{(1)} = f^{(1)}(\boldsymbol{x}; \boldsymbol{\theta}^{(1)}) \tag{2.2}$$

$$\boldsymbol{h}^{(k)} = f^{(k)}(\boldsymbol{h}^{(k-1)}; \boldsymbol{\theta}^{(k)}) \quad 1 < k < K \tag{2.3}$$

$$\boldsymbol{y} = f^{(K)}(\boldsymbol{h}^{(K-1)}; \boldsymbol{\theta}^{(K)}) \tag{2.4}$$

where k denotes the k^{th} layer (K layers in total). $[\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)} ... \boldsymbol{\theta}^{(K)}]$ forms the parameter set $\boldsymbol{\theta}$ and $\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)} ... \boldsymbol{h}^{(k)}$ are intermediate representations known as hidden states. Each layer can be expressed as a linear mapping followed by a non-linear transformation:

$$\boldsymbol{h}^{(k)} = \sigma_a^{(k)}(\boldsymbol{W}^{(k)} \boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)})) \tag{2.5}$$

The linear mapping is parameterised using $\boldsymbol{\theta}^{(k)} = \{\boldsymbol{W}^{(k)}, \boldsymbol{b}^{(k)}\}$, where $\boldsymbol{W}^{(k)}$ is a pre-multiplication matrix and $\boldsymbol{b}^{(k)}$ is a bias vector. The non-linear transformation $\sigma_a^{(k)}$ is known as an activation function. Two hyperparameters are to be determined for each layer: the activation function $\sigma_a^{(k)}$, and the size of the hidden state $\boldsymbol{h}^{(k)}$, which also determines the size of $\boldsymbol{W}^{(k)}$ and $\boldsymbol{b}^{(k)}$.

Given a sufficiently large hidden state size, a single layer feed-forward network is able to approximate any arbitrary function in theory [120, 86]. However, its modelling power is restricted by some practical issues: the hidden state cannot be infinitely large; and training might not converge to a global minimum. In general, a larger number of hidden layers will

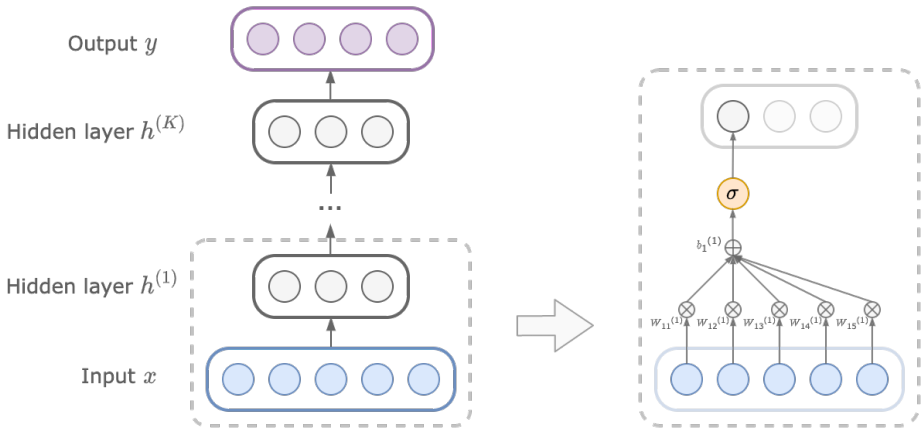


Fig. 2.1 Illustration of a simple deep neural network with K hidden layers: mapping a vector input of size 5 into a vector output of size 4. The left part shows an overview of the feed-forward structure, and the right part demonstrates the operations within each layer (using the first node of layer one $\boldsymbol{h}_1^{(1)}$ as an example).

lead to stronger modelling power [12], and residual connections [74] are commonly adopted to aid convergence in deep networks. To optimally approximate the function, both the number of hidden layers and the size of each layer need to be tuned.

Activation functions

In order for neural networks to compute non-trivial problems using a limited number of nodes, activation functions need to be piecewise continuous and locally bounded [124]. In addition, activations are required to be differentiable since neural networks are typically trained using gradient descent (see Section 2.3).

Commonly used activation functions are sigmoid and softmax functions. The sigmoid function, also known as the logistic function, is usually used for binary classification tasks. It normalises the input into a probability distribution between 0 and 1. Given an input $\boldsymbol{x}$ of dimension D, the sigmoid activation operates on each vector entry x_d:

$$\sigma_{\text{sigmoid}}(x_d) = \frac{e^{x_d} - e^{-x_d}}{e^{x_d} + e^{-x_d}} \tag{2.6}$$

The softmax function is usually used for multi-class classifications. It normalises a set of D inputs into a probability distribution such that they sum up to 1:

$$\sigma_{\text{softmax}}(x_d) = \frac{e^{x_d}}{\sum_{j=1}^{J} e^{x_j}} \tag{2.7}$$

$$\sum_{d=1}^{D} \sigma_{\text{softmax}}(x_d) = 1 \tag{2.8}$$

Hyperbolic tangent and Rectified Linear Unit (ReLU) [148] functions are commonly used for hidden layer activations. The hyperbolic tangent function, expressed as tanh, has a similar S-shape as sigmoid, with a different dynamic range from -1 to 1.

$$\sigma_{\tanh}(x_d) = \frac{1}{1+e^{-x_d}} \tag{2.9}$$

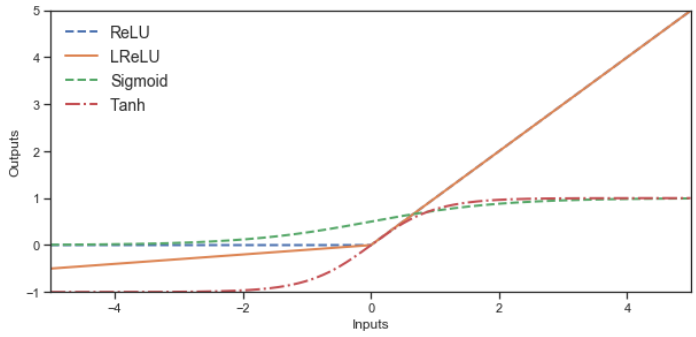


Fig. 2.2 Comparing hidden layer activation functions (α is set at 0.1 for LReLU in this plot)

The tanh function adds non-linearities and prevent extremely large values in hidden layers. However, in deep networks where a large number of hidden layers are used, it suffers from saturation issues. The function becomes flat as the input goes to extreme values, causing slow convergence when used with gradient-based optimisation methods [61]. ReLU is introduced to alleviate this vanishing gradient problem, the positive part of which stays linear when extending to large values:

$$\sigma_{\text{ReLU}}(x_d) = \max(0, x_d) \tag{2.10}$$

One of the limitations of ReLU is known as 'dying ReLU' [138]. Large weight updates could potentially make the activation input become negative, thus causing the output to stay at 0. To prevent the negative part from saturation, a group of modified ReLUs were proposed, such as Exponential Linear Unit (ELU) [38], Parametric ReLU [73], and Leaky Rectified Linear Unit (LReLU) [138]. LReLU adopts a small slope for negative values as opposed to a flat slope:

$$\sigma_{\text{LReLU}}(x_d) = \max(\alpha x_d, x_d) \quad 0 < \alpha < 1 \tag{2.11}$$

Figure 2.2 compares four types of hidden layer activations. Both the sigmoid and tanh functions are S-shaped, and gradually flatten out when extending to extreme values. ReLU and LReLU both maintain constant gradients regardless of the increasing inputs.

Residual connection

The residual connection [74] is a type of skip-connection that learns residual functions with reference to the layer. Such connection can be easily implemented by superimposing the input of each sub-layer to its output:

$$g(\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{x} \tag{2.12}$$

where $\boldsymbol{x}$ is the layer input and $f(\boldsymbol{x})$ is the residual function to be learned. The underlying rationale is that it is easier to optimise the residual mapping than the original mapping. In extreme cases where the identity mapping is to be learned, the residual function $f(\boldsymbol{x})$ can be pushed to zero, as opposed to fitting an identity mapping by a stack of nonlinear layers.

2.1.2 Recurrent neural networks

Recurrent neural networks (RNNs) [178] are commonly used for tasks that involve sequential inputs, such as speech and language. RNNs process the input sequence one element at a time, and maintain a hidden state that contains information about all the elements preceding it (unidirectional) or those both preceding and following it (bidirectional). The hidden states serve as a memory mechanism that enables recurrent networks to process sequential data of variable length, and to scale up to much longer sequences.

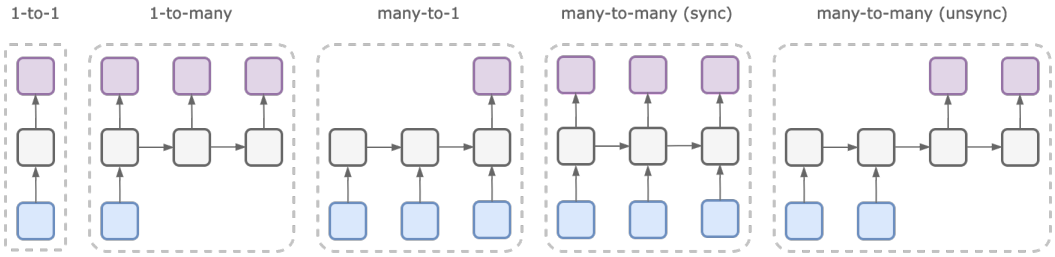


Fig. 2.3 Different types of unidirectional RNN architectures. Blue, grey and purple squares denote the input, hidden and output vectors respectively [102].

Figure 2.3 shows different types of RNNs. The 1-to-1 mapping can be viewed as a vanilla FFN, and the rest of the architectures allow variable-length inputs and outputs. Different types of RNNs have different use cases, and some examples are given as follows. 1-to-1 can model image classification tasks. 1-to-many can model image captioning tasks where the input is a single vector image and the output is a sequence of words. Many-to-1 can model sentiment analysis where the input is a word sequence and the output is a sentiment class. Synchronised many-to-many can model sentence segmentation where the input is a word sequence and the output is a tag sequence indicating whether to put a full stop after each

word ('synchronised' means the output sequence has a one-to-one correspondence with the input sequence), and unsynchronised many-to-many can be used for machine translation.

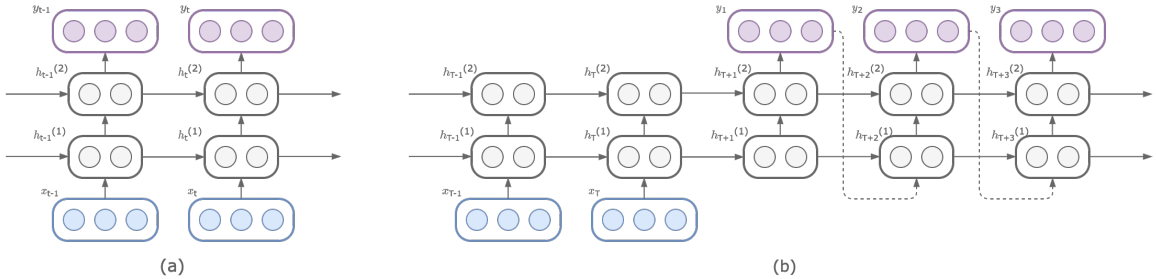


Fig. 2.4 Illustration of RNN architectures (a) two-layer synchronised many-to-many RNNs (b) two-layer unsynchronised many-to-many RNNs

A more detailed illustration of RNNs is shown in Figure 2.4, using the unidirectional two-layer many-to-many RNN as an example. Given an input sequence $\boldsymbol{x}_{1:T}$, the synchronised output sequence is $\boldsymbol{y}_{1:T}$ and the unsynchronised output sequence is $\boldsymbol{y}_{1:N}$. The hidden state $\boldsymbol{h}_t^{(k)}$ at time step t and layer k is dependent on the hidden state of the previous layer at the same time step $\boldsymbol{h}_{t-1}^{(k)}$, as well as the previous time step at the same layer $\boldsymbol{h}_t^{(k-1)}$.

$$\boldsymbol{h}_t^{(k)} = f^{(k)}(\boldsymbol{h}_t^{(k-1)}, \boldsymbol{h}_{t-1}^{(k)}; \boldsymbol{\theta}^{(k)}) \quad 1 < k < K \tag{2.13}$$

In the synchronised case, the first hidden layer is dependent on the input sequence and the output sequence is dependent on the last hidden layer, both at the aligned time steps:

$$\boldsymbol{h}_t^{(1)} = f^{(1)}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}^{(1)}; \boldsymbol{\theta}^{(1)}) \tag{2.14}$$

$$\boldsymbol{y}_t = f^{(K)}(\boldsymbol{h}_t^{(K-1)}; \boldsymbol{\theta}^{(K)}) \tag{2.15}$$

Unsynchronised RNNs can be divided into two stages. The first stage converts the input into a sequence of hidden states, and the structure remains similar to the synchronised case. The second stage generates predictions auto-regressively, and the first hidden layer takes the output predicted at the previous time step as its input:

$$\boldsymbol{y}_1 = f^{(K)}(\boldsymbol{h}_{T+1}^{(K-1)}; \boldsymbol{\theta}^{(K)}) \tag{2.16}$$

$$\boldsymbol{h}_{T+2}^{(1)} = f^{(1)}(\boldsymbol{y}_1, \boldsymbol{h}_{T+1}^{(1)}; \boldsymbol{\theta}^{(1)}) \tag{2.17}$$

$$\boldsymbol{y}_2 = f^{(K)}(\boldsymbol{h}_{T+2}^{(K-1)}; \boldsymbol{\theta}^{(K)}) \tag{2.18}$$

Bidirectional RNNs

Unidirectional RNNs only account for information from the elements preceding the current node. Bidirectional RNNs [186] are used to capture information from both the past and the future elements. Figure 2.5 illustrates a simple two-layer bidirectional RNN structure, which consists of a pair of unidirectional RNNs running in opposite directions.

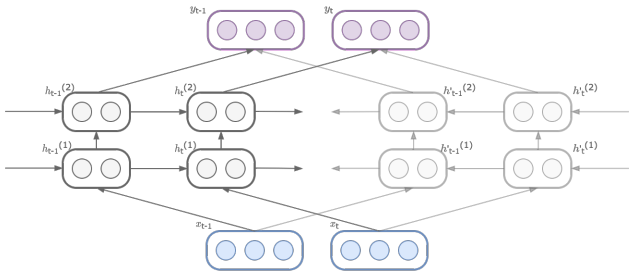


Fig. 2.5 Illustration of a two-layer bidirectional synchronised many-to-many RNN

At each time step t, the forward and backward networks recur in opposite directions, and the joint hidden state $\boldsymbol{h}_t$ is a concatenation of the forward and backward hidden vectors at the last hidden layer. Equation 2.19 and 2.20 describe the forward and backward hidden state generation respectively, and Equation 2.21 describes the joint hidden states.

$$\overrightarrow{\boldsymbol{h}}_t^{(k)} = \overrightarrow{f}^{(k)}(\overrightarrow{\boldsymbol{h}}_t^{(k-1)}, \overrightarrow{\boldsymbol{h}}_{t-1}^{(k)}; \overrightarrow{\boldsymbol{\theta}}^{(k)}) \tag{2.19}$$

$$\overleftarrow{\boldsymbol{h}}_t^{(k)} = \overleftarrow{f}^{(k)}(\overleftarrow{\boldsymbol{h}}_t^{(k-1)}, \overleftarrow{\boldsymbol{h}}_{t-1}^{(k)}; \overleftarrow{\boldsymbol{\theta}}^{(k)}) \tag{2.20}$$

$$\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}}_t^{(K-1)}, \overleftarrow{\boldsymbol{h}}_t^{(K-1)}] \tag{2.21}$$

RNNs, LSTMs and GRUs

Conventional RNNs adopts feed-forward layers, and each layer is parameterised using a feed-forward matrix $\boldsymbol{W}_f^{(k)}$, a recurrent matrix $\boldsymbol{W}_r^{(k)}$ and a bias $\boldsymbol{b}_r^{(k)}$.

$$\boldsymbol{h}_t^{(k)} = f^{(k)}(\boldsymbol{h}_t^{(k-1)}, \boldsymbol{h}_{t-1}^{(k)}; \boldsymbol{\theta}^{(k)}) = \sigma^{(k)}(\boldsymbol{W}_f^{(k)}\boldsymbol{h}_t^{(k-1)} + \boldsymbol{W}_r^{(k)}\boldsymbol{h}_{t-1}^{(k)} + \boldsymbol{b}^{(k)}) \tag{2.22}$$

Deep feed-forward networks tend to suffer from gradient saturation issues, and a key constraint of vanilla RNNs is their difficulty in maintaining long-term dependencies. At each time step, the gradients either grow or shrink, and thus the information at an early position will either explode or decay to zero over a long sequence of propagation [82]. Gated Recurrent Unit (GRU) [33] and Long Short Term Memory (LSTM) [83] are proposed to mitigate this long-term dependency issue. Both LSTM and GRU store previous activation values in long sequences. Each unit consists of multiple gates that are used for controlling

the information flow. Gates are capable of learning what to remember and what to forget in the memory unit at each time step.

A vanilla RNN block pass on information from the previous hidden state and the input through a simple tanh activation. In the first hidden layer, the input to the RNN block is the system input $\boldsymbol{x}_t$. In the k^{th} layer ($k > 1$), the input is the hidden state from the layer below at the same time instance $\boldsymbol{h}_t^{(k-1)}$. For simplicity purposes, the equations listed below only describe cases for layer $k = 1$, and they can be easily generalised to $k > 1$ cases by replacing $\boldsymbol{x}_t$ using $\boldsymbol{h}_t^{(k-1)}$. To describe vanilla RNNs, Equation 2.22 can be simplified as:

$$\boldsymbol{h}_t = \sigma_{\text{tanh}}(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{W}_r \boldsymbol{h}_{t-1} + \boldsymbol{b}) \tag{2.23}$$

A GRU block contains two gates: an update gate and a reset gate. The update gate determines whether to update the cell state with the current activation. The reset gate determines whether the previous cell state is important. The state transition is described as:

$$\boldsymbol{z}_t = \sigma_{\text{sigmoid}}(\boldsymbol{W}_f^z \boldsymbol{x}_t + \boldsymbol{W}_r^z \boldsymbol{h}_{t-1} + \boldsymbol{b}^z) \tag{2.24}$$

$$\boldsymbol{r}_t = \sigma_{\text{sigmoid}}(\boldsymbol{W}_f^r \boldsymbol{x}_t + \boldsymbol{W}_r^r \boldsymbol{h}_{t-1} + \boldsymbol{b}^r) \tag{2.25}$$

$$\tilde{\boldsymbol{h}}_t = \sigma_{\text{tanh}}(\boldsymbol{W}_f^y \boldsymbol{x}_t + \boldsymbol{W}_r^y [\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}] + \boldsymbol{b}^y) \tag{2.26}$$

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t \tag{2.27}$$

where $\odot$ denotes element-wise multiplication. $\boldsymbol{z}_t$, $\boldsymbol{r}_t$ denote the values generated from the update and the reset gates respectively. $\boldsymbol{h}_{t-1}$ is the in-going hidden state, $\tilde{\boldsymbol{h}}_t$ is an intermediate state activation, and $\boldsymbol{h}_t$ is the out-going hidden state.

LSTM uses a second hidden state, called memory cell $\boldsymbol{c}_t$. It acts like an accumulator, accounting for three element-wise gating functions: forget $\boldsymbol{f}_t$, input $\boldsymbol{i}_t$ and output $\boldsymbol{o}_t$ gates.

$$\boldsymbol{f}_t = \sigma_{\text{sigmoid}}(\boldsymbol{W}_f^f \boldsymbol{x}_t + \boldsymbol{W}_r^f \boldsymbol{h}_{t-1} + \boldsymbol{b}^f + \boldsymbol{W}_m^f \boldsymbol{c}_{t-1}) \tag{2.28}$$

$$\boldsymbol{i}_t = \sigma_{\text{sigmoid}}(\boldsymbol{W}_f^i \boldsymbol{x}_t + \boldsymbol{W}_r^i \boldsymbol{h}_{t-1} + \boldsymbol{b}^i + \boldsymbol{W}_m^i \boldsymbol{c}_{t-1}) \tag{2.29}$$

$$\boldsymbol{o}_t = \sigma_{\text{sigmoid}}(\boldsymbol{W}_f^o \boldsymbol{x}_t + \boldsymbol{W}_r^o \boldsymbol{h}_{t-1} + \boldsymbol{b}^o + \boldsymbol{W}_m^o \boldsymbol{c}_{t-1}) \tag{2.30}$$

$$\tilde{\boldsymbol{c}}_t = \sigma_{\text{tanh}}(\boldsymbol{W}_f^c \boldsymbol{x}_t + \boldsymbol{W}_r^c \boldsymbol{h}_{t-1} + \boldsymbol{b}^c) \tag{2.31}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{2.32}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \sigma_{\text{tanh}}(\boldsymbol{c}_t) \tag{2.33}$$

where the subscripts f, r, m denote feed-forward, recurrent and memory functions; and the superscripts f, i, o denote forget, input and output gates respectively.

2.1.3 Attention mechanisms

Attention is motivated by how human pay attention to different regions of an image or correlations between words in a sentence [216]. In deep learning, attention mechanisms can be considered as a vector of importance weights. For example, when predicting a word in a sentence, the attention vector indicates how strongly it is correlated with other words, and the weighted sum of their values is used as the approximation of the target. The first attention mechanism was introduced by Bahdanau et al. [6]. It addresses the bottleneck problem that arises in neural machine translation, where the output only has limited access to the information provided by the input. In the simplest form, an attention mechanism compresses a variable-length input sequence $\boldsymbol{x}_{1:T}$ into a fixed length context vector $\boldsymbol{c}$:

$$\boldsymbol{c} = \sum_{t=1}^{T} \alpha_t \boldsymbol{x}_t \quad \sum_{t=1}^{T} \alpha_t = 1 \tag{2.34}$$

$$\alpha_t = \frac{\exp(f_{\text{att}}(\boldsymbol{x}_t; \boldsymbol{\theta}_{\text{att}}))}{\sum_{t'=1}^{T} \exp(f_{\text{att}}(\boldsymbol{x}_{t'}; \boldsymbol{\theta}_{\text{att}}))} \tag{2.35}$$

where α_t denotes the relative saliency between $\boldsymbol{c}$ and $\boldsymbol{x}_t$, and the attention function f_{att} is a fully-connected layer. The scores $\alpha_{1:T}$ can be viewed as an importance distribution over the input sequence $\boldsymbol{x}_{1:T}$, and thus its elements sum to one. More advanced forms of attention mechanisms (discussed below) are commonly adopted for sequence-to-sequence modelling.

Cross-attention

Cross-attention usually operates on two different sequences, an example of which is the encoder-decoder attention in sequence-to-sequence tasks (see Section 2.2.2). Given an input sequence $\boldsymbol{x}_{1:T}$ and an output sequence $\boldsymbol{y}_{1:N}$, the attention mechanism generates an alignment score vector $\boldsymbol{\alpha}_n = [\alpha_{n,1}, ..., \alpha_{n,T}]^T$ at each step n. Each vector entry $\alpha_{n,t}$ accounts for the relative saliency of the input element $\boldsymbol{x}_t$ to the output element $\boldsymbol{y}_n$. The input $\boldsymbol{x}_{1:T}$ can therefore be mapped into a context vector $\boldsymbol{c}_n$ using weights from $\boldsymbol{\alpha}_n$. The context sequence $\boldsymbol{c}_{1:N}$ extracts relevant information from the input sequence $\boldsymbol{x}_{1:T}$ according to the output $\boldsymbol{y}_{1:N}$.

$$\boldsymbol{c}_n = \sum_{t=1}^{T} \alpha_{n,t} \boldsymbol{x}_t \quad \sum_{t=1}^{T} \alpha_{n,t} = 1 \tag{2.36}$$

$$\alpha_{n,t} = \frac{\exp(f_{\text{att}}(\boldsymbol{y}_n, \boldsymbol{x}_t; \boldsymbol{\theta}_{\text{att}}))}{\sum_{t'=1}^{T} \exp(f_{\text{att}}(\boldsymbol{y}_n, \boldsymbol{x}_{t'}; \boldsymbol{\theta}_{\text{att}}))} \tag{2.37}$$

where $\alpha_{n,t}$ is obtained by normalising the output of the attention function f_{att} parameterised with $\boldsymbol{\theta}_{\text{att}}$. The attention function is central to the attention mechanism, and can take various forms. The initial work on attention mechanisms proposes the additive function [6], which

uses a fully-connected feed-forward layer to compute saliencies:

$$f_{\text{att}}(\boldsymbol{y}_n, \boldsymbol{x}_t; \boldsymbol{\theta}_{\text{att}}) = \boldsymbol{u}^T \sigma_{\tanh}(\boldsymbol{W}[\boldsymbol{y}_n, \boldsymbol{x}_t]) \quad \boldsymbol{\theta}_{\text{att}} = \{\boldsymbol{u}, \boldsymbol{W}\} \tag{2.38}$$

where the output element $\boldsymbol{y}_n$ and input element $\boldsymbol{x}_t$ are concatenated, and fed through a single-layer feed-forward network. The additive attention does not account for positional information, thus agnostic to the order of $\boldsymbol{x}_{1:T}$. Attention mechanisms reflect correlations between the elements in two sequences, and the network is usually jointly trained as part of a sequence-to-sequence model. Following additive attention, several alternatives are introduced in the literature. Table 2.1 summarises various forms of popular attention functions.

Name	$f_{\text{att}}(\boldsymbol{y}_n, \boldsymbol{x}_t; \boldsymbol{\theta}_{\text{att}})$
Additive [6]	$\boldsymbol{u}^T \sigma_{\tanh}(\boldsymbol{W}[\boldsymbol{y}_n, \boldsymbol{x}_t])$
Cosine [68]	$\beta(\boldsymbol{y}_n^T \boldsymbol{x}_t)/(\|\boldsymbol{y}_n\|\|\boldsymbol{x}_t\|)$
General [136]	$\boldsymbol{y}_n^T \boldsymbol{W} \boldsymbol{x}_t$
Dot-product [136]	$\boldsymbol{y}_n^T \boldsymbol{x}_t$
Scaled dot-product [203]	$\boldsymbol{y}_n^T \boldsymbol{x}_t / \sqrt{D_x}$

Table 2.1 Popular attention functions. β is a hyperparameter which amplifies or attenuates the precision of the focus. D_x is the dimension of input $\boldsymbol{x}_t$.

A more general formulation of attention mechanisms makes use of three main components, namely queries $\boldsymbol{q}_{1:N}$, keys $\boldsymbol{k}_{1:T}$, and values $\boldsymbol{v}_{1:T}$ [203]. To compare these three components to the attention mechanism described above, the query is analogous to the output $\boldsymbol{y}_{1:N}$, and the value is analogous to the input $\boldsymbol{x}_{1:T}$. Intuitively, each query can carry as much information from the values by weighting them with the keys. Keys and values have the same length, since they can be viewed as two representations of the same sequence. In most cases, keys and values are set to be identical. Equation 2.36 and 2.37 can be expressed as:

$$\boldsymbol{c}_n = \textstyle\sum_{t=1}^{T} \alpha_{n,t} \boldsymbol{v}_t \tag{2.39}$$

$$\alpha_{n,t} = \frac{\exp(f_{\text{att}}(\boldsymbol{q}_n, \boldsymbol{k}_t; \boldsymbol{\theta}_{\text{att}}))}{\sum_{t'=1}^{T} \exp(f_{\text{att}}(\boldsymbol{q}_n, \boldsymbol{k}_{t'}; \boldsymbol{\theta}_{\text{att}}))} \tag{2.40}$$

The attention mechanism captures saliencies between the query $\boldsymbol{q}_n$ and the keys $\boldsymbol{k}_{1:T}$. The values $\boldsymbol{v}_{1:T}$ are then reweighed using the generated attention scores $\boldsymbol{\alpha}_n$, and the weighted sum becomes the context vector $\boldsymbol{c}_n$. The attention function f_{att} stays the same as described in Table 2.1.

Self-attention

Different from the cross-attention mechanism that extracts relative saliencies between the key and the query sequences, self-attention [30, 159] is proposed to extract relations between different positions of the same sequence, which is especially useful for feature representation generation. It can be considered as a special case of the general attention, where the key, value and query sequences are identical:

$$\boldsymbol{k}_{1:T} = \boldsymbol{v}_{1:T} = \boldsymbol{q}_{1:N} \quad N = T \tag{2.41}$$

Equation 2.39 and 2.40 can therefore be modified as:

$$\boldsymbol{c}_n = \sum_{n'=1}^{N} \alpha_{n,n'} \boldsymbol{k}_{n'} \tag{2.42}$$

$$\alpha_{n,n'} = \frac{\exp(f_{\text{att}}(\boldsymbol{k}_n, \boldsymbol{k}_{n'}; \boldsymbol{\theta}_{\text{att}}))}{\sum_{n''=1}^{N} \exp(f_{\text{att}}(\boldsymbol{k}_n, \boldsymbol{k}_{n''}; \boldsymbol{\theta}_{\text{att}}))} \tag{2.43}$$

The context sequence $\boldsymbol{c}_{1:N}$ has the same length as the key sequence $\boldsymbol{k}_{1:N}$, and it can be viewed as a better representation of the keys. Scaled dot-product (see Table 2.1) is a common choice for the attention function f_{att} in self-attention. It helps mitigate problems caused by $\boldsymbol{c}_n$ becoming too large when the vector dimension D_k gets large. Compared with RNNs, using self-attention in representation learning reduces inductive bias, i.e. when extracting saliencies between different elements in one sequence, self-attention does not make assumptions about the neighbouring elements being more important compared to the further away elements.

Multi-head attention

Vanilla attention mechanisms only use one attention function at a time, whereas multi-head attention [203] combines multiple attention mechanisms in parallel. An attention 'head' is an attention mechanism with one set of parameters, and multi-head attention consists of multiple heads with the same structure, but different parameters. Figure 2.6 illustrates the multi-head attention mechanism and Equation 2.44 gives the expression at step n.

$$\boldsymbol{c}_n = \boldsymbol{W}_c[\boldsymbol{c}_n^1, \boldsymbol{c}_n^2, ..., \boldsymbol{c}_n^H] \tag{2.44}$$

where $\boldsymbol{c}_n$ is the joint context vector at step n, and the attention context $\boldsymbol{c}_n^h$ is computed following Equation 2.39 and 2.40. A total of H heads are simply concatenated and linearly transformed into the expected dimensions using $\boldsymbol{W}_c$. The intuition behind is that the multiple heads expand the model's ability to focus on different positions, and they also give the

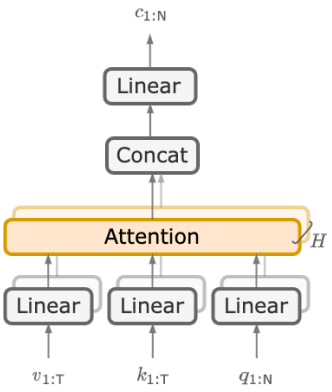


Fig. 2.6 Illustration of multi-head attention [203]

attention layer multiple representation subspaces [203]. Both self-attention and multi-head attention are essential mechanisms in Transformer blocks (see Section 2.1.4).

2.1.4 Transformer blocks

Transformer blocks proposed in 'Attention is all you need' [203] are entirely built on self-attention mechanisms without using sequence-aligned recurrent networks. They improve upon traditional attention mechanisms and make it possible to model sequence tasks without recurrent units. The basic Transformer block (known as the Transformer encoder block in [203]) is used to form a better feature representation of an input sequence.

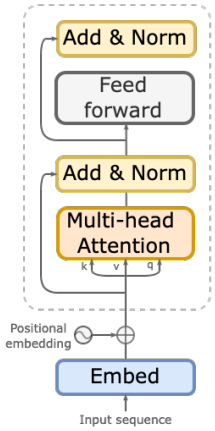


Fig. 2.7 Illustration of the basic Transformer block. k, v, q denotes keys, values and queries in the multi-head attention mechanism.

Figure 2.7 shows an illustration of the basic Transformer block. An input sequence is first embedded into continuous feature representations, superimposed with positional embeddings, and then passed to the Transformer block. A Transformer block is composed of two layers:

a multi-head self-attention layer, and a simple fully-connected feed-forward network. The multi-head self-attention (described in Section 2.1.3) adopts scaled dot-product attention (in Table 2.1) to avoid context vector becoming too large when multiple Transformer blocks are stacked together. The feed-forward layer typically consists of two linear transformations with a ReLU non-linearity in between. All sub-layers (attention, feed-forward layers) adopt a residual connection followed by layer normalisation. Transformer blocks are designed to handle sequential data. With the use of self-attention mechanisms, they allow parallel processing of all positions in the input sequence. Compared with RNNs, Transformer blocks also lift the inductive bias exerted upon neighbouring elements.

Layer normalisation

Layer normalisations [3] normalise the activations of a layer for each instance in a batch independently. The output vector is normalised so that its mean is zero and its standard deviation is one. Element-wise scaling and shifting are then applied to the normalised vector.

$$g(\boldsymbol{x}) = \boldsymbol{a} \odot \frac{f(\boldsymbol{x}) - \boldsymbol{\mu}}{\sigma} + \boldsymbol{b} \tag{2.45}$$

$$\mu = \frac{1}{D}\sum_{d=1}^{D} f(\boldsymbol{x})_d \tag{2.46}$$

$$\sigma^2 = \frac{1}{D}\sum_{d=1}^{D} (f(\boldsymbol{x})_d - \mu)^2 \tag{2.47}$$

where $\boldsymbol{\mu}$ and σ are the element-wise mean and standard deviation of the layer output $f(\boldsymbol{x})$; d is the element index and D is the vector dimension; $\boldsymbol{a}$ and $\boldsymbol{b}$ are learnable parameters used for scaling and shifting; $g(\boldsymbol{x})$ is the final layer output. Layer normalisation directly estimates the statistics within each layer so that the process does not introduce any new dependencies between training cases. It works well for both RNNs and Transformers in improving the training time as well as the generalisation performance [3].

Positional embedding

As discussed in Section 2.1.3, self-attention is permutation invariant, and thus positional encodings are important to provide order information to the model. The positional embedding $\boldsymbol{e}^p$ has the same dimension as the textual embedding $\boldsymbol{e}^t$, and they can be directly superimposed. The standard Transformer block uses sinusoidal functions of varying frequencies:

$$\boldsymbol{e} = \boldsymbol{e}^t + \boldsymbol{e}^p \tag{2.48}$$

$$e^p_{n,d} = \begin{cases} \sin(\frac{n}{10000^{d/D}}) & \text{if } d \text{ is even} \\ \cos(\frac{n}{10000^{d/D}}) & \text{if } d \text{ is odd} \end{cases} \tag{2.49}$$

where $e^p_{n,d}$ denotes the d^{th} element of the positional embedding $\boldsymbol{e}^p_n$, and D denotes the embedding dimension. Figure 2.8 shows an illustration of the sinusoidal embedding. Each dimension of the positional embedding corresponds to a sinusoidal wave of different wavelengths in different dimensions, from 2π to $10000 \cdot 2\pi$. The sinusoidal cyclic formulation allows the model to extrapolate to sequence lengths longer than the ones encountered during training [203]. Another widely used encoding is the learned positional embedding [59], which assigns each element with a learned vector that encodes its absolute position.

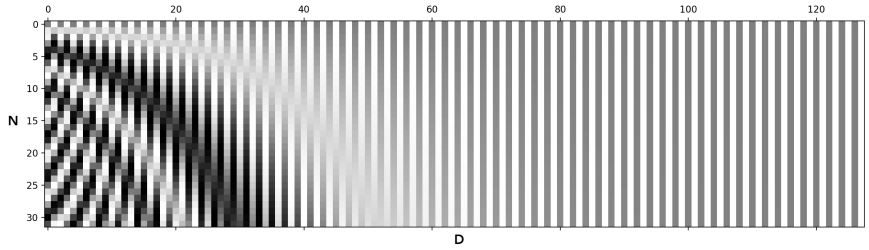


Fig. 2.8 Sinusoidal positional embedding with N=32 and D=128. The value is between -1 (black) and 1 (white), and the value 0 is in grey [218].

2.2 Sequence models

Sequence data is commonly seen in day-to-day life, e.g. speech and text are both sequential. The previous section reviewed different types of neural networks. In this section, more advanced sequence models are discussed. To keep the discussion more general, the input is set as a vector sequence $\boldsymbol{x}_{1:T}$, and the output form depends on the nature of the task.

2.2.1 Sequence tagging

Sequence tagging encompasses a variety of tasks, e.g. part-of-speech tagging, named entity recognition. Given an input vector sequence $\boldsymbol{x}_{1:T}$, a sequence tagger maps each input vector $\boldsymbol{x}_t$ into a set of class labels $\boldsymbol{y}_t$, forming an output sequence $\boldsymbol{y}_{1:T}$:

$$\hat{\boldsymbol{y}}_{1:T} = \text{argmax}_{\boldsymbol{y}_{1:T} \in \mathcal{Y}} P(\boldsymbol{y}_{1:T} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \tag{2.50}$$

Sequence tagging models usually consist of two steps: a context extraction step followed by a classification step. Context dependency plays a significant role in modelling sequential data, and the context extraction step generates a contextual feature of each input token conditioned

on the entire input sequence:

$$\boldsymbol{h}_{1:T} = f_h(\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_h) \tag{2.51}$$

where f_h denotes the context extraction function, and $\boldsymbol{h}_{1:T}$ denotes the contextual feature. The two commonly used context extraction architectures are RNNs and Transformer blocks (discussed in Section 2.1.2 and 2.1.4). Recurrent networks extract contextual dependencies via recurrent units, and bidirectional LSTM is a popular choice [240, 130, 116]. Although widely used, RNNs suffer from long training time due to their recurrent nature. Transformer blocks have been reported to perform poorly on sequence tagging tasks [71], since the direction and relative distance information is lost in the original sinusoidal position embedding [227]. A learned positional embedding can be used to address this problem. Transformer blocks replace recurrent units with self-attention mechanisms, thus enabling parallel training regardless of how long the input sequence is. This allows the model to be trained on a large amount of data to learn a strong language model, and helps generate better quality contextual features $\boldsymbol{h}_{1:T}$.

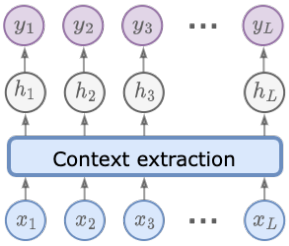


Fig. 2.9 Illustration of the softmax classification head for sequence tagging models

The classifier then takes the contextual feature $\boldsymbol{h}_{1:T}$ as the input, under the assumption that the output is no longer dependent of the system input $\boldsymbol{x}_{1:T}$, and makes predictions for the output tag sequence $\boldsymbol{y}_{1:T}$:

$$\begin{aligned} P(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}) &\approx P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{1:T}; \boldsymbol{\theta}_y) \\ &= \textstyle\prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{h}_{1:T}; \boldsymbol{\theta}_y) \end{aligned} \tag{2.52}$$

Two commonly used classification heads are softmax and conditional random field (CRF) [113], and softmax (Figure 2.9) is adopted throughout this thesis. Directly applying the softmax function makes the assumption that given the contextual feature $\boldsymbol{h}_t$, the current tag prediction $\boldsymbol{y}_t$ is independent of the rest of the sequence and the previous predictions $\boldsymbol{y}_{1:t-1}$. Thus, the autoregressive sequence tagging is simplified into a non-autoregressive process, and therefore

Equation 2.52 can be simplified as:

$$P(\boldsymbol{y}_{1:T}|\boldsymbol{h}_{1:T};\boldsymbol{\theta}_y) \approx \prod_{t=1}^{T} P(\boldsymbol{y}_t|\boldsymbol{h}_t;\boldsymbol{\theta}_y) \tag{2.53}$$

$$P(\boldsymbol{y}_t|\boldsymbol{h}_t;\boldsymbol{\theta}_y) = \sigma_{\text{softmax}}(\boldsymbol{W}_y\boldsymbol{h}_t + \boldsymbol{b}_y) \tag{2.54}$$

where $\boldsymbol{W}_y$ and $\boldsymbol{b}_y$ are the weight matrix and the bias vector.

2.2.2 Sequence-to-sequence

Sequence-to-sequence tasks include automatic speech recognition (see Section 3.2.2), neural machine translation (see Section 3.4) and text-to-speech synthesis etc. The models learn mappings from an input sequence $\boldsymbol{x}_{1:T}$ to an output sequence $\boldsymbol{y}_{1:N}$:

$$\hat{\boldsymbol{y}}_{1:N} = \text{argmax}_{\boldsymbol{y}_{1:N}\in\mathcal{Y}} P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \tag{2.55}$$

In the scope of this thesis, it is assumed that the output generation is an autoregressive process, i.e. the current prediction is conditioned on all the previous predictions ($\boldsymbol{y}_0$ is usually initialised using a special begin-of-sequence token that is independent of the input sequence):

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) = \prod_{n=1}^{N} P(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1},\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \tag{2.56}$$

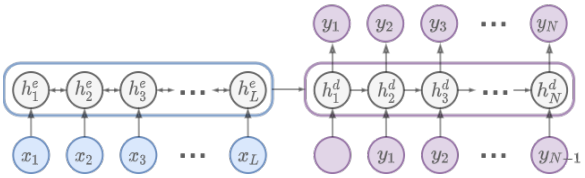


Fig. 2.10 Illustration of a vanilla encoder decoder sequence-to-sequence model. The blue block indicates the encoder process, and purple indicates the decoder process.

A vanilla formulation of sequence-to-sequence tasks is to use an encoder to extract contextual features from the input sequence, and a decoder to auto-regressively generate predictions based on the encoder states. Figure 2.10 illustrates this encoder decoder process, and it can be formulated as:

$$\boldsymbol{h}^e_{1:T} = f_{\text{enc}}(\boldsymbol{x}_{1:T};\boldsymbol{\theta}_{\text{enc}}) \tag{2.57}$$

$$\boldsymbol{h}^d_n = f_{\text{dec}}(\boldsymbol{h}^d_{n-1},\boldsymbol{y}_{n-1};\boldsymbol{\theta}_{\text{dec}}) \quad \boldsymbol{h}^d_1 = \boldsymbol{h}^e_T \tag{2.58}$$

$$P(\boldsymbol{y}_n|\boldsymbol{y}_{1:n-1},\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(\boldsymbol{y}_n|\boldsymbol{h}^d_n;\boldsymbol{\theta}_y) = f_y(\boldsymbol{h}^d_n;\boldsymbol{\theta}_y) \tag{2.59}$$

$$\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{enc}},\boldsymbol{\theta}_{\text{dec}},\boldsymbol{\theta}_y\} \tag{2.60}$$

where $\boldsymbol{h}^e_{1:T}$ and $\boldsymbol{h}^d_{1:N}$ are the encoder and decoder hidden states. The current decoder state $\boldsymbol{h}^d_n$ only depends on the previous decoder state $\boldsymbol{h}^d_{n-1}$ and the previous prediction $\boldsymbol{y}_{n-1}$. The initial decoder state $\boldsymbol{h}^d_1$ is set to be the last encoder hidden state $\boldsymbol{h}^e_T$, which summarises the entire input sequence. The only connection that passes information from the encoder to the decoder is $\boldsymbol{h}^e_T$, yet a single vector is not sufficient in passing contextual information from the input sequence [136, 68]. Attention-based models are therefore introduced to address this issue.

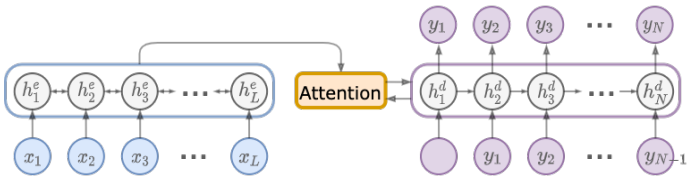


Fig. 2.11 Illustration of an attention-based encoder decoder sequence-to-sequence model

An attention-based encoder-decoder framework is illustrated in Figure 2.11. As the name suggests, it consists of three components: an encoder, a decoder and an attention mechanism connecting the two. The encoder extracts contextual information from the input sequence and generates a contextual feature vector sequence. The attention mechanism computes the relative saliencies between the input and output sequences, and feed the alignment information to the decoder. The decoder compresses the encoded sequence into a fixed length vector at each time step, and generates the output. The attention-based encoder decoder process is formulated as:

$$\boldsymbol{h}^e_{1:T} = f_{\text{enc}}(\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_{\text{enc}}) \tag{2.61}$$

$$\boldsymbol{\alpha}_n = f_{\text{att}}(\boldsymbol{h}^d_{n-1}, \boldsymbol{h}^e_{1:T}; \boldsymbol{\theta}_{\text{att}}) \quad \boldsymbol{c}_n = \textstyle\sum_{t=1}^{T} \alpha_{n,t} \boldsymbol{h}^e_t \tag{2.62}$$

$$\boldsymbol{h}^d_n = f_{\text{dec}}(\boldsymbol{h}^d_{n-1}, \boldsymbol{y}_{n-1}, \boldsymbol{c}_n; \boldsymbol{\theta}_{\text{dec}}) \tag{2.63}$$

$$P(\boldsymbol{y}_n | \boldsymbol{y}_{1:n-1}, \boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \approx P(\boldsymbol{y}_n | \boldsymbol{h}^d_n, \boldsymbol{c}_n; \boldsymbol{\theta}_y) = f_y(\boldsymbol{h}^d_n, \boldsymbol{c}_n; \boldsymbol{\theta}_y) \tag{2.64}$$

$$\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{enc}}, \boldsymbol{\theta}_{\text{att}}, \boldsymbol{\theta}_{\text{dec}}, \boldsymbol{\theta}_y\} \tag{2.65}$$

where $\boldsymbol{\alpha}_n$ represents the relative saliency weights of the decoder state $\boldsymbol{h}^d_{n-1}$ and the encoder state sequence $\boldsymbol{h}^e_{1:T}$; the context vector $\boldsymbol{c}_n$ is a weighted sum over the encoder states according to the predicted weights $\boldsymbol{\alpha}_n$; the prediction step of $\boldsymbol{y}_n$ is conditioned on both the decoder state $\boldsymbol{h}^d_n$ and the contextual feature $\boldsymbol{c}_n$. The use of attention mechanism allows the decoder to access the entire encoder state sequence, and generates a different saliency weight distribution at different prediction step n. With such enhanced dependencies between the input and output sequences, the model gains stronger modelling power, and handles longer sequences better.

Conventional sequence-to-sequence models are formulated using recurrent units such as LSTMs, whose recurrent nature allows the network to extract the bidirectional context of a sequence. However, when faced with long sequences, recurrent networks tend to suffer from the 'forgetting' issue and take a long time to train. The Transformer architecture [203] replaces recurrent units with self-attention mechanisms. In a Transformer-based sequence-to-sequence model, two types of blocks are defined: an encoder block forms a contextual feature of an input sequence, and a decoder block extracts relative saliencies between two sequences. Figure 2.12 shows an illustration of the Transformer encoder decoder structure, as well as their inter-block connections. The Transformer encoder consists of a stack of N identical encoder blocks, and the decoder has N identical decoder blocks.

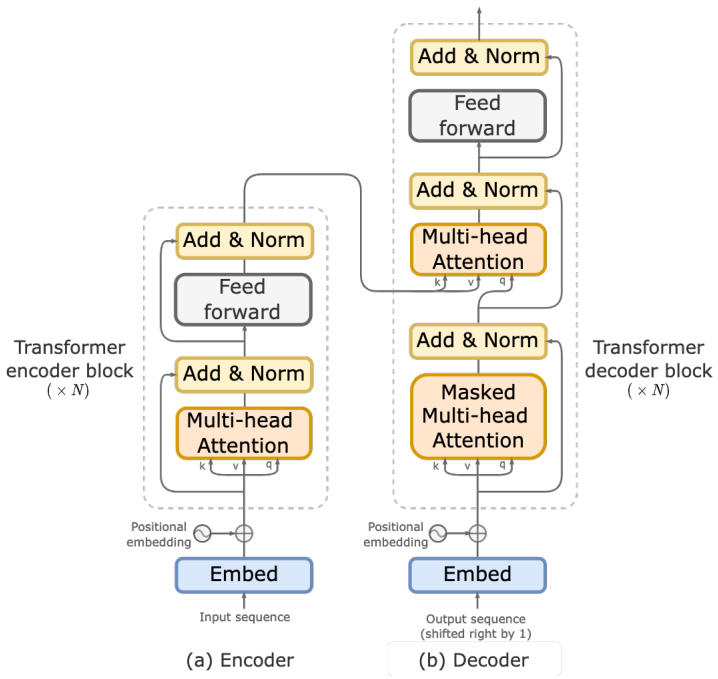


Fig. 2.12 Illustration of Transformer-based sequence-to-sequence models [203]. N denotes the number of blocks in the encoder and the decoder.

Each encoder block (discussed in Section 2.1.4) consists of a multi-head self-attention layer and a fully-connected feed-forward network. Compared with the encoder block, the decoder block has two main differences. It has an extra multi-head cross-attention layer between the self-attention and the feed-forward layers. The cross-attention is used to extract the alignment between input and output sequences, where the key (and value) is the output from the Transformer encoder and the query is the output of the decoder self-attention. Another change from the encoder is that the decoder self-attention is imposed with a triangular mask (see Figure 2.13) such that the current element is prevented from

attending to future elements. Compared with recurrent networks, Transformers are capable of processing much longer sequences with parallel processing of all positions. The use of attention mechanisms and future masks (cross-attention in decoder) largely speed up the training process, thus allowing large-scale pre-training on vast amounts of data (see Section 2.4).

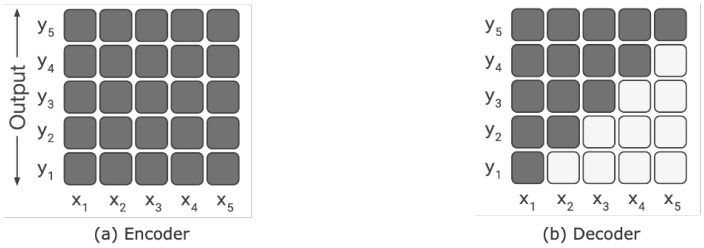


Fig. 2.13 The attention patterns in Transformer encoder and decoder blocks [173]

2.3 Training

Given a set of training data $\mathcal{D}_{\text{train}}$, network training determines the optimal parameters $\hat{\boldsymbol{\theta}}$ of a neural network $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$. The training objective is usually to minimise a loss function $\mathcal{L}(\boldsymbol{\theta})$ that is chosen to improve the model prediction $\hat{\boldsymbol{y}}$:

$$\hat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \tag{2.66}$$

where $\boldsymbol{\theta}$ is the set of parameter variables, and $\hat{\boldsymbol{\theta}}$ is the optimal estimate of the parameters. When $\boldsymbol{x}$ and $\boldsymbol{y}$ are clearly defined and examples of both are available during training, supervised learning is adopted to learn the exact mapping. In cases where only $\boldsymbol{x}$ is available, unsupervised learning is used to learn feature detectors without requiring the labelling of $\boldsymbol{y}$, and the learned feature detectors can be used to help reconstruct the input $\boldsymbol{x}$. When there is a small amount of labelled data (paired $\boldsymbol{x}$ and $\boldsymbol{y}$) with a large amount of unlabelled data (only $\boldsymbol{x}$) available, semi-supervised learning can be used to guide the training under weak supervision.

The focus of this thesis is on supervised and semi-supervised approaches. It is defined that there is an input $\boldsymbol{x}^{(j)}$ and a reference output $\boldsymbol{y}^{(j)}$ in the j^{th} training instance of $\mathcal{D}_{\text{train}}$, and a hypothesis output $\hat{\boldsymbol{y}}^{(j)}$ can be computed:

$$\mathcal{D}_{\text{train}} = \{\{\boldsymbol{x}^{(1)},\boldsymbol{y}^{(1)}\},\{\boldsymbol{x}^{(2)},\boldsymbol{y}^{(2)}\},...\{\boldsymbol{x}^{(J)},\boldsymbol{y}^{(J)}\}\} \tag{2.67}$$

$$\hat{\boldsymbol{y}}^{(j)} = f(\boldsymbol{x}^{(j)};\boldsymbol{\theta}) \tag{2.68}$$

This following sections will discuss the training criteria as well as the optimisation processes that are used to derive the optimal parameter set $\hat{\boldsymbol{\theta}}$.

2.3.1 Training criteria

Training criteria are defined using loss functions, the exact form of which depends on the nature of the tasks. For classification tasks, $\boldsymbol{y}$ is a one-hot class vector, and $\hat{\boldsymbol{y}}$ denotes the post softmax output that represents the probability of $\boldsymbol{x}$ belonging to each of the D classes. The objective is to maximise the probability of assigning the correct class label:

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \text{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p(\boldsymbol{x},\boldsymbol{y})}[\boldsymbol{y}^T \hat{\boldsymbol{y}}] \\ &= \text{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim p(\boldsymbol{x},\boldsymbol{y})}[-\log(\boldsymbol{y}^T \hat{\boldsymbol{y}})]\end{aligned} \tag{2.69}$$

Linking back to the parameter estimation defined in Equation 2.66, the loss function can therefore be approximated by the cross-entropy loss on the training set $\mathcal{D}_{\text{train}}$:

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}) &= -\tfrac{1}{J}\textstyle\sum_{j=1}^{J} \log(\boldsymbol{y}^{(j)T} \hat{\boldsymbol{y}}^{(j)}) \\ &= -\tfrac{1}{J}\textstyle\sum_{j=1}^{J}\sum_{d=1}^{D} y_d^{(j)} \log(\hat{y}_d^{(j)})\end{aligned} \tag{2.70}$$

For regression tasks, $\boldsymbol{y}$ is a continuous vector, and the objective is to minimise the distance between the ground truth $\boldsymbol{y}$ and the estimation $\hat{\boldsymbol{y}}$. A common mean squared error (MSE) loss can be adopted to minimise the L_2 distance between the two:

$$\mathcal{L}(\boldsymbol{\theta}) = \tfrac{1}{J}\textstyle\sum_{j=1}^{J} \|\hat{\boldsymbol{y}}^{(j)} - \boldsymbol{y}^{(j)}\|_2^2 \tag{2.71}$$

For sequence models discussed in Section 2.2, the objective is to minimise the Kullback-Leibler (KL) divergence [110] between the true and the estimated sequence distributions:

$$\hat{\boldsymbol{\theta}} = \text{argmax}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_{1:T},\boldsymbol{y}_{1:N}\sim p(\boldsymbol{x}_{1:T},\boldsymbol{y}_{1:N})} \text{KL}\{P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T}) || P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta})\} \tag{2.72}$$

where $\boldsymbol{x}_{1:T}$ and $\boldsymbol{y}_{1:N}$ denote the input and output sequences[1]. In practice, the loss function can be approximated by minimising the Negative Log-Likelihood (NLL) loss over the training instances sampled from the true distribution:

$$\mathcal{L}(\boldsymbol{\theta}) = -\tfrac{1}{J}\textstyle\sum_{j=1}^{J} \log P(\boldsymbol{y}_{1:N}^{(j)}|\boldsymbol{x}_{1:T}^{(j)};\boldsymbol{\theta}) \tag{2.73}$$

[1]For sequence tagging tasks, the input and output sequences are equal in length, i.e. $T = N$

For non-autoregressive tasks where the current prediction is independent of the previous predictions, the loss function can be simply expanded by summing over all tokens of the output sequence:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{J}\sum_{j=1}^{J}\sum_{n=1}^{N}\log P(\boldsymbol{y}_n^{(j)}|\boldsymbol{x}_{1:T}^{(j)};\boldsymbol{\theta}) \tag{2.74}$$

For autoregressive tasks, the sequence distribution $P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta})$ is factorised across time as described in Equation 2.56. The key problem lies in approximating the token distribution. A commonly adopted approach is teacher forcing [220], where the token distribution is computed with the reference output history $\boldsymbol{y}_{1:n-1}$. The loss function can be expanded as:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{J}\sum_{j=1}^{J}\sum_{n=1}^{N}\log P(\boldsymbol{y}_n^{(j)}|\boldsymbol{y}_{1:n-1}^{(j)},\boldsymbol{x}_{1:T}^{(j)};\boldsymbol{\theta}) \tag{2.75}$$

Teacher forcing allows efficient training since the token distributions over the entire output sequence can be computed in parallel with the use of reference output history. However, it suffers from exposure bias [175], which arises from the mismatch between training and inference. The model is guided with the reference output history during training, whereas the hypothesised output history must be used during inference. Such mismatch will lead to accumulated errors along the inference process. To further address exposure bias, scheduled sampling [11, 52], professor forcing [114], and attention forcing [49, 50] approaches were proposed. In this thesis, teacher forcing is adopted as the default training approach for sequence models to ensure efficient training.

Another line of approach is called Minimum Bayes Risk (MBR) training [60], which trains the model at the sequence-level, as opposed to token-level training described in Equation 2.75. Given a distance metric $D(\boldsymbol{y}_{1:N},\hat{\boldsymbol{y}}_{1:N})$ between the ground truth $\boldsymbol{y}_{1:N}$ and a hypothesised output $\hat{\boldsymbol{y}}_{1:N}$, MBR training directly minimises the distance:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{J}\sum_{j=1}^{J}\sum_{\hat{\boldsymbol{y}}_{1:T}^{(j)}\in\mathcal{Y}^{(j)}} P(\hat{\boldsymbol{y}}_n^{(j)}|\boldsymbol{x}_{1:T}^{(j)};\boldsymbol{\theta})D(\boldsymbol{y}_{1:N}^{(j)},\hat{\boldsymbol{y}}_{1:N}^{(j)}) \tag{2.76}$$

where $\mathcal{Y}^{(j)}$ denotes the entire search space of $\hat{\boldsymbol{y}}_{1:N}^{(j)}$. Compared with token-level training, sequence-level training is exempted from exposure bias by taking into account the hypothesised sequence as a whole during training. The distance metric D can be chosen to be any arbitrary form, whereas in token-level training the loss function needs to be differentiable. It is a common practice to choose the distance metric to be the same as the evaluation metric, e.g. Word Error Rate for speech recognition, BLEU [158] for machine translation. However, directly optimising the sequence loss is quite challenging, since the expectation over the entire $\mathcal{Y}^{(j)}$ is intractable. In practice, MBR training is often realised with a reduced search

space $\widetilde{\mathcal{Y}}^{(j)}$ [187] or using Monte Carlo approximation [223]. Larger search space requires longer training time, whereas smaller search space gives a rough approximation and makes the training less stable [175].

In general, as the number of training samples N increases, the estimated $\hat{\boldsymbol{\theta}}$ gets closer to the true global minimum and generalises better to examples that are not seen at training time. Model capacity often scales with the complexity of the network, i.e. number of layers, and dimension of hidden states. Complex networks are more capable of capturing high degrees of non-linearities, yet more likely to get stuck at local minimums due to limited training samples. This problem is commonly known as overfitting [25], where a neural network fits exactly against its training data, but fails to perform accurately against unseen data. Several regularisation techniques (discussed in Section 2.3.2) are used to improve generalisation by limiting model capacity or smoothing the loss function.

2.3.2 Optimisation

Neural network training is the process of solving Equation 2.66, yet there is usually no closed-form solution for the optimisation problem. Since the networks are deliberately designed to be differentiable, a gradient descent approach [177] is adopted together with error backpropagation [178, 76]. The optimisation is an iterative refinement process, and each iteration can be described using:

$$\boldsymbol{\theta}[\tau+1] = \boldsymbol{\theta}[\tau] - \Delta\boldsymbol{\theta}[\tau] = \boldsymbol{\theta}[\tau] - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}[\tau]} \tag{2.77}$$

where $\boldsymbol{\theta}[\tau]$ denotes the parameter estimation at iteration step τ and η is the learning rate. The iterative learning process ends when τ hits a pre-determined maximum number of iterations, or when an early stopping criterion (discussed below) is satisfied. The optimisation process searches for a global minimum on a high-dimensional loss surface, and the negative gradient follows the direction of the steepest descent at the current location.

The vanilla optimisation process operates on the loss function over the entire training set, and thus suffers from slow training as well as poor generalisation performance. A more commonly adopted training scheme in recent years is stochastic gradient descent (SGD) [20], which speeds up training and adds extra regularisation measure. It partitions the data into multiple batches followed by random shuffling, and updates the parameters based on one batch at a time. Gradients are calculated on a small batch of data each time, thus regularising the model through constraining its capacity. The name 'stochastic' arises from the noise introduced by estimating the gradient over the entire training set using a small batch.

A known issue with stochastic gradient descent is slow convergence. Sudden direction changes in gradients can be caused by switching batches, and consequently slow down the optimisation process. Momentum methods are introduced to stabilise gradients by updating parameters using a rolling sum of current and previous gradients. Commonly used momentum approaches includes Nesterov accelerated gradient (NAG) [151], Adaptive Gradient Algorithm (Adagrad) [51] and Adaptive Moment Estimation (Adam) [105]. Adam is adopted throughout this thesis, which uses both the first and second moment to decelerate sudden changes in gradient updates:

$$\Delta\boldsymbol{\theta}[\tau] = \eta \frac{\boldsymbol{m}[\tau]}{\sqrt{\boldsymbol{v}[\tau]+\varepsilon}} \tag{2.78}$$

$$\boldsymbol{m}[\tau] = \frac{1}{1-\beta_1^\tau}(\beta_1 \boldsymbol{m}[\tau-1] + (1-\beta_1)\boldsymbol{g}[\tau]) \tag{2.79}$$

$$\boldsymbol{v}[\tau] = \frac{1}{1-\beta_2^\tau}(\beta_2 \boldsymbol{v}[\tau-1] + (1-\beta_2)\boldsymbol{g}^2[\tau]) \tag{2.80}$$

$$\boldsymbol{g}[\tau] = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\boldsymbol{\theta}}|_{\boldsymbol{\theta}[\tau]} \tag{2.81}$$

where $\boldsymbol{g}[\tau]$ is the gradient at the current step τ; $\boldsymbol{m}[\tau]$ and $\boldsymbol{v}[\tau]$ are estimates of the first moment (the mean) and the second moment (the variance) of the gradients respectively; β_1 and β_2 are hyperparameters controlling the decay rates. Adam stores an exponentially decaying average of past squared gradients $\boldsymbol{v}[\tau]$ and keeps an exponentially decaying average of past gradients $\boldsymbol{m}[\tau]$. If momentum is seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface [79].

Each iteration of gradient descent methods updates the parameters based on their gradients, and backpropagation is used to compute the gradients in weight space, with respect to the loss function. The underlying principle of backpropagation is to apply the chain rule for derivatives: the derivative of the loss function can be expressed as a product of derivatives between each layer (working backwards), and the gradients of the weights between each layer are calculated using a modification of the partial products.

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{W}^{(k)}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{z}^{(k)}} \frac{\partial \boldsymbol{z}^{(k)}}{\partial \boldsymbol{W}^{(k)}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{z}^{(k)}} \boldsymbol{x}^{(k)T} \tag{2.82}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{z}^{(k)}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{z}^{(k+1)}} \frac{\partial \boldsymbol{z}^{(k+1)}}{\partial \boldsymbol{y}^{(k)}} \frac{\partial \boldsymbol{y}^{(k)}}{\partial \boldsymbol{z}^{(k)}} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{z}^{(k+1)}} \boldsymbol{W}^{(k+1)T} \frac{\partial \boldsymbol{y}^{(k)}}{\partial \boldsymbol{z}^{(k)}} \tag{2.83}$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{y}^{(k)} = \sigma(\boldsymbol{z}^{(k)}) = \sigma(\boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)}) \tag{2.84}$$

where $\theta = \{\boldsymbol{W}^{(1)}, ..., \boldsymbol{W}^{(K)}\}$ is the parameter set; $\boldsymbol{x}^{(k)}$, $\boldsymbol{y}^{(k)}$ are the input and output of the k^{th} layer; and $\boldsymbol{z}^{(k)}$ is the pre-activation layer output. Equation 2.83 shows the chain rule operating on the k^{th} layer, and Equation 2.82 calculates the derivative of the weight matrix $\boldsymbol{W}^{(k)}$. FFNs

are optimised by repeatedly applying backpropagation to propagate gradients through all modules, and RNNs can also be trained using the same approach by unfolding along their hidden layers, which is known as backpropagation through time [219, 109].

Once the gradients are calculated, the gradient descent step updates all layers simultaneously, assuming that the parameters stay unchanged relative to each other. Such assumption ignores the higher order effects, and would potentially lead to a rough loss surface. An ill conditioned loss surface tends to be particularly sensitive to its hyperparameters, and consequently makes the training process less stable or even fail to reach convergence. Such effects become even greater with deeper networks. The following sections will discuss various approaches to mitigate this issue by making the loss surface smoother.

Regularisation

Regularisation techniques are widely used to overcome the overfitting issue by improving generalisability. Limiting the size and number of layers is a straight-forward approach to restrict models from high degrees of non-linearities. Weight decay [147] is also effective in limiting model capacity, i.e. adding a penalty term to the loss function $\mathcal{L}(\boldsymbol{\theta})$ to limit the absolute magnitude of the model parameters:

$$\mathcal{L}'(\boldsymbol{\theta}) = \mathcal{L}(\boldsymbol{\theta}) + \alpha\|\boldsymbol{\theta}\|_2 \tag{2.85}$$

where α is a weight penalty coefficient. Another common approach is early stop [25], i.e. during training, a held-out validation set is used to evaluate the model performance, and training will be terminated if the validation loss stops decreasing for a few consecutive iterations. Dropout [196] restricts networks from achieving their full capacity during training by randomly dropping a certain percentage of the hidden units in the network. Batch and layer normalisation techniques (discussed in Section 2.1.4) both help smooth the loss function and consequently speed up convergence.

Parameter initialisation

At the beginning of the training, model parameters need to be assigned initial values. The optimisation process can be sensitive to the initial values, especially with complex networks and small training sets. Poor initialisation can lead to convergence to local minima or even gradient saturation [63]. Glorot initialisation [61] is introduced to mitigate this issue. Biases are initialised to be 0 and every element w_{ij} in the weight matrix $\boldsymbol{W}$ is initialised using a normal distribution:

$$w_{ij} \sim \mathcal{N}(-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}) \tag{2.86}$$

where N is the number of columns in $\boldsymbol{W}$. With the rise of Transformers and large-scale pre-training, the state-of-the-art initialisation uses large pre-trained models (see Section 2.4). Model parameters are initialised using the optimised values of another network with a similar architecture pre-trained on a related task. This allows efficient use of data from related tasks, and speeds up convergence on the target task by bootstrapping its parameters in a favourite state that is closer to its optimum.

Learning rate scheduling

Learning rate scheduling, also known as warm-up, is used to help adaptive optimisers (Adam for example) compute correct statistics of the gradients at the beginning of training. When the dataset is highly differentiated, the training might suffer from 'early over-fitting', i.e. if the training data batch happens to include a cluster of correlated observations, the initial training will potentially skew towards those features. Warm-up [64] is used to mitigate this issue, without which the training might require additional epochs to reach desirable convergence. It begins with a high learning rate to learn simple patterns, and then lower the learning rate at later stages to learn more complex patterns [128]. Equation 2.87 describes a commonly used scheduling strategy:

$$\eta[\tau] = \begin{cases} \eta_p \tau / \tau_p & 0 \leq \tau \leq \tau_p \\ \eta_p e^{-\alpha\tau} & \tau > \tau_p \end{cases} \tag{2.87}$$

where τ is the iteration step, α is a decay coefficient, η_p and τ_p are the peak learning rate and its corresponding iteration. The learning rate is set to increase linearly to a specific peak value in the first few epochs, followed by exponential weight decay to zero [121].

2.4 Large pre-trained models

In recent years, large-scale pre-training has achieved great successes on a wide range of natural language processing tasks. The initial efforts aim to learn word-to-vector (word2vec) mappings, such as Skip-gram [145], CBOW [143] and GloVe [161]. The learned models generate distributed and continuous representations of words, which are able to capture semantic meanings. However, the embeddings are context-free and thus cannot extract correlations and higher-level concepts. More advanced techniques aim to learn contextual word embeddings, such as CoVe [140] and ELMo [183]. The embedding mapping functions modelled using recurrent networks can only be trained on a limited amount of data, due to its long training time that scales with sequence lengths. With the emergence of the

Transformer architectures, more recent approaches are able to achieve more generalised language modelling purposes, as well as train on vast amounts of unlabelled data. The main advantages of pre-training are to extract high-quality contextual embeddings to help downstream training, and to provide a good initialisation point to speed up convergence.

This section gives a brief introduction of different variants of the recent Transformer-based pre-training models. The existing pre-trained models are categorised into three groups in terms of their architectures, namely Transformer encoder, decoder and encoder-decoder structures. The encoder and decoder both use self-attention to extract contextual information from the input, and the decoder adopts an additional causal mask to prevent tokens from attending to their future context. The encoder-decoder architecture follows the original Transformer formulation with cross-attention.

Transformer encoder

BERT [104], short for Bidirectional Encoder Representations from Transformers, is a bidirectional language model consisting of multiple Transformer encoder blocks. It is trained on free-text, and is able to predict both the left and right context. The pre-training tasks are:

- Masked language model (MLM): 15% of the tokens are randomly masked out in the input sequence, each replaced with either a mask token [MASK] or a random word. The model makes predictions on the masked words, without additional information about which of the words have been replaced.
- Next sentence prediction (NSP): Sentence pairs (A, B) are sampled so that there is a 50% chance that B follows A, and the model is trained to predict a binary label indicating whether B is the next sentence of A.

These training objectives encourage BERT to consider bidirectional context, and enhance sentence-level understanding. In addition to the word embeddings and positional embeddings that are proposed in the original Transformer, the input embeddings in BERT also include segment embeddings, allowing input sequences to contain two sentences. Another trick is to apply the WordPiece [185] tokenisation, which divides words into smaller sub-word units so that it can handle rare or unknown words. The Transformer architecture speeds up the training process, thus allowing BERT to be trained on over 3 billion words at the pre-training stage. Once pre-trained, BERT can be fine-tuned to downstream tasks by simply adding a classification head. Typical downstream applications include sentence classification (one label per sentence), sequence tagging (one label per token), as well as question answering tasks (predict the start and end of a span in a sentence).

RoBERTa [134] is a robustly optimised BERT approach, which uses an optimised training recipe to achieve better results. The recipe includes: train with a larger batch size for a longer

time; remove the NSP objective; concatenate multiple sentences into a longer segment input; dynamically change the mask pattern used for the masked language model objective. The hyperparameter fine-tuning also have a large impact on model performance.

ALBERT [117] is a lightweight version of BERT. Under the same configuration, ALBERT can be trained with 18 times fewer parameters and is about 1.7 times faster. It proposes two parameter efficient modifications: factorised embedding parameterisation decomposes the embedding matrix $V \times H$ into two matrices $V \times E$ and $E \times H$; cross-layer parameter sharing forces the same set of parameters to be shared across multiple layers. Both approaches make significant cut to the number of parameters without compromising much performance. Another tweak is to replace NSP with a sentence order prediction (SOP) objective, which requires a coherent understanding of the full segment.

XLNet [229] aims to address two critical issues in BERT: the MLM training assumes that the masked tokens are conditionally independent, and the special mask token '[mask]' never occurs at the fine-tuning stage. XLNet adopts a permutation language modelling (PLM) objective, which is a causal language model (CLM) over all possible permutations of words within a sentence. The PLM bypasses the independence assumption on masked tokens, and learns to capture the bidirectional context. To further account for the positional information, a two-stream self-attention is introduced with share parameters. In addition to the general self-attention for content extraction, it adds a second attention steam to encode the position to be predicted.

XLM [41] extends the monolingual language models to cross-lingual scenarios. Apart from the common MLM objective, it proposes a translation language model (TLM) objective which extends MLM to pairs of sentences in different languages. The TLM attends to both the source and target languages, and encourages alignment between the two.

XLM-R, short for XLM-RoBERTa, is an XLM [41] model trained with the RoBERTa [134] criteria. It adopts the translation language model (TLM) objective to learn the alignment between the source and target language pairs. The main changes from the original XLM are that XLM-R drops the language embedding to allow better code-switching, and it scales up to a much larger set of languages and more training data. XLM-R is pre-trained on over 2.5 TB of clean CommonCrawl corpus in 100 languages, and obtains state-of-the-art performance on various cross-lingual NLP tasks.

Transformer decoder

GPT [171], GPT-2 [172], GPT-3 [21] are a series of Generative Pre-training Transformers proposed by OpenAI. The architecture is an autoregressive language model which consists of multiple layers of Transformer decoders. One limitation of the GPT series is that they only account for the unidirectional left-to-right context. Compared with GPT, GPT-2 is 10

times larger and allows zero-shot transfer. By attaching task specific labels at the end of each input sequence, the model is able to infer task-specific conditional probabilities without fine-tuning. GPT-3 adopts the exact same architecture as GPT-2, but is 10 times larger than GPT-2. The parameters in the giant model are spliced along the width and depth dimensions, in order to fit onto GPUs for training. Under the few-shot setting, GPT-3 achieves comparable performance as the fine-tuned BERT models on benchmark NLP tasks.

Transformer encoder-decoder

BART [125], short for Bidirectional and AutoRegressive Transformer, jointly trains a BERT-like bidirectional encoder and a GPT-like autoregressive decoder [217]. It corrupts text by adding various noising transformations, and trains the model to de-noise the corrupted version and recover the original context. It is found that span-based masks and sentence shuffling are the most effective noising functions.

T5 [173], short for 'Text-to-Text Transfer Transformer', is an encoder-decoder style generative language model following the original Transformer architecture. T5 proposes to use a unified 'Natural Language Decathlon' framework [141] to convert all text-based tasks into a text-to-text question-answering format. The training process can be split into two stages. In the initial pre-training stage, T5 adopts the unsupervised de-noising objective following BERT, and the 750 gigabyte Colossal Clean Crawled Corpus (C4) corpus is used for training. In the supervised fine-tuning stage, task-specific prefixes are added to the input sequence to indicate the nature of the downstream tasks, and the model can then be separately fine-tuned to a wide range of NLP tasks, such as machine translation and document summarisation. This general purpose pre-training allows efficient transfer learning, and has achieved state-of-the-art results on many NLP benchmarks.

2.5 Summary

This chapter reviewed the fundamentals of neural networks, and covered a wide range of deep learning techniques that are further used in the following chapters to model a wide range of spoken language tasks. Section 2.1 reviewed the basic units of neural networks, which are then used as building blocks in Section 2.2 for sequence tagging and sequence-to-sequence models. Section 2.3 discussed the training criteria and the gradient descent based optimisation process. Finally, Section 2.4 discussed the state-of-the-art Transformer-based pre-training approaches that are used in future chapters to initialise individual modules of spoken language tasks.

Chapter 3

Individual Modules

Chapter 2 reviewed a wide range of deep learning techniques, including sequence models, training methods, as well as large-scale pre-training approaches. In Chapter 3, these techniques are applied to formulating individual modules of spoken language tasks.

Due to the lack of end-to-end data, spoken language tasks are often modelled as multimodular cascaded systems. This chapter first gives a multimodular cascaded formulation of spoken language tasks, and discusses the approximations being made for the cascaded structure. The rest of this chapter focuses on the modelling of individual modules in spoken language systems. Automatic speech recognition (ASR) converts speech into transcriptions, and is often considered as the upstream module of spoken language tasks. The generative hybrid structure and the discriminative end-to-end ASR models are both described. Connecting to the upstream ASR module, various text-based tasks can be used as the downstream modules. Disfluency detection (DD), neural machine translation (NMT) and grammatical error correction (GEC) are used in this thesis as the example downstream tasks. DD helps remove speech disfluencies, NMT translates one language into another, and GEC converts grammatically incorrect sentences into correct ones.

3.1 Multimodular systems

Spoken language processing can be broadly viewed as a pattern recognition problem. It takes a sequence of speech input $\boldsymbol{x}_{1:T}$ and generates a sequence of textual output $\boldsymbol{y}_{1:N}$. The maximum a posteriori (MAP) decision can be expressed as:

$$\hat{\boldsymbol{y}}_{1:N} = \text{argmax}_{\boldsymbol{y}_{1:N} \in \mathcal{Y}} P(\boldsymbol{y}_{1:N} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \tag{3.1}$$

For complicated objectives that do not have sufficient data to train the system end-to-end, a commonly adopted approach is to break down the complex task into multiple simpler modules, and use intermediate representations to pass information across modular connections. Equation 3.1 can therefore be expressed as a multiplication of multiple modules:

$$\begin{aligned} P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) &= \sum_{\boldsymbol{z}_{1:L}^{(1)}\in\mathcal{Z}^{(1)}} \dots \sum_{\boldsymbol{z}_{1:L}^{(M)}\in\mathcal{Z}^{(M)}} P(\boldsymbol{y}_{1:N},\boldsymbol{z}_{1:L}^{(1)},\boldsymbol{z}_{1:L}^{(2)},\dots,\boldsymbol{z}_{1:L}^{(M)}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \\ &= \sum_{\boldsymbol{z}_{1:L}^{(1)}\in\mathcal{Z}^{(1)}} \dots \sum_{\boldsymbol{z}_{1:L}^{(M)}\in\mathcal{Z}^{(M)}} P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L}^{(1)},\dots,\boldsymbol{z}_{1:L}^{(M)},\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \\ &\qquad \left(\prod_{m=2}^{M} P(\boldsymbol{z}_{1:L}^{(m)}|\boldsymbol{z}_{1:L}^{(1)},\dots,\boldsymbol{z}_{1:L}^{(m-1)},\boldsymbol{x}_{1:T};\boldsymbol{\theta})\right) P(\boldsymbol{z}_{1:L}^{(1)}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \end{aligned} \tag{3.2}$$

where there are a total of $(M+1)$ modules, $\boldsymbol{z}_{1:L}^{(1)}...\boldsymbol{z}_{1:L}^{(M)}$ denote the intermediate representations[1], and $\mathcal{Z}^{(1)}...\mathcal{Z}^{(M)}$ denote their respective search spaces[2]. To make the modules separable, it is assumed that the current module prediction is independent of all the intermediate variables from the upstream modules, except for the direct input of the current module. Therefore, Equation 3.2 can be simplified as:

$$\begin{aligned} P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx \sum_{\boldsymbol{z}_{1:L}^{(1)}\in\mathcal{Z}^{(1)}} \dots \sum_{\boldsymbol{z}_{1:L}^{(M)}\in\mathcal{Z}^{(M)}} & P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L}^{(M)};\boldsymbol{\theta}) \\ & \left(\prod_{m=2}^{M} P(\boldsymbol{z}_{1:L}^{(m)}|\boldsymbol{z}_{1:L}^{(m-1)};\boldsymbol{\theta})\right) P(\boldsymbol{z}_{1:L}^{(1)}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \end{aligned} \tag{3.3}$$

However, it is still computationally expensive to realise the full marginalisation in Equation 3.3. It can be further simplified so that the marginalisation over intermediate variables is approximated using a most likely hypothesis:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L}^{(M)};\boldsymbol{\theta}) \tag{3.4}$$

$$\boldsymbol{z}_{1:L}^{(m)} \approx \text{argmax}_{\boldsymbol{z}_{1:L}^{(m)}\in\mathcal{Z}^{(M)}}\{P(\boldsymbol{z}_{1:L}^{(m)}|\boldsymbol{z}_{1:L}^{(m-1)};\boldsymbol{\theta})\} \tag{3.5}$$

Such approximations lead to a cascaded formulation of spoken language processing tasks, where each individual module can be separately trained in their corresponding domains.

[1]Here L is used to denote the sequence lengths for all $\boldsymbol{z}_{1:L}^{(1)}...\boldsymbol{z}_{1:L}^{(M)}$ to simplify equations. In practice, L varies for different intermediate variables.

[2]Equation 3.2 assumes that the intermediate variables are discrete sequences, and therefore summations are used for marginalisation purposes. If they are continuous variables, integrations will be used instead.

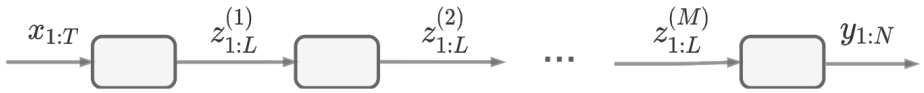


Fig. 3.1 Illustration of a multimodular cascaded system

Figure 3.1 shows an illustration of the multimodular cascaded system. Take spoken language translation (SLT) as an example, a cascaded SLT system can be modelled using an automatic speech recognition (ASR) module followed by a neural machine translation (NMT) module, and speech transcripts are used as the intermediate variable $\boldsymbol{z}_{1:L}$ connecting the two modules. The underlying assumption is that the intermediate variable encapsulates all the information from its upstream modules, and then passes them onto its downstream modules. This approximation gives rise to various issues in cascaded systems, and more detailed module combination approaches will be discussed in Chapter 4. The focus of this chapter is to review the modelling of individual modules in cascaded spoken language systems, and the sequence modelling approaches discussed in Section 2.2 will be applied in the following sections to formulate the speech and text processing modules.

For a general spoken language processing task, the initial stage is usually to convert speech signals into transcriptions, and thus an automatic speech recognition (ASR) module is needed. Following the upstream ASR, there are usually one or more downstream textual processing modules, and the downstream tasks covered in this thesis are listed below:

- Disfluency detection (DD) removes speech disfluencies so that speech transcriptions can be made more text-like. It is commonly used as a post-processing procedure for speech transcripts, which recovers a fluent text flow for downstream processing.
- Neural machine translation (NMT) translates sentences into a foreign language. When operating on speech transcriptions, it can be seen as the downstream module of a spoken language translation (SLT) system.
- Grammatical error correction (GEC) corrects grammatical errors in a sentence. When used to correct grammar errors in speech transcriptions, it can be seen as part of a spoken grammatical error correction (SGEC) system.

3.2 Automatic speech recognition

Automatic speech recognition (ASR) is a sequence-to-sequence task that converts input speech into its corresponding transcriptions. It aims to maximise the posterior probability of

a word sequence $w_{1:L}$ given the observation utterance $\boldsymbol{x}_{1:T}$[3].

$$\hat{w}_{1:L} = \text{argmax}_{w_{1:L} \in \mathcal{W}} P(w_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_{\text{ASR}}) \tag{3.6}$$

The initial breakthrough in speech recognition is the use of generative hybrid modelling [8, 170, 150], where speech is decomposed at the different acoustic levels with separate models. More recent work has been transiting to discriminative end-to-end (E2E) modelling [67, 168] where speech sequences are directly translated into transcripts using a single network. Hybrid models tend to be more robust when faced with a limited amount of training data, whereas E2E models allow potential tighter integration with downstream tasks of the ASR module. This section gives a brief overview of the hybrid and E2E models, which will be used in the following chapters as part of the spoken language processing tasks.

3.2.1 Hybrid models

In a traditional hybrid model, the posterior can be rewritten using a joint distribution:

$$P(w_{1:L} | \boldsymbol{x}_{1:T}) = \frac{P(w_{1:L}, \boldsymbol{x}_{1:T})}{\sum_{w'_{1:L} \in \mathcal{W}} P(w'_{1:L}, \boldsymbol{x}_{1:T})} \tag{3.7}$$

where the denominator can be dropped since it is independent of $w_{1:L}$. The hybrid model can therefore be viewed as a generative model, which optimises for the joint distribution $P(w_{1:L}, \boldsymbol{x}_{1:T})$. The joint distribution can be further expanded using Bayes Law:

$$P(w_{1:L}, \boldsymbol{x}_{1:T}) = P(\boldsymbol{x}_{1:T} | w_{1:L}) P(w_{1:L}) \tag{3.8}$$

where $P(\boldsymbol{x}_{1:T} | w_{1:L})$ accounts for the acoustic model (AM) that approximates the observation distribution given a word sequence, and $P(w_{1:L})$ is the prior distribution of the word sequence determined by a language model (LM).

Acoustic model (AM)

Since modelling alignment between the observation and word sequences can be difficult on different length scales, word sequences are often decomposed into state sequences $s_{1:T}$ that have the same length T as the observation sequences:

$$P(\boldsymbol{x}_{1:T} | w_{1:L}) = \sum_{s_{1:T} \in \mathcal{S}} P(\boldsymbol{x}_{1:T}, s_{1:T} | w_{1:L}) \tag{3.9}$$

[3] $\boldsymbol{\theta}_{\text{ASR}}$ is omitted for the rest of this section.

where $\mathcal{S}$ is the collection of the state sequences. The posterior can be further expanded into:

$$P(\boldsymbol{x}_{1:T}, s_{1:T}|w_{1:L}) = P(s_{1:T}|w_{1:L})P(\boldsymbol{x}_{1:T}|s_{1:T}, w_{1:L}) \tag{3.10}$$

Hidden Markov Model (HMM) is adopted to compute the joint likelihood of the observation and the state sequences [58, 57], and two assumptions are made in HMM in order to simplify the probability estimation. The first one assumes that the current state is conditionally independent of all other states, given the previous state:

$$P(s_{1:T}|w_{1:L}) = \prod_{t=1}^{T} P(s_t|s_{t-1}, w_{1:L}) \tag{3.11}$$

The second assumption is that the current observation is conditionally independent of all other observations, words and states, given the current state:

$$P(\boldsymbol{x}_{1:T}|s_{1:T}, w_{1:L}) = \prod_{t=1}^{T} P(\boldsymbol{x}_t|s_t) \tag{3.12}$$

The posterior distribution can therefore be approximated as:

$$P(\boldsymbol{x}_{1:T}|w_{1:L}) = \sum_{s_{1:T}\in\mathcal{S}} \prod_{t=1}^{T} P(\boldsymbol{x}_t|s_t)P(s_t|s_{t-1}) \tag{3.13}$$

where $P(\boldsymbol{x}_t|s_t)$ is the emission probability and $P(s_t|s_{t-1})$ is the state transition probability. In practice, the states are often defined by dividing a word into sub-word units (phones), then converting those sub-word units into states.

Language model (LM)

$P(w_{1:L})$ denotes the prior distribution of a word sequence $w_{1:L}$ jointly occurring, which can be decomposed into:

$$P(w_{1:L}) = P(w_1) \prod_{l=2}^{L} P(w_l|w_{1:l-1}) \tag{3.14}$$

The dimension of the distribution grows exponentially with the sequence length L. To make the distribution tractable, conditional independence assumptions are made. N-gram approximation [7] is commonly used, which assumes that the probability of the current word is conditionally independent of all other words, given the previous $(N-1)$ words:

$$P(w_{1:L}) \approx P(w_1) \prod_{l=2}^{L} P(w_l|w_{l-N+1:l-1}) \tag{3.15}$$

The quality of the estimation increases with N, yet it also makes the system less statistically reliable due to the limited amount of training data. If an N-gram phrase does not exist in the training corpus, its probability estimation will be zero, which prevents the phrase from being

generated in any occasion. Smoothing, discounting [103] and back-off [106] methods are used to improve the statistical reliability of the N-gram language model.

Neural language models has shown to be more powerful in generalising to unseen corpora [14]. A commonly adopted approach is recurrent neural network language model (RNNLM) [144], which models the probability distribution of the current word w_l as:

$$P(w_l|w_{1:l-1}) = P(w_l|w_{l-1}, \boldsymbol{h}_{l-2}; \boldsymbol{\theta}) \tag{3.16}$$

where the current word prediction is conditioned on the previous word w_{l-1} and the back history representation $\boldsymbol{h}_{l-2}$. The previous word sequence $w_{1:l-2}$ is represented using a hidden state $\boldsymbol{h}_{l-2}$, which is modelled using a recurrent network:

$$\boldsymbol{h}_{l-2} = f(\boldsymbol{h}_{l-3}, w_{l-2}; \boldsymbol{\theta}_{\text{rnn}}) \tag{3.17}$$

The network is trained to maximise the probability of observing a word given its history (or in some cases future) context. The main advantages of neural based models are that they are able to extend to a much larger context window, and can be easily generalised when encountered with unseen words.

Recognition

The posterior distribution combining the acoustic and language models can be expressed as:

$$P(w_{1:L}|\boldsymbol{x}_{1:T}) \propto P(w_{1:L})^{\gamma} \sum_{s_{1:T} \in \mathcal{S}} \prod_{t=1}^{T} P(\boldsymbol{x}_t|s_t)^{\alpha} P(s_t|s_{t-1})^{\alpha} \tag{3.18}$$

where γ, α are the language model and acoustic model scaling factors. The language and acoustic models are separately trained with dramatically different dynamic ranges, and the scaling factors are used to reach a balanced range between the two. At the decoding stage, hypotheses can be represented using lattices [156], which are directed acyclic graphs composed of vertices and arcs. Each arc is associated with a word hypothesis, an acoustic model score and a language model score. Each vertex corresponds to a time stamp that connects an arbitrary number of incoming and outgoing arcs.

To reduce computational cost, speech recognition is often performed as such: lattices are generated with a first pass decoding using a weak language model such as a bigram or trigram LM; in the second pass, the lattice re-scoring is performed using a more advanced language model such as a higher order N-gram model or RNNLM. The re-scored lattices can be decoded using Viterbi [205] to obtain an N-best list of the recognition hypotheses.

3.2.2 End-to-end models

Generative hybrid ASR systems require separate training of an acoustic model and a language model, both of which can be complex. In contrast, end-to-end (E2E) ASR[4] refers a group of discriminative approaches which directly optimise for the posterior $P(w_{1:L}|\boldsymbol{x}_{1:T})$. This allows a much simpler training pipeline, reduces both the training and decoding time, and also enables joint optimisation with its downstream processing. With abundant training data, E2E ASR models have been shown to outperform traditional hybrid models both in academia [213] and in industry [180, 127]. However, current E2E ASR systems need orders of magnitude more training data than hybrid ASR systems to achieve similar performance, since E2E models tend to overfit to the training corpus when it is limited [112].

There are three main branches of E2E ASR models [126]: (a) Connectionist Temporal Classification (CTC) [66], (b) Attention-based Encoder-Decoder (AED) [34], and (c) Recurrent neural network Transducer (RNN-T) [65]. Figure 3.2 illustrates the three model architectures, where the encoder and decoder networks usually adopt RNN or Transformer blocks discussed in Section 2.1.

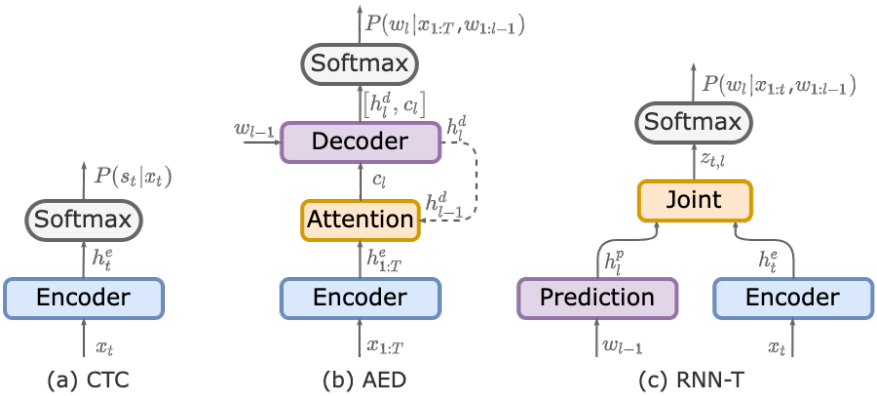


Fig. 3.2 Architectures of three popular E2E ASR models [75].

The CTC-based ASR maps the input speech into an output label sequence, with blank labels in between neighbouring output labels to construct a CTC path that has the same length as the input. The CTC model is the first E2E approach widely used in ASR [72, 242], yet most CTC-based works require external language models due to the lack of implicit language modelling [67, 137]. The RNN-T model is most commonly used for streaming ASR [168, 180], since its current prediction is conditioned on the previously predicted words and the input speech until the current time step. In this thesis, AED-based ASR is

[4] 'End-to-end' is a commonly used name in the speech community, which indicates the use of a single network for ASR modelling. The underlying commonality of this line of approaches is that they all adopt the discriminative objective to directly optimise for the posterior distribution.

adopted. It uses an attention mechanism to align the speech and transcription sequences. In spoken language tasks, the alignment information is beneficial for tighter integration with the downstream text processing modules (see Chapter 4).

Attention-based Encoder-Decoder (AED)
AED-based ASR (see Figure 3.2b) uses an attention mechanism to directly calculate the alignment between the input speech and the output labels. The underlying structure is an attention-based sequence-to-sequence model (discussed in Section 2.2.2): the encoder converts the input speech into a sequence of hidden representations; the decoder compresses the encoded sequence into a fixed length vector through a weighted sum generated from the attention mechanism, and then makes a word prediction accordingly. The training objective is to maximise the probability of the output sequence:

$$\mathcal{L}_{\text{AED}} = -\log P(w_{1:L}|\boldsymbol{x}_{1:T}) = -\log \prod_{l=1}^{L} P(w_l|\boldsymbol{x}_{1:T}, w_{1:l-1}) \tag{3.19}$$

With the current word prediction being conditioned on all previously generated words, an implicit language model is built in to AED models. However, AED models usually suffer from latency issues, since the attention is to be applied to the entire encoded sequence, which forces it to wait until the encoding process to be completed before proceeding forward. A popular approach to building low-latency streaming AED systems is to apply attention on chunks of the input speech [32].

3.3 Disfluency detection

Disfluency is a commonly observed linguistic phenomenon in spontaneous spoken language. A typical disfluency structure consists of reparandum, interregnum and repair regions [191]:

$$\text{I want a train } [\overbrace{\textit{to Oxford}}^{\text{reparandum}} \overbrace{\textit{uh I mean}}^{\text{interregnum}} + \overbrace{\text{to London}}^{\text{repair}}]$$

where reparandum refers to repetitions and restarts, and interregnum includes filled pauses and discourse markers. The underlying fluent sentence can be recovered by removing the associated words in the reparandum and interregnum regions. Interregnum regions often consist of fixed phrases of filled pauses and discourse markers (‘um’, ‘you know’ etc.) that are easy to detect using rule-based methods [99]. Automatic disfluency detection therefore focuses on reparandum detection.

Disfluency detection (DD) models can be generally divided into two categories: parsing-based models and sequence tagging models. Parsing-based methods [28, 241] identify

disfluency structures by learning the syntactic structure of spoken sentences. They can jointly perform parsing and detection, yet the training requires a large amount of annotated tree-banks. Alternatively, sequence labelling methods assign fluent and disfluent tags to each word [169, 234]. In this work, DD is modelled as a sequence tagging task, where each input word w_l is assigned a binary disfluency tag d_l (illustrated in Figure 3.3):

$$\hat{d}_{1:L} = \text{argmax}_{d_{1:L}} P(d_{1:L}|w_{1:L}; \boldsymbol{\theta}_{\text{DD}}) \tag{3.20}$$

The general framework of sequence tagging models are discussed in Section 2.2.1. The model posterior can be expanded depending on the nature of the sequence tagging process. In the scope of this thesis, sequence tagging is always modelled as a non-autoregressive process where the current prediction d_l is conditionally independent of the prediction history:

$$P(d_{1:L}|w_{1:L}; \boldsymbol{\theta}_{\text{DD}}) = \prod_{l=1}^{L} P(d_l|d_{1:l-1}, w_{1:L}; \boldsymbol{\theta}_{\text{DD}}) \approx \prod_{l=1}^{L} P(d_l|w_{1:L}; \boldsymbol{\theta}_{\text{DD}}) \tag{3.21}$$

Initial works on tagging models [234] adopt hand-crafted features as inputs e.g. words, part-of-speech tags and N-gram based patterns. Under the context of spoken language processing, the DD model in this thesis serves as a downstream module of an ASR system, thus taking speech transcriptions as inputs. Features such as part-of-speech are not readily available for speech transcripts, neither do punctuations and capitalisations. Therefore, a sequence of unpunctuated and uncapitalised word tokens are used as the only input to the DD module.

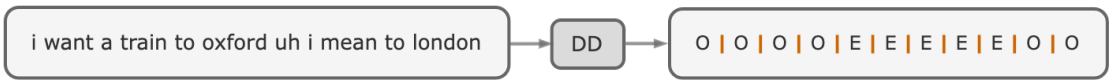


Fig. 3.3 An example of disfluency tagging: ‘O’ and ‘E’ denote fluent and disfluent tags.

Sequence tagging models first extract contextual features from the input sequence, and the extracted features are fed to a classification layer. Disfluency detection is a simple binary classification problem with ‘E’ and ‘O’ respectively denoting disfluent and fluent tags. The classification head therefore adopts a simple feed-forward layer followed with a sigmoid function. The tagging performance largely depends on the quality of the feature representations. The following sections will introduce two realisations of the non-autoregressive disfluency tagger with different feature extractors: a simple RNN-based approach, and a more advanced large pre-trained Transformer-based architecture.

3.3.1 Recurrent network

An RNN-based DD model (Figure 3.4) first converts words $w_{1:L}$ into continuous embedding vectors $\boldsymbol{e}_{1:L}$, and then uses a bidirectional long short term memory (BLSTM) layer (see Section 2.1.2) for contextual feature extraction:

$$\boldsymbol{e}_l = \boldsymbol{W}^e w_l \tag{3.22}$$

$$\overrightarrow{\boldsymbol{h}}_l = \text{LSTM}(\boldsymbol{e}_l, \overrightarrow{\boldsymbol{h}}_{l-1}; \overrightarrow{\boldsymbol{\theta}}) \quad \overleftarrow{\boldsymbol{h}}_l = \text{LSTM}(\boldsymbol{e}_l, \overleftarrow{\boldsymbol{h}}_{l+1}; \overleftarrow{\boldsymbol{\theta}}) \tag{3.23}$$

$$\boldsymbol{h}_l = [\overrightarrow{\boldsymbol{h}}_l, \overleftarrow{\boldsymbol{h}}_l] \tag{3.24}$$

where $\boldsymbol{e}_l$ is the embedding of the word w_l, and $\boldsymbol{W}^e$ is the trainable embedding matrix. $\overrightarrow{\boldsymbol{h}}_l, \overleftarrow{\boldsymbol{h}}_l$ denote the left and right context representations, and $\overrightarrow{\boldsymbol{\theta}}, \overleftarrow{\boldsymbol{\theta}}$ are the LSTM parameters. The concatenation of the two context vectors becomes the contextual representation $\boldsymbol{h}_l$, which is then fed through a feed-forward classification layer:

$$P(d_l|\boldsymbol{h}_l, \boldsymbol{\theta}_{\text{cls}}) = \sigma_{\text{sigmoid}}(\boldsymbol{W}^f \sigma_{\tanh}(\boldsymbol{W}^h \boldsymbol{h}_l + \boldsymbol{b}^h)) \tag{3.25}$$

where the parameter set $\boldsymbol{\theta}_{\text{cls}}$ contains $\{\boldsymbol{W}^f, \boldsymbol{W}^h, \boldsymbol{b}^h\}$. Given the ground truth labels, a standard negative log-likelihood loss can be used for training.

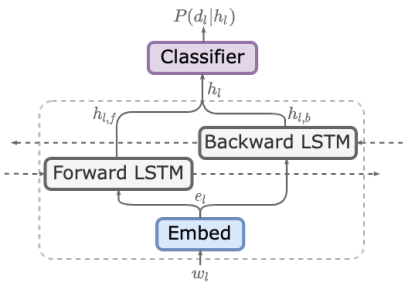


Fig. 3.4 Illustration of an RNN-based DD model. The dotted rectangle outlines the contextual feature extraction process.

The BLSTM generates reasonably good quality feature representations, since it is able to extract contextual information from both sides of the current word. However, the training time of the vanilla RNN-based tagger scales with sequence lengths, which prohibits it from large-scale training on large corpora.

3.3.2 Large pre-trained model: BERT

BERT [104] is a Transformer encoder based large language model trained on a giant collection of free text corpora (described in Section 2.4). It can be further fine-tuned on specific tasks without additional customisation of network architectures.

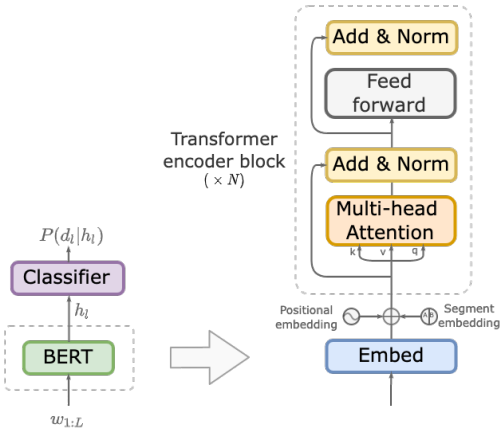


Fig. 3.5 Illustration of a BERT-based DD model. The DD model overview is on the left, and the BERT internal architecture is shown on the right.

Figure 3.5 shows an illustration of the BERT-based DD model. Compared with the RNN-based DD, the feature extractor is replaced by BERT, and the classifier layer stays the same. Equation 3.22-3.24 are simply replaced using:

$$\boldsymbol{h}_{1:L} = \text{BERT}(w_{1:L}) \tag{3.26}$$

where $\boldsymbol{h}_{1:L}$ denotes the contextual feature. With the Transformer architecture, BERT is a strong language model pre-trained on a large amount of data. It is therefore capable of extracting high quality feature representations. The BERT-based DD model is initialised with the pre-trained BERT, and further fine-tuned using the disfluency detection objective.

3.4 Neural machine translation

Machine translation translates source text in one language to target text in another language. Early works on machine translation mainly uses hand-crafted translation rules [107]. The ideas of statistical machine translation (SMT) were first introduced in the 1940s [90], where translations are generated based on statistical models. With the rise of deep neural networks, neural-based statistical models are gaining more interest. Neural machine translation (NMT)

is a neural-based approach of SMT, which can be viewed as an end-to-end trainable sequence-to-sequence model with variable length inputs and outputs.

Fig. 3.6 An example of English to German (En-De) translation

An NMT model conducts a sequence-to-sequence mapping from an input sequence $w_{1:L}$ to an output sequence $y_{1:N}$ (illustrated in Figure 3.6):

$$\hat{y}_{1:N} = \text{argmax}_{y_{1:N} \in \mathcal{Y}} P(y_{1:N} | w_{1:L}; \boldsymbol{\theta}_{\text{NMT}}) \tag{3.27}$$

Initial works on NMT make use of attention mechanisms together with RNNs for sequence modelling, and achieve comparable performance to the existing phrase-based systems [6, 222]. Further advances are made with the introduction of the Transformer-based translation models [203]. Another line of work explores edit-based approaches [48, 139], which targets at translation problems that only require small changes to the input, such as post editing, style transfer and grammatical error correction. This thesis mainly adopts the attention-based approach, since it allows translation between two distanced language pairs.

This section discusses various realisations of attention-based sequence-to-sequence models (see Section 2.2.2) under the context of machine translation: an RNN-based NMT and a Transformer-based NMT models are discussed, following which a large-scale pre-trained Transformer-based T5 model is also described.

3.4.1 Recurrent network

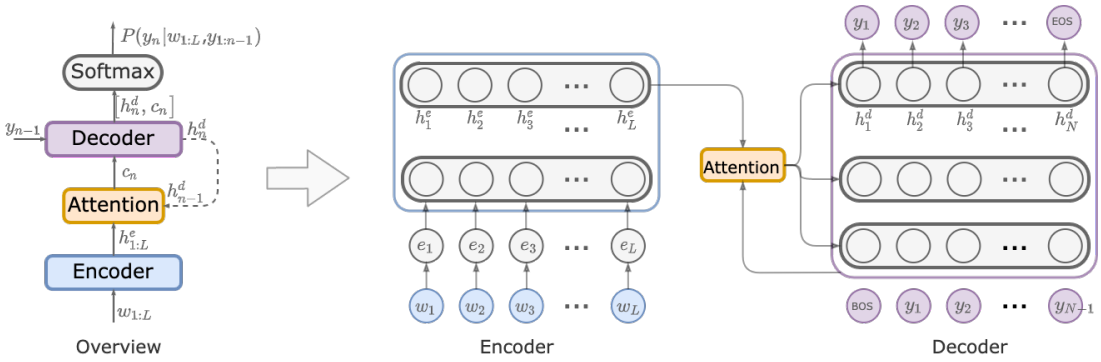


Fig. 3.7 Illustration of an RNN-based NMT model [222]. The model overview is shown on the left, and the right part shows a more detailed architecture.

Under the general framework of the attention-based encoder decoder model, an RNN-based NMT model consists of an encoder, an attention mechanism and a decoder. Figure 3.7 shows an illustration of the model architecture. Each word token w_l in the input $w_{1:L}$ is mapped into an embedding $\boldsymbol{e}_l$ via a multiplication with a learnable embedding matrix $\boldsymbol{W}_e$:

$$\boldsymbol{e}_l = \boldsymbol{W}_e w_l \tag{3.28}$$

The rest of the model follows the attention-based sequence-to-sequence described in Section 2.2.2. The encoder maps the embeddings $\boldsymbol{e}_{1:L}$ into context vectors $\boldsymbol{h}^e_{1:L}$. The attention mechanism generates attention weights $\boldsymbol{\alpha}_n$ over the input context $\boldsymbol{h}^e_{1:L}$ according to the previous decoder state $\boldsymbol{h}^d_{n-1}$ (some implementations use hidden vectors from lower layers of the decoder instead). Based on the context vector $\boldsymbol{c}_n$ and the previous predictions $y_{1:n-1}$, the decoder generates a new state $\boldsymbol{h}^d_n$, and makes the prediction for y_n.

$$\boldsymbol{h}^e_{1:L} = f_{\text{enc}}(\boldsymbol{e}_{1:L}; \boldsymbol{\theta}_{\text{enc}}) \tag{3.29}$$

$$\boldsymbol{\alpha}_n = f_{\text{att}}(\boldsymbol{h}^d_{n-1}; \boldsymbol{h}^e_{1:L}, \boldsymbol{\theta}_{\text{att}}) \quad \boldsymbol{c}_n = \textstyle\sum_{l=1}^{L} \alpha_{n,l} \boldsymbol{h}^e_l \tag{3.30}$$

$$\boldsymbol{h}^d_n = f_{\text{dec}}(\boldsymbol{h}^d_{n-1}, y_{n-1}, \boldsymbol{c}_n; \boldsymbol{\theta}_{\text{dec}}) \tag{3.31}$$

$$\hat{y}_n \sim P(y_n | \boldsymbol{h}^d_n; \boldsymbol{c}_n, \boldsymbol{\theta}_y) \tag{3.32}$$

The encoder and decoder functions f_{enc} and f_{dec} are composed of multiple RNN layers. The attention weights can be interpreted as the alignment between the inputs and outputs, which allows the decoder to attend to the most relevant segments to the next token prediction. The first token fed to the decoder is a special 'beginning of sentence' (BOS) token, and the sequence generation terminates when a special 'end of sentence' (EOS) token is predicted by the decoder. At the training stage, the reference back history of the output sequence is used in the decoder, whereas at the inference stage, the hypothesised back history is used.

RNNs are designed to handle sequential data, and their recurrent nature enables an infinite receptive field [192] both for the encoder context vectors and for the decoder states. However, they suffer from the vanishing gradient problem [16], and approaches like LSTM and GRU (see Section 2.1.2) are used to mitigate this issue. In addition, residual [73] and highway [197] connections are widely used to speed up convergence. Another known issue of RNN-based models is the long training time, which prohibits the sequence-to-sequence model from being trained on a large amount of data.

3.4.2 Transformer

The Transformer-based NMT also consists of an encoder, an attention mechanism and a decoder conceptually. The three components serve similar purposes as those in RNN-based NMT, yet the network architecture is designed differently. Figure 3.8 illustrates a Transformer-based NMT model (Transformer blocks are discussed in Section 2.1.4).

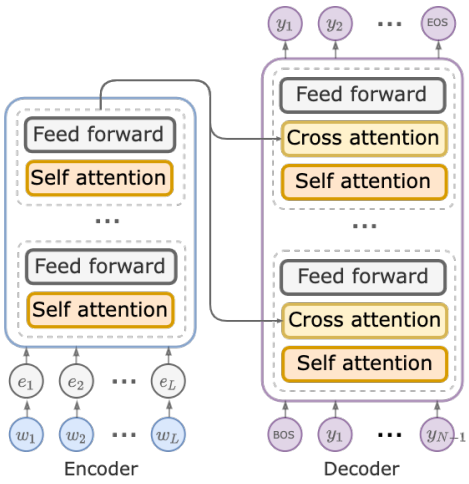


Fig. 3.8 Illustration of a Transformer-based NMT model [203]

The Transformer encoder is composed of multiple encoder blocks, each containing a multi-head self-attention layer and a feed-forward layer. With the self-attention layer replacing the recurrent units, the encoder is able to convert input sequences into feature vectors in constant time regardless of the sequence length. Different from RNN-based models, the attention mechanism and the conceptual decoder are merged into the Transformer decoder. The Transformer decoder has multiple decoder blocks, each containing a multi-head self-attention layer, a multi-head cross-attention layer and a feed-forward layer. The cross-attention layer replaces the encoder-decoder attention in RNN-based models, which aligns the encoded feature vectors and the decoder states. The self-attention layer serves the same purpose as the RNN decoder, which generates decoder states from the previous predictions.

At training time, the reference back history is used in the decoder, and an additional triangular mask is applied to the decoder self-attention layer to prevent it from attending to future words. The use of self-attention allows parallel processing of the input and output sequences, and therefore the training time is independent of the lengths of input and output sequences. At inference time, the hypothesised back history is used, and thus the sequence generation time scales linearly with the output sequence length. Under sufficient training data, the Transformer-based NMT usually out-performs the RNN-based NMT. The transformer

structure allows parallel training, reduces training time on large datasets, and enables a larger receptive field with the self-attention layers.

3.4.3 Large pre-trained model: T5

T5 [173] is an encoder-decoder style Transformer model which adopts a unified framework that formulates all text-based tasks with a question-answering format (described in Section 2.4). To adapt the original T5 model to the domain of interest, further fine-tuning can be easily conducted on supervised corpus that is pre-processed into a question answering format. Figure 3.9 shows an example of the fine-tuning pipeline. In the case of machine translation, a task specific prefix 'translate A into B:' is added to the source sequence, where 'A' and 'B' are replaced with the source and target languages respectively. In general, fine-tuned T5 models out-perform conventional Transformers trained from scratch, and the main advantage comes from the good initialisation point yielded from the large scale pre-training.

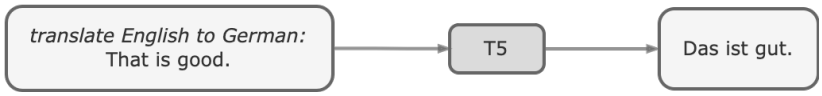


Fig. 3.9 An example pipeline of T5 fine-tuning of the En-De task [173]

3.5 Grammatical error correction

The problem of automatic assessment of second language has been wildly studied in computer-assisted language learning (CALL). Among others, grammatical construction is one of the key aspects of language assessment, and grammatical error correction (GEC) has attracted considerable interest over the past few years [22, 153, 44].

Fig. 3.10 An example of the GEC process

A GEC model converts a grammatically incorrect input sentence $w_{1:L}$ into a grammatically correct output sequence $y_{1:N}$. An example of the GEC process is shown in Figure 3.10, and the prediction process can be formulated as:

$$\hat{y}_{1:N} = \text{argmax}_{y_{1:N} \in \mathcal{Y}} P(y_{1:N} | w_{1:L}; \boldsymbol{\theta}_{\text{GEC}}) \tag{3.33}$$

Inspired by machine translation, GEC is often viewed as a translation task. Phrase-based statistical machine translation (SMT) [236, 101], and more recent neural machine translation (NMT) models [233, 184] have both achieved high performance in GEC. An alternative line of work adopts an edit-based approach, which emphasises the characteristics of the GEC task by directly editing certain parts of the sentence [35, 157]. In this thesis, the translation-based approaches are adopted, since it allows more complex transformations from the source to target sequences. This section discusses the translation-based GEC approaches realised using RNNs and Transformers.

3.5.1 Recurrent network and Transformer

The RNN-based and Transformer-based GEC models both follow the general formulation of an attention-based encoder decoder structure, which are similar to the NMT models introduced in Section 3.4.1 and 3.4.2 respectively. The main difference is that in GEC tasks, the grammatically incorrect source sequence may contain spelling mistakes, which significantly increases the number of out-of-vocabulary (OOV) words. One remedy to the high OOV rate is to switch to character level inputs, and it allows attention mechanism to directly access each character [224]. In the scope of this thesis, the inputs to GEC models are always speech transcriptions, which have a fixed set of vocabulary. Although speech recognition might introduce extra transcription errors, it avoids any OOV words, and thus the conventional token level inputs can be used.

3.5.2 Large pre-trained model: Gramformer

Gramformer [45] is a Transformer encoder-decoder model, initialised with T5 pre-training and further fine-tuned on GEC corpora. The original T5 model is not trained with the GEC task, therefore an additional task prefix ‘gec: ’ is assigned, and the fine-tuning pipeline is shown in Figure 3.11.

Fig. 3.11 An illustration of Gramformer training: fine-tuning T5 on GEC corpora

There is a limited amount of manual GEC annotation, and thus several GEC corpora with synthetically generated errors [129] are adopted for Gramformer training: WikiEd [69] extracts edits from Wikipedia revisions to form correction pairs; C4 [198] adopts tagged corruption models to generate a desired type of errors by attaching an error tag to a clean

sentence; PIE [2] adopts BERT to introduce synthetic errors through local sequence editing. Benefited from large-scale pre-training, The Gramformer outperforms Transformer models trained from scratch. It inherits a strong language model from T5, and further transfers the knowledge into the GEC domain. With additional manually annotated correction pairs, the Gramformer model can be further fine-tuned to the domain of interest.

3.6 Summary

This chapter applied the sequence modelling techniques introduced in Chapter 2, and formulated individual modules of spoken language tasks. Section 3.1 motivated the cascaded formulation of multimodular spoken language systems, and the rest of this chapter discussed in detail how each individual module can be modelled.

The first step in spoken language tasks is to convert speech into transcripts, and Section 3.2 reviewed both the generative hybrid ASR and the discriminative end-to-end ASR systems. Taking the speech transcripts as the inputs, several downstream processes were described: Section 3.3 discussed the Disfluency Detection (DD) task, and laid the focus on the sequence-tagging style approach; Section 3.4 and 3.5 respectively introduced the Neural Machine Translation (NMT) and Grammatical Error Correction (GEC) tasks, both of which are modelled using the sequence-to-sequence style approach in this thesis. For each of the downstream tasks, both the basic recurrent network based models, and the more advanced Transformer-based pre-trained models were discussed.

Chapter 4

Module Combination Approaches

One of the key challenges in building spoken language applications is the lack of end-to-end data. As discussed in Chapter 3, the multimodular cascaded structure is commonly adopted so that individual modules can be separately trained with sufficient amounts data from the associated domains. However, the multimodular formulation gives rise to issues like information loss, error propagation and domain mismatches. This chapter first discusses the challenges facing multimodular spoken language systems, and gives an overview of various module combination techniques. Three categories of spoken language systems are considered with an increasing level of module integration: cascaded, integrated and end-to-end systems.

Cascaded systems are discussed in Section 4.2. Individual modules in a cascaded system are separately trained and loosely coupled at inference time, which leads to domain mismatch and error propagation issues. Domain adaptation, supervised and semi-supervised error mitigation approaches are described to tackle these issues. Section 4.3 investigates integrated systems. They are composed of more tightly integrated modules, and require a small amount of end-to-end data to adapt to the target domain. Discrete information passing and embedding passing approaches are introduced to propagate richer information across the modular connection via discrete and continuous sequences. Section 4.4 discusses end-to-end systems. They do not rely on interpretable intermediate variables to aid training, and therefore require a large amount of end-to-end training data to reach convergence. Data augmentation and meta-learning approaches are discussed to overcome the low data efficiency issue.

4.1 Challenges in multimodular systems

As discussed in Section 3.1, spoken language processing tasks can be formulated as:

$$\boldsymbol{y}_{1:N} \sim P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \tag{4.1}$$

where $\boldsymbol{x}_{1:T}$ is the input speech sequence, and $\boldsymbol{y}_{1:N}$ is the output text sequence. Due to the lack of end-to-end training corpus, the task is often broken down into multiple simpler modules[1]:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) = \sum_{\boldsymbol{z}_{1:L}\in\mathcal{Z}} P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L},\boldsymbol{x}_{1:T};\boldsymbol{\theta}_y)P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_z) \tag{4.2}$$

$$\approx \sum_{\boldsymbol{z}_{1:L}\in\mathcal{Z}} P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L};\boldsymbol{\theta}_y)P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_z) \tag{4.3}$$

$$\approx P(\boldsymbol{y}_{1:N}|\hat{\boldsymbol{z}}_{1:L};\boldsymbol{\theta}_y) \quad \hat{\boldsymbol{z}}_{1:L} = \text{argmax}_{\boldsymbol{z}_{1:L}\in\mathcal{Z}} P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_z) \tag{4.4}$$

where $\boldsymbol{z}_{1:L}$ denotes the intermediate variable, and $\mathcal{Z}$ denotes its corresponding search space. Equation 4.4 shows the cascaded formulation of the multimodular system, which is derived with two important approximations. The first approximation (from Equation 4.2 to 4.3) assumes that given the intermediate variable $\boldsymbol{z}_{1:L}$, the downstream process is independent of the input signals $\boldsymbol{x}_{1:T}$. $\boldsymbol{z}_{1:L}$ can be viewed as the modular connection between the upstream and the downstream modules. The second approximation (from Equation 4.3 to 4.4) collapses the marginalisation over the intermediate variable with the most likely hypothesis. $\hat{\boldsymbol{z}}_{1:L}$ is used as the input of the subsequent translation module, instead of traversing through the entire search space $\mathcal{Z}$. For example, a cascaded spoken language translation (SLT) system consists of an upstream automatic speech recognition (ASR) module and a downstream neural machine translation (NMT) module, with the modular connection $\boldsymbol{z}_{1:L}$ being speech transcripts[2]. Modules in the cascaded system are separable, so that they can be individually trained with their corresponding corpora. However, due to the approximations being made in the derivations and the separately trained modules, the cascaded systems are faced with several challenges.

- Loss of information: As approximated in 4.3, given the intermediate variable $\boldsymbol{z}_{1:L}$, it is assumed that the downstream module is independent of the input $\boldsymbol{x}_{1:T}$, and therefore the output $\boldsymbol{y}_{1:N}$ is generated with $\boldsymbol{z}_{1:L}$ as the only input. The conditional independence only

[1] To simply the equations, it is assumed that there is only one intermediate variable $\boldsymbol{z}_{1:L}$ in this system. The two modules are parameterised with $\boldsymbol{\theta}_z$ and $\boldsymbol{\theta}_y$ respectively. The discussions in this chapter can be easily generalised to cases with more than two modules. The full multimodular formulation is in Section 3.1.

[2] The SLT example will be used across this chapter to help put the abstract ideas into context. Similar ideas can be easily generalised to other spoken language processing tasks.

holds when $\boldsymbol{z}_{1:L}$ encapsulates all the information from its upstream modules, which is not always the case. The loss of information issue arises when $\boldsymbol{z}_{1:L}$ fails to deliver the full context of the input $\boldsymbol{x}_{1:T}$. For example in a cascaded SLT system, prosodic information in speech signals cannot be conveyed via transcriptions. Prosody provides emphasis to certain words or parts of the speech, and conveys the speaker's attitude and emotional state while speaking [190]. Loss of prosodic information would potentially give rise to ambiguity in the downstream translation module.

- Error propagation: As approximated in 4.4, when computing the output distribution of $\boldsymbol{y}_{1:N}$, the marginalisation over $\mathcal{Z}$ is approximated using the most likely hypothesis $\hat{\boldsymbol{z}}_{1:L}$. This forces early decisions to be made at the upstream module, and any error in $\hat{\boldsymbol{z}}_{1:L}$ would propagate through to the downstream task. The downstream module is not trained on erroneous inputs, and therefore errors in $\hat{\boldsymbol{z}}_{1:L}$ tends to degrade the module performance. This is known as the error propagation problem. In the cascaded SLT example, error propagation often refers to the erroneous transcriptions $\hat{\boldsymbol{z}}_{1:L}$ causing disruptions to the translation process.

- Domain mismatch[3]: Cascaded systems allow individual modules to be separately trained in their corresponding domains. However, the training domains are often different from the evaluation domain, which will potentially lead to performance degradation. In the cascaded SLT example, the ASR module is trained on spoken corpora, whereas the NMT module is often trained on written text. Written and spoken languages usually have very different formats and styles: spontaneous spoken language shows a large set of disfluencies such as repetitions and false starts, which is not accounted for by the NMT module during training. Another example is the spoken grammatical error correction task (SGEC, will be further discussed in Chapter 7). SGEC usually consists of three modules, namely ASR, disfluency detection and error correction. Due to limited data availability, there is no training data that allows the three modules to be trained under the same target domain. Another important aspect is that individually trained modules usually have different dynamic ranges. Therefore, module posteriors cannot be superimposed with simple product, and additional calibration or weighting is required when linking modules together.

With the introduction of the end-to-end trainable encoder-decoder models (reviewed in Section 2.2.2), direct end-to-end (E2E) approaches are gaining more attention [214, 95].

[3]In this thesis, domain can be considered as a slightly broader concept compared with dataset. Different training datasets can belong to the same domain, e.g. the spoken language translation domain can contain paired datasets for both ASR and NMT model training. Therefore, in the following section, a dataset is defined using both domain information and input / output variables.

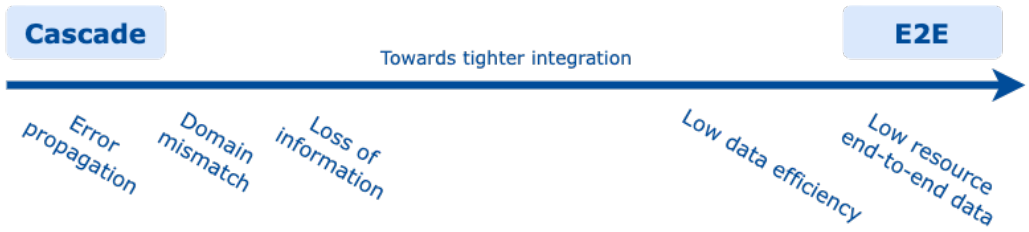


Fig. 4.1 The spectrum of module integration

Figure 4.1 shows a spectrum of module integration. The end-to-end style systems sit on the most tightly integrated end, opposite to the loosely coupled cascaded systems. End-to-end systems directly model the output probability of $\boldsymbol{y}_{1:N}$ given the input $\boldsymbol{x}_{1:T}$, and therefore avoid the aforementioned issues caused by module decomposition. They require stronger modelling power to model complex structured data in a single network, and with great capability comes the issue of low data efficiency.

- Low data efficiency: The strong modelling power of end-to-end systems are often achieved through delicately designed networks. As discussed in Section 2.1, complicated networks usually lead to a rough cost function surface, which slows down convergence and reduces data efficiency. It is shown that performance gains in end-to-end SLT systems is often achieved through incorporating additional training data [162, 97].

- Limited end-to-end data: There is usually an abundance of data pairs to train individual modules, yet end-to-end corpora of spoken language tasks is only available in small quantities due to increased cost for more complicated annotation processes. Restricted by both low data efficiency and limited training resource, it is very challenging to build high quality end-to-end systems.

This section has so far analysed the two ends of the module integration spectrum, cascaded and end-to-end, and discussed various challenges facing spoken language systems. There is a potential trade-off between modelling power and data efficiency: cascaded systems can be efficiently trained with loosely coupled modules, yet assumptions made for module decomposition will lead to issues like information loss, error propagation and domain mismatch; end-to-end systems with strong modelling power are exempt from module decomposition issues, yet low data efficiency and limited end-to-end data prevent direct systems from further improvements.

The rest of this chapter will discuss integration approaches to tackle the aforementioned issues, aiming to strike a balance between modelling power and data efficiency. The discussion here assumes that the multimodular system has two modules with the modular

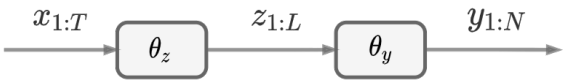


Fig. 4.2 Illustration of a two-module cascaded system. The upstream and downstream modules are parameterised with $\boldsymbol{\theta}_z$ and $\boldsymbol{\theta}_y$.

connection being $\boldsymbol{z}_{1:L}$ (see Figure 4.2), and similar ideas can be extended to systems with more than two modules. To assist future discussions, the training data notation is defined as:

$$\mathcal{D}_{\text{domain}}(\boldsymbol{a}, \boldsymbol{b}) \tag{4.5}$$

where $\boldsymbol{a}, \boldsymbol{b}$ are the input and output variables. This chapter considers a total of three data domains: $\mathcal{D}_{\text{up}}$ and $\mathcal{D}_{\text{dn}}$ respectively denote the upstream and downstream domains, and $\mathcal{D}_{\text{tgt}}$ denotes the target domain. For example in the SLT task, $\mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z})$ denotes the speech $\boldsymbol{x}$ and transcription $\boldsymbol{z}$ pair from the upstream ASR domain, and $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z})$ denotes the same speech to transcription pair but from the target SLT data domain[4]. In cases where end-to-end data is needed for training, $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$ denotes the speech, transcription and translation triplet from the target SLT domain.

The following discussion on spoken language systems is split into three overarching categories: loosely coupled cascade, tightly integrated, and direct end-to-end systems. Loosely coupled cascaded systems do not require end-to-end data, and individual modules can be separately trained with $\mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z})$ and $\mathcal{D}_{\text{dn}}(\boldsymbol{z}, \boldsymbol{y})$. Integrated systems can be initialised with data pairs corresponding to individual modules $\mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z})$ and $\mathcal{D}_{\text{dn}}(\boldsymbol{z}, \boldsymbol{y})$, and requires further fine-tuning with a small amount of end-to-end data $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{y})$. Direct end-to-end systems can only be trained on end-to-end data $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{y})$. Sometimes the intermediate variable $\boldsymbol{z}$ is also included in the target domain to form a data triplet $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{y})$.

4.2 Cascaded systems

Loosely coupled cascaded systems combine individual modules via direct concatenation, and they can be formulated as:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \approx P(\boldsymbol{y}_{1:N}|\hat{\boldsymbol{z}}_{1:L}; \boldsymbol{\theta}_y) \quad \hat{\boldsymbol{z}}_{1:L} = \text{argmax}_{\boldsymbol{z}_{1:L} \in \mathcal{Z}} P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z) \tag{4.6}$$

where $\boldsymbol{x}_{1:T}$, $\boldsymbol{y}_{1:N}$ denote the input and output sequences. $\hat{\boldsymbol{z}}_{1:L}$ denotes the most likely hypotheses of the modular connection $\boldsymbol{z}_{1:L}$, which is a vector sequence of discrete variables following

[4] Alternatively, under the context of an SLT task, $\mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z})$ can be expressed as $\mathcal{D}_{\text{ASR}}(\boldsymbol{x}, \boldsymbol{z})$, and $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z})$ can be expressed as $\mathcal{D}_{\text{SLT}}(\boldsymbol{x}, \boldsymbol{z})$.

the distribution parameterised with $\boldsymbol{\theta}_z$. Figure 4.3 shows an illustration of the training and evaluation processes of the cascaded system.

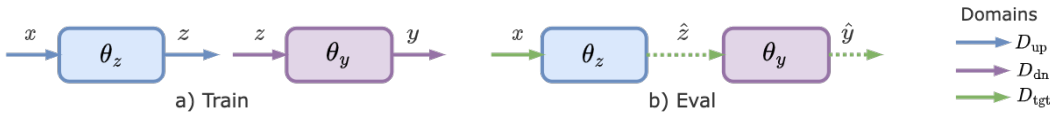


Fig. 4.3 Illustration of the training and evaluation processes of a two-module cascaded system. Dotted lines indicate hypotheses, and data domains are colour coded: blue indicates $\mathcal{D}_{\text{up}}$, purple indicates $\mathcal{D}_{\text{dn}}$, and green indicates the target domain for evaluation $\mathcal{D}_{\text{tgt}}$.

The cascaded system does not require any end-to-end annotation for training. Individual modules can be separately trained following the individual module formulations (discussed in Section 3) with their respective training corpora[5]:

$$\mathcal{L}(\boldsymbol{\theta}_z) = -\frac{1}{J}\sum_{j=1}^{J} \log P(\boldsymbol{z}_{1:L}^{(j)} | \boldsymbol{x}_{1:T}^{(j)}; \boldsymbol{\theta}_z) \quad \{\boldsymbol{x}_{1:T}^{(j)}, \boldsymbol{z}_{1:L}^{(j)}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z}) \tag{4.7}$$

$$\mathcal{L}(\boldsymbol{\theta}_y) = -\frac{1}{J}\sum_{j=1}^{J} \log P(\boldsymbol{y}_{1:N}^{(j)} | \boldsymbol{z}_{1:L}^{(j)}; \boldsymbol{\theta}_y) \quad \{\boldsymbol{z}_{1:L}^{(j)}, \boldsymbol{y}_{1:N}^{(j)}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{z}, \boldsymbol{y}) \tag{4.8}$$

where $\{\boldsymbol{x}_{1:T}^{(j)}, \boldsymbol{z}_{1:L}^{(j)}\}$ and $\{\boldsymbol{z}_{1:L}^{(j)}, \boldsymbol{y}_{1:N}^{(j)}\}$ indicate the j^{th} training pairs for the upstream and downstream modules. As discussed in Section 4.1, vanilla cascade suffers from issues arising from assumptions made for module decomposition. The following sections will introduce approaches to improve cascaded systems without altering the nature of each module.

4.2.1 Domain adaptation

Cascaded systems with modules trained in their individual domains usually perform poorly at the inference stage due to domain mismatches. The domain mismatch issues are two-fold: the mismatch between the upstream and downstream domains, and the mismatch between the training and inference domains. Domain adaptation (DA) [176, 10] is a commonly adopted approach to transfer the model trained from the source domain to the target domain. Assuming there is a small amount of training data available in the target domain for both the upstream and downstream modules, the domain adaptation training can be conducted as:

$$\mathcal{L}(\boldsymbol{\theta}_z^{\text{DA}}) = -\log P(\boldsymbol{z}_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z^{\text{DA}}) \quad \{\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z}) \tag{4.9}$$

$$\mathcal{L}(\boldsymbol{\theta}_y^{\text{DA}}) = -\log P(\boldsymbol{y}_{1:N} | \boldsymbol{z}_{1:L}; \boldsymbol{\theta}_y^{\text{DA}}) \quad \{\boldsymbol{z}_{1:L}, \boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{z}, \boldsymbol{y}) \tag{4.10}$$

[5]This section adopts the Negative Log-Likelihood (NLL) loss with teacher forcing training (discussed in Section 2.3.2) as the default objective function. To simplify the notation, the data index and the average over the training set $\frac{1}{J}\sum_{j=1}^{J}$ will be omitted in the following sections.

Figure 4.4 illustrates the domain adaptation. Compared with Equation 4.7 and 4.8, the upstream domain is transferred from $\mathcal{D}_{\text{up}}$ to $\mathcal{D}_{\text{tgt}}$, and the downstream domain is transferred from $\mathcal{D}_{\text{dn}}$ to $\mathcal{D}_{\text{tgt}}$. Training individual modules under the same target domain helps alleviate the domain mismatch between modules, and also closes the gap towards the target domain.

Fig. 4.4 Illustration of the domain adaptation training. The green colour indicates that training is conducted under the target domain $\mathcal{D}_{\text{tgt}}$.

4.2.2 Error mitigation

Cascaded systems also suffer from error propagation. Erroneous outputs from the upstream module tend to cause disruptions to downstream tasks. To mitigate error propagation, a straight forward approach is to adapt the downstream module to the upstream output: the upstream module is fixed, and the downstream module is trained with the hypothesised modular connection $\hat{\boldsymbol{z}}$ (illustrated in Figure 4.5.a):

$$\hat{\boldsymbol{z}}_{1:L} = \text{argmax}_{\boldsymbol{z}_{1:L} \in \mathcal{Z}} P(\boldsymbol{z}_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z) \tag{4.11}$$

$$\mathcal{L}(\boldsymbol{\theta}_y^{\text{EM}}) = -\log P(\boldsymbol{y}_{1:N} | \hat{\boldsymbol{z}}_{1:L}; \boldsymbol{\theta}_y^{\text{EM}}) \quad \{\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{y}) \tag{4.12}$$

where $\hat{\boldsymbol{z}}_{1:L}$ is the most likely hypothesis from the upstream module. Tuning with the hypothesis $\hat{\boldsymbol{z}}_{1:L}$ allows the downstream module to account for the errors in the input during training, and therefore helps mitigate error propagation during inference. Ideally, the training pair $\{\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:N}\}$ is sampled from $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{y})$ if the target domain data is available.

However, the fully annotated target domain corpus is not always readily available. In cases where the target domain data is only partially annotated, the supervised fine-tuning approach described above is no longer feasible. For spoken language tasks, it is quite common that the target domain corpus only contains annotations for the upstream module $\{\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z})$, for example in the spoken language translation task, it is relatively easy to obtain manual transcripts for the upstream ASR module, compared with the reference translations for the downstream NMT module. In this case, a semi-supervised error mitigation approach can be adopted instead: the pseudo references for the output $\boldsymbol{y}_{1:N}$ can be obtained by feeding the manual annotation for the modular connection $\boldsymbol{z}_{1:L}$ through the downstream

module (Figure 4.5.b):

$$\hat{\boldsymbol{z}}_{1:L} = \text{argmax}_{\boldsymbol{z}_{1:L}} P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z \in \mathcal{Z}) \tag{4.13}$$

$$\boldsymbol{y}^p_{1:N} = \text{argmax}_{\boldsymbol{y}_{1:N} \in \mathcal{Y}} P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L}; \boldsymbol{\theta}_y) \tag{4.14}$$

$$\mathcal{L}(\boldsymbol{\theta}_y^{\text{EMsemi}}) = -\log P(\boldsymbol{y}^p_{1:N}|\hat{\boldsymbol{z}}_{1:L}; \boldsymbol{\theta}_y^{\text{EMsemi}}) \quad \{\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{z}) \tag{4.15}$$

where $\boldsymbol{y}^p_{1:N}$ denotes the pseudo reference, and $\boldsymbol{\theta}_y^{\text{EMsemi}}$ denotes the parameters updated with semi-supervised training. The main difference between the semi-supervised and supervised error mitigation is in the loss function, where the pseudo reference $\boldsymbol{y}^p_{1:N}$ is used in the semi-supervised training as opposed to the ground truth reference $\boldsymbol{y}_{1:N}$ used in Equation 4.12.

Fig. 4.5 Illustration of the supervised and semi-supervised error mitigation training. The upstream module in grey is fixed, and the downstream module in green is updated.

Semi-supervised fine-tuning enables the downstream module to account for erroneous inputs from the target domain, yet its effectiveness is largely constrained by the quality of the pseudo references. To further mitigate the potential degradation caused by the poor quality pseudo references, self-distillation can be applied. Self-distillation originated from knowledge distillation [81], which often trains a student model to learn from predictions made by a teacher model. The teacher is usually superior to the student, i.e. the teacher is a more complex model than the student, or an ensemble teacher for a single model student. Self-distillation [237] is originally proposed in the computer vision community. It extends the idea of knowledge distillation by proposing to use the same model for both the teacher and the student. Self-distillation has proved effective in improving both image and text based tasks [226]. To conduct self-distillation under the semi-supervised fashion, the teacher model is fixed as $\boldsymbol{\theta}_y^{\text{EMsemi}}$ from Equation 4.15, and the student model $\boldsymbol{\theta}_y^{\text{EMdistil}}$ is a self-distilled version of the teacher. The training objective is to minimise the KL divergence of the per word posterior distribution between the teacher and the student:

$$\mathcal{L}_{\text{KL}}(\boldsymbol{\theta}_y^{\text{EMdistil}}) = \sum_{n=2}^{N} \text{KL}\{P(\boldsymbol{y}^p_n|\boldsymbol{y}^p_{1:n-1}, \hat{\boldsymbol{z}}_{1:L}; \boldsymbol{\theta}_y^{\text{EMsemi}}) || P(\boldsymbol{y}^p_n|\boldsymbol{y}^p_{1:n-1}, \hat{\boldsymbol{z}}_{1:L}; \boldsymbol{\theta}_y^{\text{EMdistil}})\} \tag{4.16}$$

where $\boldsymbol{y}^p_n$ denotes the n^{th} token in the pseudo reference sequence $\boldsymbol{y}^p_{1:N}$, and the loss function is summed over the output sequence length N. In addition to the empirical successes of

self-distillation, the intuition behind adopting self-distillation on semi-supervised data are two folded: it provides regularisation under a small amount of training data; and also guides the student model with richer probability distributions rather than relying solely on the one-hot pseudo references.

An alternative line of work aims to obtain higher quality hypotheses for the modular connection, which mitigates error propagation by passing fewer error to the downstream module. For example in the cascaded SLT system, using transcripts with fewer ASR errors will cause less disruption to the translation module, and consequently help improve the translation quality. Lattice rescoring [225] and N-best list[6] rescoring with shallow language model fusion [36] are commonly adopted approaches for sequence-to-sequence tasks. The idea is to over generate hypotheses at the decoding stage, and then adopt a stronger language model to rescore and choose the best candidate. For a cascaded system with N-best rescoring on the intermediate variable $\boldsymbol{z}_{1:L}$, Equation 4.6 can be re-written as:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(\boldsymbol{y}_{1:N}|\hat{\boldsymbol{z}}_{1:L};\boldsymbol{\theta}_y) \tag{4.17}$$

$$\hat{\boldsymbol{z}}_{1:L} = \text{argmax}_{\boldsymbol{z}_{1:L}\in\mathcal{Z}}\{P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:L},\boldsymbol{\theta}_z)^{\alpha}P(\boldsymbol{z}_{1:L}|\boldsymbol{\theta}_{\text{LM}})^{\beta}\} \quad |\mathcal{Z}| = M \tag{4.18}$$

where $\mathcal{Z}$ denotes the top M candidates, and $\hat{\boldsymbol{z}}_{1:L}$ denotes the 1-best candidate chosen from the rescoring process. $\boldsymbol{\theta}_{\text{LM}}$ refers to the language model, and α,β are the weight coefficients to fuse the posterior distribution with the language model. The language model training (discussed in Section 3.2.1) only requires monolingual data from the domain of interest, e.g. $\boldsymbol{z}_{1:L} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{z})$ in the SLT example. An extension to this approach is to propagate lattices or N-best candidates across the modular connection, as opposed to a single hypothesis. This requires additional end-to-end data for training, and will be further discussed in Section 4.3.1.

4.3 Integrated systems

This section considers more tightly integrated systems, which pass richer information flows across modular connections. Integrated systems preserve the modular concept, but with softer connections between neighbouring modules to allow richer information propagation. Compared with cascaded systems where individual modules are separately trained in their corresponding domains, integrated systems require a small amount of end-to-end corpora from the target domain for training. Figure 4.6 shows an illustration of the integrated system.

[6] N-best list is a conventionally used notion referring to the top N candidates from sequence generation tasks. N not to be confused with the sequence length in $\boldsymbol{y}_{1:N}$.

Fig. 4.6 Illustration of the initialisation and joint training processes of an integrated two-module system. The initialisation process can operate on modular data, whereas the in-domain training requires end-to-end data from the target domain.

An integrated system is usually trained in two stages: at the initialisation stage, individual modules are trained with module specific data; at the in-domain training stage, end-to-end corpora are used to adapt the system to the target domain. The initialisation stage allows individual modules to make use of module specific corpora, which often come in large quantities. With a better initialisation, the system can be more efficiently adapted to the target domain under limited end-to-end data[7]. The following sections will discuss two forms of the integrated systems: discrete information passing propagates information via discrete sequences; embedding passing propagates information across modules in continuous forms.

4.3.1 Discrete information passing

As the name suggests, discrete information passing uses discrete variables to pass information across modules. The main differences from the vanilla cascaded system are that: the modular connection is not restricted to a single sequence, and the modules require further in-domain training with end-to-end corpora[8].

4.3.1.1 Lattice / N-best list passing

In order to propagate richer information and mitigate error propagation, a natural extension from cascaded systems is to pass multiple hypotheses of $\boldsymbol{z}_{1:L}$ to the downstream module, as opposed to the 1-best prediction. The integrated system can therefore be formulated as:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(\boldsymbol{y}_{1:N}|\mathcal{Z};\boldsymbol{\theta}_y^{\text{DIP}}) \tag{4.19}$$

$$\hat{\boldsymbol{z}}_{1:L}^{(i)} \sim P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_z) \quad \mathcal{Z} = \{\hat{\boldsymbol{z}}_{1:L}^{(1)},...,\hat{\boldsymbol{z}}_{1:L}^{(M)}\} \tag{4.20}$$

[7] It is also feasible to skip the initialisation stage in cases where there is abundant end-to-end data.

[8] The vanilla cascaded system can be seen as a special case of the integrated system with discrete information passing, when a) the discrete modular connection is chosen to be a single sequence, and b) the modules are not further trained with end-to-end data.

where $\hat{\boldsymbol{z}}_{1:L}^{(i)}$ denotes the i^{th} best hypothesis from the upstream module, and $\mathcal{Z}$ denotes the M-best list, or a lattice, which also represents the top M hypotheses but in a more compact structure. The collected discrete sequences $\mathcal{Z}$ carries richer information from the upstream module compared with the 1-best estimation, which also helps alleviate the error propagation issue in the cascaded formulation. For example when applying this approach to the speech translation task, $\mathcal{Z}$ denotes the M-best ASR hypotheses. $\mathcal{Z}$ contains more information about the input speech $\boldsymbol{x}_{1:T}$ compared with the 1-best transcripts, and the richer context will potentially benefit the downstream translation process.

At the initialisation stage, the modules can be individually initialised in their corresponding domains following Equation 4.7 and 4.8. At the in-domain training stage, the downstream module needs to be partially modified to account for the inputs in the form of M-best lists or lattices, and training requires end-to-end data from the target domain[9]:

$$\mathcal{L}(\boldsymbol{\theta}_y^{\text{DIP}}) = -\log P(\boldsymbol{y}_{1:N}|\mathcal{Z};\boldsymbol{\theta}_y^{\text{DIP}}) \quad \{\boldsymbol{x}_{1:T},\boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x},\boldsymbol{y}) \tag{4.21}$$

The downstream processing can be divided into two steps conceptually: the first step extracts similarities and differences among the candidates $\mathcal{Z}$, and embeds discrete sequences into a single sequence of continuous feature representations; the second step simply performs a standard sequence-to-sequence task. Different approaches are adopted for the feature extraction step: Zhang et el. [238] proposes to use a lattice Transformer, which applies an attention mechanism over lattices to obtain latent representations; Liu et el. [133] fuses the n-best list using convolutional neural networks, and passes fused features into the downstream sequence-tagging module; Leng et el. [123] first extracts the alignment between the N-best candidates, and then feeds a concatenation of the aligned sequences into the downstream sequence-to-sequence module.

Propagating richer information in discrete forms allows the downstream module to account for a larger hypothesis space. It exploits the voting effect that cross-verifies token correctness among multiple candidates, and thus recovers from early decision errors to some extent. The modules in the system with discrete information passing can not be simultaneously optimised, since the discrete modular connection prohibits gradient backpropagation. A potential remedy is to adopt gradient estimators such as straight-through estimator [15] or Gumbel softmax [96] to approximate the gradient and work around the discrete bottleneck. Another line of work with embedding passing (see Section 4.3.2) allows joint optimisation through continuous modular connections. Error propagation can be mitigated with an increasing level of complexity involved in the modular connection $\mathcal{Z}$. However, maintaining a rich

[9] If the reference $\boldsymbol{z}_{1:L}$ is available to form a data triplet $\mathcal{D}_{\text{tgt}}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{y})$, the upstream module can also be adapted to the target domain following Equation 4.9

search space is computationally expensive, and with a limited search space, the information loss remains an issue. In the speech translation example, discrete transcription sequences cannot pass prosodic information, which will potentially incur ambiguity in the downstream translation.

4.3.1.2 Reranking

An alternative discrete information passing approach is to adopt an external reranker overlooking the cascaded system, which directly accesses the hypotheses of the intermediate variable $\boldsymbol{z}_{1:L}$. Rerankers are widely used in sequence generation tasks, such as speech recognition [132, 225], and translation tasks [39, 154, 122]. Figure 4.7 shows an illustration of a reranking process. Given a set of hypotheses from a sequence generation task, a reranker assigns new scores to each of them, and selects the best candidate among all. The underlying idea is that maximum likelihood training for sequence generation tasks does not always lead to posteriors that align with task specific metrics, such as BLEU scores. A reranker can therefore be used to rescore the candidates, ideally according to the metric of interest. Serving as a post-processing step, rerankers have access to all the N-best hypotheses. This allows rerankers to obtain a more accurate approximation of the posterior probabilities, and mitigates exposure bias by directly training on the model generated sequences.

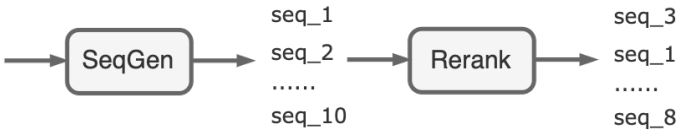


Fig. 4.7 Illustration of a reranking process operating on the 10-best hypotheses of a sequence generation model. The sequences are initially ordered with decreasing posterior probabilities from top to bottom, and then reordered using the reranking scores.

There are mainly two lines of work on rerankers: generative and discriminative reranking. Generative methods [135, 91] rescore hypotheses with strong language models, rather than directly predicting the rank ordering. Lattice rescoring with language model fusion is a commonly adopted approach in speech recognition [132]. Noisy-channel decoding [231, 232] scores hypotheses by incorporating three language models: a forward model that scores the hypothesis given the source, a backward model that scores the source conditioned on the hypothesis, and a target-side contextual language model. More recent works adopt a masked language model for rescoring [181]. Discriminative approaches directly predict the rank ordering according to the metrics of interest. Minimum Bayes Risk decoding [111] can be seen as a non-trainable discriminative reranking process, which ranks candidates with

sequence-level cost functions. Trainable discriminative rerankers can be considered as an alternative to sequence-level training (discussed in Section 2.3.1), which directly optimises for a task specific distance metric. Naskar et el. [18] proposes to use energy based models, and the reranker is optimised for a pair-wise margin loss on hypotheses sampled from the output space. Lee et el. [122] maps the N-best hypotheses to a probability distribution according to the desired metric. In this thesis, the discriminative line of work is adopted in order to directly optimise for the metric of interest, and the reranker model discussed below is an extension of Lee et el.'s work [122].

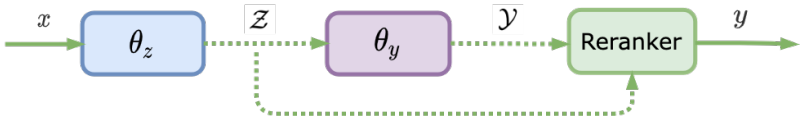


Fig. 4.8 Illustration of a reranking process operating on a two-module cascaded system

The reranking idea can be generalised to multimodular cascaded systems. Figure 4.8 illustrates a reranking pipeline operating on a two-module cascaded system. Individual modules in the cascaded systems are separately trained, and N-best hypotheses are generated for both the intermediate variable $\mathcal{Z}$ and the system output $\mathcal{Y}$. The reranker takes into account hypotheses in $\mathcal{Z}$, and predicts a new rank ordering for $\mathcal{Y}$:

$$\hat{\boldsymbol{y}}_{1:N} = \text{argmax}_{\boldsymbol{y}_{1:N} \in \mathcal{Y}} P(\boldsymbol{y}_{1:N} | \mathcal{Z}; \boldsymbol{\theta}_r) \tag{4.22}$$

where $\hat{\boldsymbol{y}}_{1:N}$ denotes the optimal output according to the reranker posterior, and $\boldsymbol{\theta}_r$ denotes the reranker parameters. In addition to mitigating exposure bias, adding rerankers to cascaded systems has added merits. Hypotheses from the upstream module can be directly accessed by the reranker without additional modifications to the cascaded system. It helps propagate richer information across modules, and thus mitigate the error propagation issue. As an alternative to sequence-level training, rerankers can be directly optimised towards the metric of interest. The reranker training process is described below.

Given a cascaded system with individually trained modules, an input sequence $\boldsymbol{x}_{1:T}$ can be mapped into a set of M_z hypotheses of the intermediate variable $\boldsymbol{z}_{1:L}$, each of which is used to generate M_y hypotheses of the output $\boldsymbol{y}_{1:N}$:

$$\hat{\boldsymbol{z}}^{(i)}_{1:L} \sim P(\boldsymbol{z}_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z) \quad \mathcal{Z} = \{\hat{\boldsymbol{z}}^{(1)}_{1:L}, ..., \hat{\boldsymbol{z}}^{(M_z)}_{1:L}\} \tag{4.23}$$

$$\hat{\boldsymbol{y}}^{(i,j)}_{1:N} \sim P(\boldsymbol{y}_{1:N} | \hat{\boldsymbol{z}}^{(i)}_{1:L}; \boldsymbol{\theta}_y) \quad \mathcal{Y}^{(i)} = \{\hat{\boldsymbol{y}}^{(i,1)}_{1:L}, ..., \hat{\boldsymbol{y}}^{(i,M_y)}_{1:L}\} \tag{4.24}$$

$$\mathcal{Y} = \{\mathcal{Y}^{(1)} ... \mathcal{Y}^{(i)} ... \mathcal{Y}^{(M_z)}\} \tag{4.25}$$

where $\mathcal{Z}$ denotes the top M_z hypotheses from the upstream module, and $\hat{\boldsymbol{z}}_{1:L}^{(i)}$ denotes the i^{th} candidate in $\mathcal{Z}$. $\mathcal{Y}^{(i)}$ denotes the top M_y hypotheses generated from $\hat{\boldsymbol{z}}_{1:L}^{(i)}$ through the downstream module, $\hat{\boldsymbol{y}}_{1:N}^{(i,j)}$ denotes the j^{th} candidate in $\mathcal{Y}^{(i)}$, and $\mathcal{Y}$ denotes the whole hypotheses set of size $M_z M_y$. Individual modules $\boldsymbol{\theta}_z$ and $\boldsymbol{\theta}_y$ can be separately trained with domain specific corpora $\mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z})$ and $\mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y})$, whereas the reranker training requires end-to-end data from the target domain $\mathcal{D}_{\text{tgt}}(\boldsymbol{x},\boldsymbol{y})$. By taking into account the intermediate hypotheses $\mathcal{Z}$, the reranker aims is to produce a scalar score for each $\hat{\boldsymbol{y}}_{1:N}^{(i,j)} \in \mathcal{Y}$ indicating the quality of the candidate[10]:

$$f(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \mathcal{Z}; \boldsymbol{\theta}_r) \tag{4.26}$$

where $\boldsymbol{\theta}_r$ indicates the learnable reranker parameters, and the scoring function f can be normalised over the candidate set $\mathcal{Y}$ to form a score distribution[11]:

$$P(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}|\mathcal{Z};\boldsymbol{\theta}_r) = \tfrac{1}{Z_{\text{hyp}}} \exp\{f(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \mathcal{Z}; \boldsymbol{\theta}_r)\} \tag{4.27}$$

$$Z_{\text{hyp}} = \textstyle\sum_{\hat{\boldsymbol{y}}_{1:N}^{(i,j)} \in \mathcal{Y}} \exp\{f(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \mathcal{Z}; \boldsymbol{\theta}_r)\} \tag{4.28}$$

In order to optimise for $\boldsymbol{\theta}_r$, the training process minimises the distance between the hypothesis distribution $P(\boldsymbol{y}_{1:N}|\mathcal{Z};\boldsymbol{\theta}_r)$ with a reference distribution $P(\boldsymbol{y}_{1:N})$:

$$\mathcal{L}(\boldsymbol{\theta}_r) = \text{KL}\{P(\boldsymbol{y}_{1:N})||P(\boldsymbol{y}_{1:N}|\mathcal{Z};\boldsymbol{\theta}_r)\} \quad \{\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x},\boldsymbol{y}) \tag{4.29}$$

where KL divergence is used as the distance measure, and the end-to-end training instances come from the target domain. The reference distribution can be defined with a task specific metric function μ:

$$P(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}) = \tfrac{1}{Z_{\text{ref}}} \exp\{\mu(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \boldsymbol{y}_{1:N})/\mathcal{T}\} \tag{4.30}$$

$$Z_{\text{ref}} = \textstyle\sum_{\hat{\boldsymbol{y}}_{1:N}^{(i,j)} \in \mathcal{Y}} \exp\{\mu(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \boldsymbol{y}_{1:N})/\mathcal{T}\} \tag{4.31}$$

where $\boldsymbol{y}_{1:N}$ denotes the reference output. $P(\boldsymbol{y}_{1:N})$ is the distribution of the reference scores $\mu(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \boldsymbol{y}_{1:N})$ normalised over the entire candidate set $\mathcal{Y}$, and $\mathcal{T}$ is the temperature coefficient controlling its smoothness.

[10] A more general scoring function is $f(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \mathcal{Z}, \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_r)$, which also accounts for the input $\boldsymbol{x}_{1:T}$. In practice, $\boldsymbol{x}_{1:T}$ is often the input speech, and it is difficult to extract correlations between speech and text without a large amount of data. Therefore, only discrete information $\mathcal{Z}$ is considered here for the purpose of reranker training.

[11] For simplicity purposes, the hypothesis probability of an instance $\hat{\boldsymbol{y}}_{1:N}^{(i,j)}$ is expressed as $P(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}|\mathcal{Z};\boldsymbol{\theta}_r)$, short for $P(\boldsymbol{y}_{1:N} = \hat{\boldsymbol{y}}_{1:N}^{(i,j)}|\mathcal{Z};\boldsymbol{\theta}_r)$. Similarly, $P(\boldsymbol{y}_{1:N} = \hat{\boldsymbol{y}}_{1:N}^{(i,j)})$ is expressed as $P(\hat{\boldsymbol{y}}_{1:N}^{(i,j)})$.

The scoring function f is central to the reranking process. It produces a scalar score indicating how likely each hypothesis $\hat{\boldsymbol{y}}_{1:N}^{(i,j)}$ is the top-ranked candidate among $\mathcal{Y}$ given the intermediate hypothesis set $\mathcal{Z}$. Under the simplest setup, the scoring function only accounts for the sequence $\hat{\boldsymbol{z}}_{1:L}^{(i)}$ that is used to generate the output hypothesis $\hat{\boldsymbol{y}}_{1:N}^{(i,j)} \in \mathcal{Y}^{(i)}$:

$$f(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \hat{\boldsymbol{z}}_{1:L}^{(i)}; \boldsymbol{\theta}_r) = \sigma_{\tanh}(\boldsymbol{W}\boldsymbol{h} + b) \tag{4.32}$$

$$\boldsymbol{h} = g(\hat{\boldsymbol{z}}_{1:L}^{(i)}, \hat{\boldsymbol{y}}_{1:N}^{(i,j)}; \boldsymbol{\theta}_h) \tag{4.33}$$

where g is a feature extraction process that captures the correlation between the input and output sequences of the downstream module $\boldsymbol{\theta}_z$, and further produces a hidden representation $\boldsymbol{h}$. The feature vector $\boldsymbol{h}$ is then fed through a feed-forward layer to reach a scalar score. A common practice to extract correlations between a sequence pair is to use a Transformer encoder based architecture to produce a contextual feature representation. Several large pre-trained Transformer encoders can be used for g, the exact form of which depends on the nature of the task. For example in the speech translation task, XLM-R [40] (a Transformer-based multilingual language model) is used to extract the bilingual context between the transcription and translation pairs. More detailed design choices will be discussed in Chapter 6 and 7.

In order to propagate more information from the upstream module, the feature extraction process can be extended to account for more hypotheses from $\mathcal{Z}$. The scoring function in Equation 4.32 is updated to be conditioned on a subset $\tilde{\mathcal{Z}} \subset \mathcal{Z}$:

$$f(\hat{\boldsymbol{y}}_{1:N}^{(i,j)}, \tilde{\mathcal{Z}}; \boldsymbol{\theta}_r) = \sigma_{\tanh}(\boldsymbol{W}\boldsymbol{h} + b) \tag{4.34}$$

$$\tilde{\mathcal{Z}} = \{\hat{\boldsymbol{z}}_{1:L}^{(1)}, ..., \hat{\boldsymbol{z}}_{1:L}^{(M)}\} \tag{4.35}$$

where the feature extraction process to obtain $\boldsymbol{h}$ can take various forms to account for the hypotheses in $\tilde{\mathcal{Z}}$. A straight-forward approach is to simply concatenate the hypotheses in $\tilde{\mathcal{Z}}$ with the candidate $\hat{\boldsymbol{y}}_{1:N}^{(i,j)}$:

$$\boldsymbol{h} = g(\hat{\boldsymbol{z}}_{1:L}^{(i)}, \{\hat{\boldsymbol{z}}_{1:L}^{(1)}, ..., \hat{\boldsymbol{z}}_{1:L}^{(i-1)}, \hat{\boldsymbol{z}}_{1:L}^{(i+1)}, ..., \hat{\boldsymbol{z}}_{1:L}^{(M)}\}, \hat{\boldsymbol{y}}_{1:N}^{(i,j)}; \boldsymbol{\theta}_h) \tag{4.36}$$

Vanilla concatenation treats all hypotheses in $\tilde{\mathcal{Z}}$ indifferently, and poor quality hypotheses could potentially disrupt the feature vector $\boldsymbol{h}$. An alternative to vanilla concatenation is to

adopt an attention mechanism, which assigns larger weights to the most relevant context:

$$\boldsymbol{h} = \sum_{m=1}^{M} \alpha_m \boldsymbol{h}_m \quad \sum_{m=1}^{M} \alpha_m = 1 \tag{4.37}$$

$$\alpha_m = f_{\text{att}}(\boldsymbol{h}_m, \boldsymbol{h}_i; \boldsymbol{\theta}_{\text{att}}) \tag{4.38}$$

$$\boldsymbol{h}_m = g(\hat{\boldsymbol{z}}_{1:L}^{(m)}, \hat{\boldsymbol{y}}_{1:N}^{(i,j)}; \boldsymbol{\theta}_h) \tag{4.39}$$

where f_{att} is the attention function, and the overall feature representation $\boldsymbol{h}$ is a weighted sum over the feature vectors $\boldsymbol{h}_{m\in\{1,\ldots,M\}}$. The feature vector $\boldsymbol{h}_i$ is generated from the matching hypotheses pair $\hat{\boldsymbol{z}}_{1:L}^{(i)}$ and $\hat{\boldsymbol{y}}_{1:N}^{(i,j)} \in \mathcal{Y}^{(i)}$, and the rest of the features extract correlations between unpaired hypotheses $\hat{\boldsymbol{z}}_{1:L}^{(m)}$ and $\hat{\boldsymbol{y}}_{1:N}^{(i,j)} \in \hat{\mathcal{Y}}^{(i)}$. The attention mechanism accounts for the relative saliencies between $\boldsymbol{h}_i$ and $\boldsymbol{h}_{m\in\{1,\ldots,M\}}$, which allows more flexibilities in weighing relevant hypotheses in $\tilde{\mathcal{Z}}$ over less correlated ones.

4.3.2 Embedding passing

Compressing information into discrete sequences would cause information loss. For example, speech transcriptions cannot carry prosodic information from speech due to their discrete nature. Alternatively, continuous embeddings can be used to propagate richer context across modules. Embedding passing with a two module integrated system can be formulated as:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \approx P(\boldsymbol{y}_{1:N}|\boldsymbol{e}_{1:L}; \boldsymbol{\theta}_y^{\text{EP}}) \tag{4.40}$$

$$\boldsymbol{e}_{1:L} = f(\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_e) \tag{4.41}$$

where $\boldsymbol{e}_{1:L}$ denotes the embedding connection linking the input $\boldsymbol{x}_{1:T}$ and output $\boldsymbol{y}_{1:N}$. $\boldsymbol{\theta}_e$ and $\boldsymbol{\theta}_y^{\text{EP}}$ parameterise the upstream and downstream modules respectively. The idea is to use continuous variables as the modular connection to carry richer upstream context, and thus help mitigate information loss in multimodular systems. Another advantage of using embeddings is to allow gradient backpropagation across modular connection. Therefore, all modules in the integrated system can be jointly optimised towards the metric of interest, which helps mitigate error propagation. The limitation of embedding passing is that it requires a small amount of target domain data to fine-tune the system in an end-to-end fashion. It is also constrained that the tokenisation and vocabulary of the intermediate variables connecting the upstream and downstream modules need to be matched, such that they can be mapped into the same embedding space. The following sections will discuss how the embeddings can be extracted, as well as the training of the integrated system under embedding passing.

Embedding extraction

An embedding is usually a structured representation that allows efficient training under limited end-to-end data without compromising model capacity. For multimodular systems, module connections with highly abstracted information lead to information loss and early decision errors, whereas low abstraction level results in low data efficiency. For example in the speech translation task, highly abstracted word sequences (i.e. in cascaded systems) fail to capture prosodic information and suffer from error propagation, whereas directly passing raw waveforms (i.e. in end-to-end systems) to perform downstream translation yields poor data efficiency. Under limited end-to-end data, this work adopts embeddings with a relatively high abstraction level for higher data efficiency. The embeddings $\boldsymbol{e}_{1:L}$ are directly associated with the intermediate sequences $\boldsymbol{z}_{1:L}$ in cascaded systems, and each embedding vector $\boldsymbol{e}_l$ can be viewed as a continuous representation of the context associated with each token $\boldsymbol{z}_l$. For example in speech translation, the embedding vector sequence has the same length as the transcription token sequence, and they are expected to encapsulate both the acoustic and textual context associated with each token. Figure 4.9 shows an illustration of the embedding extraction process from speech inputs.

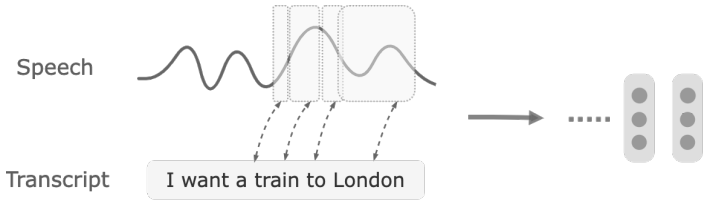


Fig. 4.9 Illustration of an embedding extraction process from speech inputs. The speech and the transcription sequence are aligned, and the relevant context is summarised into an embedding sequence.

Attention-based sequence-to-sequence models are adopted here for embedding extraction. The attention mechanism attends over relevant segments in the input sequence that match with individual tokens in the output sequence, and produces corresponding feature vectors. The extracted features corresponding to individual tokens are used as the embedding connection to the downstream module. Figure 4.10 shows an illustration of a two-module integrated system with embedding passing. The left part shows the transitions of hidden states with an increasing level of abstraction in the upstream module, and the right part shows the downstream module. The integrated system combines partially activated upstream and downstream modules, with the inactive blocks shaded in the plot. The individual modules can be initialised using domain specific data, and the system can be jointly adapted to the target domain with end-to-end data. When the input is speech sequences, the upstream module can be seen as an attention-based encoder decoder ASR (discussed in Section 3.2.2),

and the embedding extraction can be seen as an alignment process between speech and transcripts. The attention-based approach gives a soft alignment which allows overlapping attention intervals between neighbouring transcription tokens. An alternative approach is to directly extract hard embeddings with timestamps produced from hybrid ASR (discussed in Section 3.2.1), which are strictly non-overlapping between neighbouring tokens. More comparison on the soft and hard embedding extractions are discussed in Chapter 5.2.

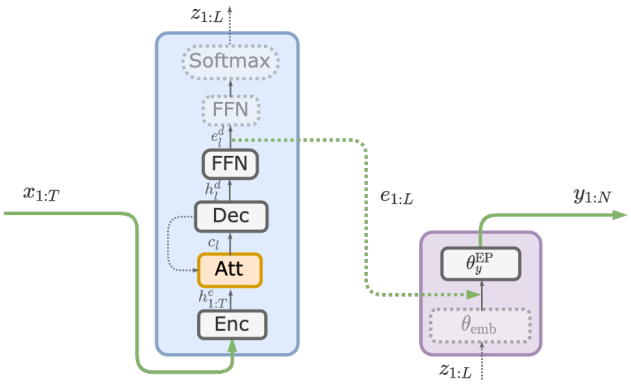


Fig. 4.10 Illustration of an integrated system with embedding passing. The green arrows indicate the information flow across the integrated system, and the shaded blocks are inactive during the in-domain end-to-end training.

The embedding extraction process follows a standard attention-based encoder-decoder formulation with $\boldsymbol{\theta}_e = \{\boldsymbol{\theta}_e^{\text{enc}}, \boldsymbol{\theta}_e^{\text{att}}, \boldsymbol{\theta}_e^{\text{dec}}, \boldsymbol{\theta}_e^{\text{ffn}}\}$. The upstream module first maps the input sequence $\boldsymbol{x}_{1:T}$ into a sequence of hidden states $\boldsymbol{h}_{1:T}^e$:

$$\boldsymbol{h}_{1:T}^e = f_{\text{enc}}(\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_e^{\text{enc}}) \tag{4.42}$$

where f_{enc} is the encoding function in sequence-to-sequence models. For speech inputs, f_{enc} will also act as a down-sampling function to reduce redundancies in speech. The encoder hidden states are fed through an attention mechanism, followed by a decoder:

$$\boldsymbol{\alpha}_l = f_{\text{att}}(\boldsymbol{h}_l^d, \boldsymbol{h}_{1:T}^e; \boldsymbol{\theta}_e^{\text{att}}) \quad \boldsymbol{c}_l = \textstyle\sum_{t=1}^{T} \alpha_{l,t} \boldsymbol{h}_t^e \tag{4.43}$$

$$\boldsymbol{h}_l^d = f_{\text{dec}}(\boldsymbol{h}_{l-1}^d, \boldsymbol{z}_{l-1}, \boldsymbol{c}_l; \boldsymbol{\theta}_e^{\text{dec}}) \tag{4.44}$$

$$\boldsymbol{e}_l^d = f_{\text{ffn}}(\boldsymbol{c}_l, \boldsymbol{h}_l^d; \boldsymbol{\theta}_e^{\text{ffn}}) \tag{4.45}$$

where f_{att} denotes the attention mechanism, f_{dec} is the decoder function, and f_{ffn} is a feed-forward layer. The hidden states of the upstream sequence-to-sequence module achieve an increasing level of information abstraction: $\boldsymbol{h}_{1:T}^e$ is the encoder hidden state that has the same length as the input; $\boldsymbol{c}_{1:L}$ is the context vector that directly derives from weighing the encoder

hidden states; $\boldsymbol{h}^d_{1:L}$ is the decoder hidden state that takes into account both the input context and the previously predicted tokens; $\boldsymbol{e}^d_{1:L}$ is a linear transform of the concatenated $\boldsymbol{c}_{1:L}$ and $\boldsymbol{h}^d_{1:L}$. $\boldsymbol{e}^d_{1:L}$ is just one layer below the softmax prediction, and it is referred to as the dynamic embedding since different embedding values may result in the same token prediction. The embedding connection can be chosen among the upstream hidden states, and is directly used as the input to the downstream module.

Sperber et el. [194] proposes attention passing, which uses context vectors $\boldsymbol{c}_{1:L}$ extracted from the attention mechanism as inputs to downstream tasks. With a low level of information abstraction, context vectors are robust against potential exposure bias, yet tend to suffer from low data efficiency. Under limited end-to-end data, it is favourable to use features with a relatively high level information abstraction for higher data efficiency. In order to strike a balance between information richness and efficient use of data, the dynamic embedding $\boldsymbol{e}^d_{1:L}$ is used as the modular connection in this thesis. In some cases, the dynamic embedding $\boldsymbol{e}^d_{1:L}$ can be concatenated with the static word embedding, and together they form the input to the downstream module. Dynamic embeddings have a many-to-one mapping from embeddings to the token prediction, whereas static embeddings have a one-to-one correspondence with the tokens, thus called static. The modular connection $\boldsymbol{e}^d_{1:L}$ maintains a richer information flow compared with discrete tokens, and it also allows higher data efficiency compared with the lower level hidden states.

Training

The embedding passing approach preserves the modular concept, and therefore at the initialisation stage, individual modules can be separately initialised with domain specific corpora using auxiliary objectives. The initialisation process takes three steps, and Figure 4.11 shows an illustration of the auxiliary tasks. The first step trains the upstream module with the auxiliary objective to optimise $\boldsymbol{\theta}_z$ with $\mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z})$:

$$\mathcal{L}(\boldsymbol{\theta}_z) = -\log P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z) \quad \{\boldsymbol{x}_{1:T}, \boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x}, \boldsymbol{z}) \tag{4.46}$$

The auxiliary output $\boldsymbol{z}_{1:L}$ follows the standard autoregressive sequence generation, and is conditioned on the dynamic embeddings $\boldsymbol{e}^d_{1:L}$:

$$P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z) = \prod_{l=1}^{L} P(\boldsymbol{z}_l|\boldsymbol{z}_{1:l-1}, \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_z) \approx \prod_{l=1}^{L} P(\boldsymbol{z}_l|\boldsymbol{e}^d_l; \boldsymbol{\theta}_{\text{aux}}) \tag{4.47}$$

$$\boldsymbol{e}^d_{1:L} = f(\boldsymbol{x}_{1:T}; \boldsymbol{\theta}_e) \quad \boldsymbol{\theta}_z = \{\boldsymbol{\theta}_e, \boldsymbol{\theta}_{\text{aux}}\} \tag{4.48}$$

The second step matches the static $\boldsymbol{e}^s_l$ and dynamic $\boldsymbol{e}^d_l$ embeddings. The upstream module is fixed and the embedding mapping function $\boldsymbol{\theta}_{\text{emb}}$ in the downstream module is updated with

the same corpora used in the first step. The training objective is to minimise the L_2 distance between the static and dynamic embeddings:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{emb}}) = \sum_{l=1}^{L} ||\boldsymbol{e}_l^d - \boldsymbol{e}_l^s|| \quad \boldsymbol{e}_l^s = f(\boldsymbol{z}_l; \boldsymbol{\theta}_{\text{emb}}) \quad \{\boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{z}) \tag{4.49}$$

where $\boldsymbol{e}_l^s$ denotes the static embedding produced from θ_{emb}. In cases where both the static $\boldsymbol{e}_{1:L}^s$ and dynamic $\boldsymbol{e}_{1:L}^d$ embeddings are fed to the downstream module, the embedding matching step can be skipped. Once the static and dynamic embeddings are matched and fixed, the third step is to initialise the downstream module with the auxiliary training data $\mathcal{D}_{\text{dn}}(\boldsymbol{z}, \boldsymbol{y})$:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{y}}^{\text{EP}}) = -\log P(\boldsymbol{y}_{1:N} | \boldsymbol{z}_{1:L}; \boldsymbol{\theta}_{\text{emb}}, \boldsymbol{\theta}_{\text{y}}^{\text{EP}}) \quad \{\boldsymbol{z}_{1:L}, \boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{z}, \boldsymbol{y}) \tag{4.50}$$

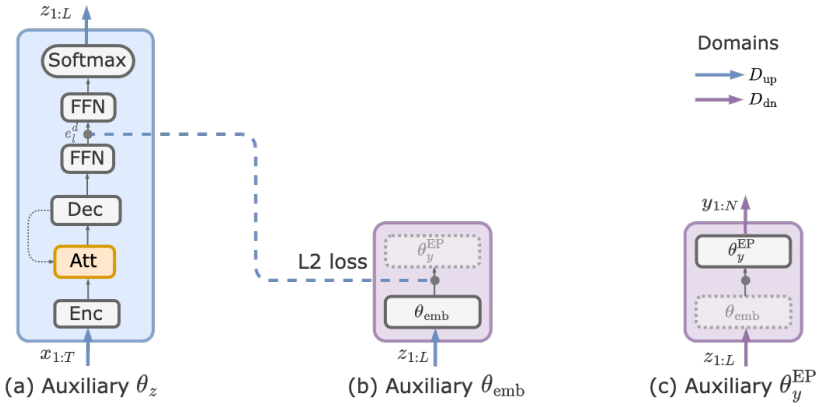


Fig. 4.11 Illustration of the auxiliary training process of an integrated system with embedding passing (modules are connected via dynamic embeddings $\boldsymbol{e}_{1:L}^d$)

Auxiliary task training is an effective approach to exploit knowledge from domain specific data. Part of the parameters in the integrated system are initialised with the auxiliary pre-training. With a good initialisation point, the integrated system can reach convergence more efficiently when trained on a limited amount of end-to-end data. At the in-domain training stage, the integrated system can be jointly trained under the target domain:

$$\mathcal{L}(\boldsymbol{\theta}_e, \boldsymbol{\theta}_{\text{y}}^{\text{EP}}) = -\log P(\boldsymbol{y}_{1:N} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_e, \boldsymbol{\theta}_{\text{y}}^{\text{EP}}) \quad \{\boldsymbol{x}_{1:T}, \boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, \boldsymbol{y}) \tag{4.51}$$

Gradients can back propagate through the continuous embedding connection, and thus the upstream $\boldsymbol{\theta}_e$ and downstream $\boldsymbol{\theta}_y^{\text{EP}}$ can be simultaneously optimised for the system output.

4.4 End-to-end systems

This section considers end-to-end systems, which no longer have the notion of modules. Given input $\boldsymbol{x}_{1:T}$ and output $\boldsymbol{y}_{1:N}$, an end-to-end system can be directly formulated with:

$$P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \tag{4.52}$$

where $\boldsymbol{\theta}$ is often modelled with an attention-based encoder-decoder architecture. Figure 4.12 shows an illustration of an end-to-end system.

Fig. 4.12 Illustration of an end-to-end system

Different from cascaded and integrated systems, the end-to-end system cannot be pre-trained with auxiliary tasks, and requires end-to-end corpora for training:

$$\mathcal{L}(\boldsymbol{\theta}) = -\log P(\boldsymbol{y}_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \quad \{\boldsymbol{x}_{1:T},\boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x},\boldsymbol{y}) \tag{4.53}$$

Directly training with end-to-end corpora avoids issues seen in cascaded systems, such as error propagation, information loss or domain mismatch. However, strong modelling power comes at the cost of low data efficiency. End-to-end systems require a large amount of end-to-end data for training, which is a scarce resource due to its high annotation cost. The following sections will discuss approaches to improve end-to-end models through boosting training data, as well as improving data efficiency by learning more general knowledge.

4.4.1 Data generation

When faced with constraints on data availability, a straight forward approach is to obtain more data. While obtaining manual annotations for spoken language applications can be costly, using machine generated annotations is a cheaper and more efficient alternative. There is usually an abundance of corpora in auxiliary domains $\mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z})$ and $\mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y})$, where $\boldsymbol{z}_{1:L}$ is the intermediate modular connection used in cascaded systems. As shown in Figure 4.13,

two transition cycles can be trained using the auxiliary data:

$$\mathcal{L}(\boldsymbol{\theta}_{xz}) = -\log P(\boldsymbol{z}_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_{xz}) \quad \{\boldsymbol{x}_{1:T},\boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z}) \tag{4.54}$$

$$\mathcal{L}(\boldsymbol{\theta}_{zx}) = -\log P(\boldsymbol{x}_{1:T}|\boldsymbol{z}_{1:L};\boldsymbol{\theta}_{zx}) \quad \{\boldsymbol{x}_{1:T},\boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z}) \tag{4.55}$$

$$\mathcal{L}(\boldsymbol{\theta}_{zy}) = -\log P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L};\boldsymbol{\theta}_{zy}) \quad \{\boldsymbol{z}_{1:L},\boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y}) \tag{4.56}$$

$$\mathcal{L}(\boldsymbol{\theta}_{yz}) = -\log P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L};\boldsymbol{\theta}_{yz}) \quad \{\boldsymbol{z}_{1:L},\boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y}) \tag{4.57}$$

For example in the speech translation task, $\boldsymbol{\theta}_{xz}$ and $\boldsymbol{\theta}_{zx}$ denote speech recognition (ASR) and speech synthesis (TTS) tasks, whereas $\boldsymbol{\theta}_{zy}$ and $\boldsymbol{\theta}_{yz}$ are neural machine translation (NMT) tasks running in reverse directions.

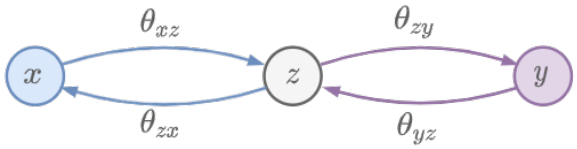


Fig. 4.13 Two transition cycles

Following the transition cycles, end-to-end corpora can be generated with the partially annotated data: $\{\boldsymbol{x}_{1:T},\boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z})$ can be augmented with $\boldsymbol{\theta}_{zy}$ to obtain $\hat{\boldsymbol{y}}_{1:N}$, and $\{\boldsymbol{z}_{1:L},\boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y})$ can be augmented with $\boldsymbol{\theta}_{zx}$ to obtain $\hat{\boldsymbol{x}}_{1:T}$:

$$\hat{\boldsymbol{y}}_{1:N} \sim P(\boldsymbol{y}_{1:N}|\boldsymbol{z}_{1:L};\boldsymbol{\theta}_{zy}) \;\; \{\boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z}) \quad \rightarrow \{\boldsymbol{x}_{1:T},\boldsymbol{z}_{1:L},\hat{\boldsymbol{y}}_{1:N}\} \in \mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{y}) \tag{4.58}$$

$$\hat{\boldsymbol{x}}_{1:T} \sim P(\boldsymbol{x}_{1:N}|\boldsymbol{z}_{1:L};\boldsymbol{\theta}_{zx}) \;\; \{\boldsymbol{z}_{1:L}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y}) \quad \rightarrow \{\hat{\boldsymbol{x}}_{1:T},\boldsymbol{z}_{1:L},\boldsymbol{y}_{1:N}\} \in \mathcal{D}_{\text{dn}}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{y}) \tag{4.59}$$

Following the data augmentation pipeline, the end-to-end pairs $\{\boldsymbol{x}_{1:T},\hat{\boldsymbol{y}}_{1:N}\}$ and $\{\hat{\boldsymbol{x}}_{1:T},\boldsymbol{y}_{1:N}\}$ can be used for end-to-end training. In the speech translation example, ASR corpora $\mathcal{D}_{\text{up}}(\boldsymbol{x},\boldsymbol{z})$ can be augmented to obtain hypothesised translations $\hat{\boldsymbol{y}}_{1:N}$ by running forward translation $\boldsymbol{\theta}_{zy}$, and NMT corpora $\mathcal{D}_{\text{dn}}(\boldsymbol{z},\boldsymbol{y})$ can be augmented to obtain synthesised speech $\hat{\boldsymbol{x}}_{1:T}$ by conducting speech synthesis $\boldsymbol{\theta}_{zx}$. Pino et al. [163] has shown that, for speech translation tasks, augmenting ASR corpora to infer automatic translations are more effective than augmenting NMT corpora. Bentivogli [17] attests that with data generation and knowledge transfer techniques, end-to-end systems can achieve comparable performance as cascaded systems. However, the synthetically generated data pairs may still suffer from degraded quality, and fine-tuning the end-to-end system in the target domain will further improve its performance.

4.4.2 Meta-learning

Training end-to-end systems on pseudo references might lead to degraded performance due to biases introduced from data augmentation. An alternative approach is to tackle the low data efficiency issue through learning more general knowledge from the readily available datasets. Meta-learning [204] refers to a group of algorithms that learns the target task from a suite of other related prediction tasks. The recent model-agnostic meta-learning algorithm (MAML) [55] proposes to learn representations through relevant tasks, so that they can be efficiently adapted to the target task with a few steps of gradient update.

The MAML idea can be extended to spoken language tasks and becomes a modality-agnostic meta-learning approach [93]. The training process is split into two phases: the meta-learning phase that trains with auxiliary tasks, and the fine-tuning phase that trains with in-domain end-to-end data. In the meta-learning phase, the goal is to find a good initialisation point for the target task using the related auxiliary tasks. To handle different modalities in auxiliary tasks, additional compression layers can be added, which will be activated depending on the task being conducted. For example, the compression layers for input speech sequences will not be activated when the inputs are text sequences. In each iteration i, an auxiliary gradient is calculated for task τ sampled from a set of auxiliary tasks:

$$\boldsymbol{\theta}_{\tau}^{i} = \boldsymbol{\theta}^{i-1} - \alpha \nabla_{\boldsymbol{\theta}^{i-1}} \mathcal{L}(\mathcal{D}_{\tau}; \boldsymbol{\theta}^{i-1}) \tag{4.60}$$

where $\boldsymbol{\theta}^{i-1}$ is the parameter from the previous iteration, $\boldsymbol{\theta}_{\tau}^{i}$ is the auxiliary parameter of task τ, and $\mathcal{D}_{\tau}$ is the training batch of task τ. The meta-gradient can then be updated with:

$$\boldsymbol{\theta}^{i} = \boldsymbol{\theta}^{i-1} - \beta \nabla_{\boldsymbol{\theta}^{i-1}} \mathcal{L}(\mathcal{D}'_{\tau}; \boldsymbol{\theta}_{\tau}^{i}) \tag{4.61}$$

where $\boldsymbol{\theta}^{i}$ is the parameter resulted from the current iteration, $\mathcal{D}'_{\tau}$ is the test batch of task τ, and α, β are learning rates. The intuition is that training with $\mathcal{D}_{\tau}$ simulates the task-specific learning, and evaluating on $\mathcal{D}'_{\tau}$ extracts the more generalisable gradient updates from the auxiliary parameters. Finally, after I iterations of meta-learning, the parameter $\boldsymbol{\theta}^{I}$ learns general knowledge from the auxiliary tasks. It can then be used as the initialisation point for fine-tuning, which follows the conventional end-to-end training.

4.5 Summary

Under limited end-to-end corpora, it is challenging to build multimodular spoken language systems. This chapter reviewed three categories of spoken language systems with an increas-

ing level of integration. The cascaded system is composed of directly concatenated modules, which can be separately trained in their corresponding domains. Loosely coupled modules suffer from domain mismatch and error propagation issues, and thus domain adaptation and error mitigation approaches were described. The integrated system aims to propagate richer information across modular connections. The challenge lies in designing the system so that individual modules can be efficiently pre-trained with domain specific corpora, and then further jointly optimised with in-domain end-to-end data. Discrete information passing and embedding passing approaches were discussed to pass discrete and continuous sequences across the modular connection. The end-to-end system requires a large amount of end-to-end training data to reach convergence. Data augmentation and meta-learning approaches were discussed to tackle the low data efficiency issue.

The contributions of this chapter are three-fold. 1) It describes the general concept of multimodular combination, and discusses different module combination approaches for spoken language systems. 2) It proposes a general framework of reranking for multimodular systems, addressing both the error propagation and information loss issues. 3) An embedding passing approach is also proposed, which passes the upstream context to the downstream modules via continuous representations. The embedding connection allows richer information propagation as well as gradient backpropagation across modules, thus enabling joint optimisation of the multimodular system. The following Chapter 5, 6 and 7 will further apply the approaches proposed in this chapter to several spoken language tasks[12].

[12]Due to limited time and computing resource, not all approaches were experimented with. The main focus of this work is on investigating the cascaded and integrated multimodular systems.

Chapter 5

Spoken Disfluency Detection

Chapter 3 and 4 respectively discussed the formulations of individual modules and various module combination approaches. This chapter investigates module combination for the spoken disfluency detection (SDD) task, which is an important pre-processing step for many spoken language tasks. Section 5.1 gives an overview of the task and describes the cascaded pipeline of spoken disfluency detection. Section 5.2 discusses the embedding passing approach for tighter modular combination. Detailed evaluation metrics, experimental setups and analyses are reported in Section 5.3.

5.1 Task descriptions

Disfluencies are commonly seen in spontaneous speech, which includes filled pauses, repetitions and false starts. Human-to-human interactions conduct a natural de-noising process that extracts the underlying fluent context. On the other hand, disfluencies pose significant challenges in computer based spoken language applications [99, 85]. Spoken disfluency detection refers to the process of converting disfluent speech into fluent text sequences, which is an important pre-processing step for many spoken language tasks.

Previous works on disfluency detection mainly focus on text based approaches that take speech transcripts as inputs. The proposed methods include parsing based noisy channel models [28, 241], and tagging based detection models [87, 234]. Text based disfluency removal mainly operates on manual transcriptions of speech, and thus known as transcription disfluency detection (TDD). State-of-the-art TDD models [208, 228] achieves very high detection accuracy, yet their performance degrades significantly when operating on ASR transcriptions. The focus of this work is spoken disfluency detection (SDD), which accounts for the entire process of converting raw speech inputs into fluent text sequences. Figure 5.1 shows a vanilla cascaded SDD system. The main detection process consists of an automatic

speech recognition (ASR) module and a tagging based disfluency detection (DD) module (individual modules were described in Section 3.2 and 3.3). Fluent text can then be obtained by feeding the disfluent transcripts together with the corresponding disfluency labels into a trivial text removal process.

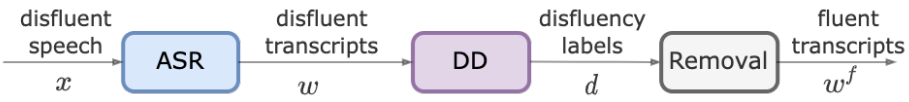


Fig. 5.1 Illustration of a cascaded spoken disfluency detection (SDD) system

Feeding the input speech $\boldsymbol{x}_{1:T}$ through the cascaded SDD pipeline, speech transcripts $w_{1:L}$ can be generated via the ASR module, and a sequence of binary disfluency tags $d_{1:L}$ can be generated via the DD module. The cascaded SDD process can be formulated as:

$$\hat{w}_{1:L} = \text{argmax}_{w_{1:L} \in \mathcal{W}} P(w_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_{\text{ASR}}) \tag{5.1}$$

$$P(d_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \approx P(d_{1:L} | \hat{w}_{1:L}; \boldsymbol{\theta}_{\text{DD}}) \approx \prod_{l=1}^{L} P(d_l | \hat{w}_{1:L}; \boldsymbol{\theta}_{\text{DD}}) \tag{5.2}$$

where $\hat{w}_{1:L}$ is the most likely hypotheses generated from the upstream ASR module. The downstream detection process directly operates on the ASR hypothesis, and disfluency tagging is modelled as a non-autoregressive process in this thesis. As discussed in Chapter 4, cascaded systems rely on a single discrete sequence to pass information, and thus tend to suffer from loss of information and error propagation issues. Under the context of spoken disfluency detection, erroneous transcriptions and loss of important prosodic cues, such as pauses and hesitations, could potentially lead to degraded detection performance. The focus of this chapter is therefore to explore tighter combination of the ASR and DD modules, and consequently improve the quality of the output fluent transcriptions.

5.2 Embedding passing

Vanilla cascaded disfluency removal suffers from ASR error propagation as well as loss of prosodic cues. This section describes the application of the embedding passing approach (introduced in Section 4.3.2) on the spoken disfluency detection task. The idea is to propagate richer acoustic information from the upstream ASR to the downstream DD module, in the form of continuous embeddings.

Figure 5.2 illustrates the embedding passing approach. An ASR module converts speech $\boldsymbol{x}_{1:T}$ into transcripts $w_{1:L}$, and for each word w_l in the speech transcripts, a corresponding acoustic representation $\boldsymbol{e}_l^d$ is extracted. The concatenation of the textual embeddings $\boldsymbol{e}_{1:L}^s$ and

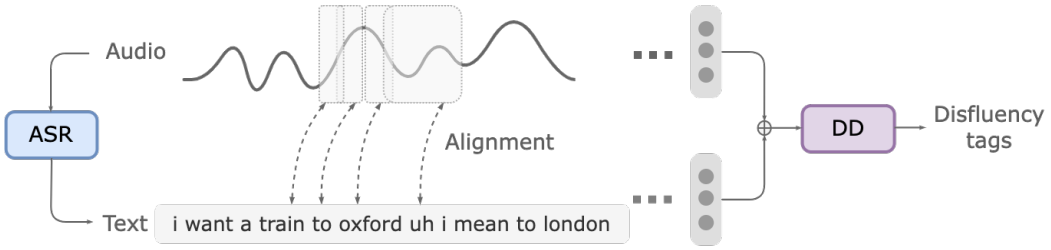


Fig. 5.2 Illustration of the embedding passing approach for spoken disfluency detection

acoustic embeddings $\boldsymbol{e}^{d}_{1:L}$ are passed on to the downstream disfluency detection module[1]:

$$\hat{w}_{1:L} = \text{argmax}_{w_{1:L}\in\mathcal{W}} P(w_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_{\text{ASR}}) \quad \boldsymbol{e}^{s}_{1:L} = f(\hat{w}_{1:L};\boldsymbol{\theta}_{\text{emb}}) \tag{5.3}$$

$$P(d_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(d_{1:L}|\hat{w}_{1:L},\boldsymbol{x}_{1:T};\boldsymbol{\theta}^{\text{EP}}_{\text{DD}}) \approx P(d_{1:L}|\{\boldsymbol{e}^{s}_{1:L},\boldsymbol{e}^{d}_{1:L}\};\boldsymbol{\theta}^{\text{EP}}_{\text{DD}}) \tag{5.4}$$

where the disfluency tags $d_{1:L}$ are conditioned on both the textual and acoustic information, and $\boldsymbol{\theta}_{\text{emb}}$ parameterises the embedding mapping function. The key to embedding passing is to extract acoustic embedding sequence $\boldsymbol{e}^{d}_{1:L}$, which has the same length as the transcription sequence, by aligning the audio $\boldsymbol{x}_{1:T}$ with the transcription $\hat{w}_{1:L}$ sequences:

$$\boldsymbol{e}^{d}_{1:L} = f(\hat{w}_{1:L},\boldsymbol{x}_{1:T};\boldsymbol{\theta}_{e}) \tag{5.5}$$

As shown in Figure 5.3, two acoustic embedding extraction processes are considered here.

Hard alignment - timestamps

Hybrid ASR models produce transcriptions together with the corresponding per-word timestamps, specifying the start and end time of each word. A straight-forward approach is to extract acoustic features following the timestamps. A recurrent layer first runs over the entire speech sequence $\boldsymbol{x}_{1:T}$ to obtain more abstract contextual hidden vectors $\boldsymbol{h}_{1:T}$.

$$\boldsymbol{h}_{1:T} = f(\boldsymbol{x}_{1:T};\boldsymbol{\theta}_{\text{enc}}) \tag{5.6}$$

where a bidirectional LSTM is used for $\boldsymbol{\theta}_{\text{enc}}$. For each token $\hat{w}_l$, the acoustic signals within the corresponding time window $\{t_{l1},t_{l2}\}$ is kept, and the rest of the sequence is masked out. As shown in Figure 5.3a, a local attention mechanism is adopted to convert the windowed

[1]The textual and acoustic embeddings respectively correspond to the static and dynamic embeddings discussed in Section 4.3.2. Here, the concatenation of both embeddings are fed to the downstream module in order to account for both textual and prosodic information.

hidden vector sequence into a fixed-length feature embedding $\boldsymbol{e}_l^d$ for word $\hat{w}_l$:

$$\boldsymbol{\alpha}_l = f(\hat{w}_l, \boldsymbol{h}_{t_{l1}:t_{l2}}; \boldsymbol{\theta}_{\text{att}}) \quad \boldsymbol{c}_l = \sum_{t=t_{l1}}^{t_{l2}} \alpha_{l,t} \boldsymbol{h}_t \tag{5.7}$$

$$\boldsymbol{e}_l^d = \boldsymbol{W}\boldsymbol{c}_l + \boldsymbol{b} \tag{5.8}$$

The timestamp based alignment provides a rigid correspondence between speech and text. It directly makes use of the by-product of hybrid ASR decoding, and the generated dynamic embeddings $\boldsymbol{e}_{1:L}^d$ can be used to form part of the input to the downstream disfluency detection. The hard embedding extraction process can be jointly optimised with the downstream module using the disfluency detection objective.

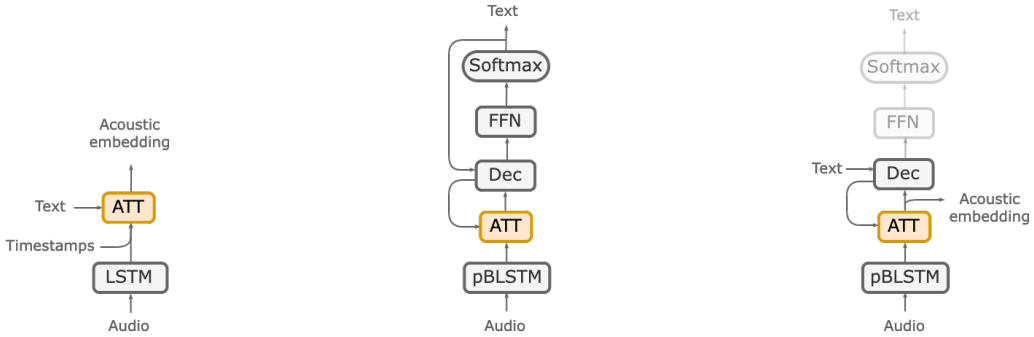


Fig. 5.3 Illustration of two alignment approaches for acoustic embedding extraction

Soft alignment - attention

The attention-based encoder decoder (AED) ASR framework (see Section 3.2.2) trains an attention mechanism over acoustics, which effectively offers an alignment between the input acoustic sequence and the output word tokens. The soft alignment approach makes use of such attention alignment trained with the AED ASR to extract relevant acoustic features. As described in Section 4.3.2, the feature extraction process consists of several steps. An acoustic encoder is first applied to reduce the time resolution and consequently speed up training. Here pyramidal bidirectional LSTM (pBLSTM) layers are adopted:

$$\boldsymbol{h}_t^j = f_{\text{enc}}(\boldsymbol{h}_{t-1}^j, [\boldsymbol{h}_{2t}^{j-1}, \boldsymbol{h}_{2t+1}^{j-1}]; \boldsymbol{\theta}_{\text{pBLSTM}}^j) \quad \boldsymbol{h}_{1:T}^0 = \boldsymbol{x}_{1:T} \quad j \in [1, J] \tag{5.9}$$

where $\boldsymbol{\theta}_{\text{pBLSTM}}^j$ parameterises the j^{th} layer, and each layer reduces the time resolution by a factor of two. The final encoded states $\boldsymbol{h}_{1:\tau}^J$ (τ is the time-reduced sequence length) are then fed through the attention mechanism to produce the speech to text alignments. For each

token w_l, a fixed length dynamic embedding $\boldsymbol{e}_l^d$ can be extracted as:

$$\boldsymbol{\alpha}_l = f_{\text{att}}(\boldsymbol{h}_l^d, \boldsymbol{h}_{1:\tau}^J; \boldsymbol{\theta}_{\text{att}}) \quad \boldsymbol{c}_l = \sum_{t=1}^{\tau} \alpha_{l,t} \boldsymbol{h}_t^J \tag{5.10}$$

$$\boldsymbol{h}_l^d = f_{\text{dec}}(\boldsymbol{h}_{l-1}^d, \hat{w}_{l-1}, \boldsymbol{c}_l; \boldsymbol{\theta}_{\text{dec}}) \tag{5.11}$$

$$\boldsymbol{e}_l^d = \boldsymbol{W}\boldsymbol{c}_l + \boldsymbol{b} \tag{5.12}$$

For the embedding extraction training, an auxiliary AED ASR training objective is adopted to optimise the attention alignment process (Figure 5.3b). For the downstream disfluency detection training, the soft embedding extraction process is fixed and operates under the ASR inference mode (Figure 5.3c). The word sequence $\hat{w}_{1:L}$ can be chosen as the predicted ASR hypotheses or any given word sequences[2].

The hard and soft alignment processes have their respective merits and drawbacks. Hard alignment can be directly trained together with its downstream disfluency detection, whereas soft alignment requires additional training of the auxiliary AED ASR. Soft alignment moves away from using explicit timestamps, thus achieving added flexibility in attending to overlapping intervals between neighbouring word in the speech sequence. Attending to the hesitations and silence intervals in between words may come in useful for the downstream disfluency detection task.

5.3 Experiments

This section analyses the impact of embedding passing on the spoken disfluency detection task, and contrasts the acoustic feature extraction processes with hard and soft alignment.

5.3.1 Evaluation metrics

F-score [167] is a standard evaluation metric for binary classification tasks, which measures the classification accuracy. Given a binary classification task that generates positive and

[2]End-to-end ASR is not as robust as hybrid ASR under a limited amount of training data, therefore the embedding vectors $\boldsymbol{e}_{1:L}^d$ are generated with the hybrid ASR transcriptions. It also allows fair comparison with the hard embedding extraction, which also makes use of the hybrid ASR transcripts.

negative predictions, the general formulation of F-score is defined as:

$$\text{Precision (P)} = \frac{\text{True positive (TP)}}{\text{True positive (TP)} + \text{False positive (FP)}} \tag{5.13}$$

$$\text{Recall (R)} = \frac{\text{True positive (TP)}}{\text{True positive (TP)} + \text{False negative (FN)}} \tag{5.14}$$

$$F_\beta = (1+\beta^2)\frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \text{Precision} + \text{Recall}} \tag{5.15}$$

where true positives are correctly identified positive predictions, false positives are incorrectly identified positive predictions, and similarly true negatives and false negatives stand for correctly and incorrectly identified negative instances. The positive class is conventionally used as the class of interest, and thus both precision and recall use the number of true positives as the numerator. The intuition behind is that precision measures the accuracy of the positive predictions, whereas recall measures the coverage of the positive predictions. F-score is a weighted mean of precision and recall, with a positive real factor β indicating the relative importance of recall compared with precision.

In disfluency detection, each word in the sequence is tagged as either fluent or disfluent, with the disfluent class defined as the positive class in the equations above. F_1 score, also known as the harmonic mean of the precision and recall, is conventionally used for transcription disfluency detection [234]. It can be easily computed for manual transcriptions annotated with disfluency labels, for example:

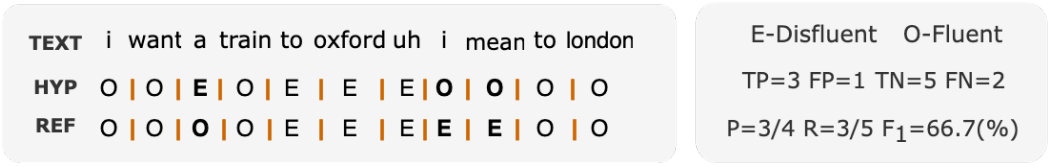


Fig. 5.4 An example evaluation of disfluency detection using F_1 score

However, it is not straight-forward to apply F_1 score to spoken disfluency detection, since there is no manual labelling available for the hypothesised transcriptions generated from the upstream ASR module. Two alternative metrics are adopted with different focuses: a modified F_1 score reflects the tagging accuracy, and an edit distance based word error rate (WER) assesses the quality of the output fluent sequences. To apply F_1 score on ASR transcripts, a set of reference disfluency labels are required for ASR transcriptions: the ASR transcripts are first aligned with the manual transcripts, and then the manual disfluency labels are mapped over onto the ASR transcripts. Figure 5.5 illustrates the pseudo tag generation process: after alignment, the matched (M) and substituted (S) words are tagged with the aligned label; the inserted (I) and deleted (D) words are excluded from the F_1 scoring.

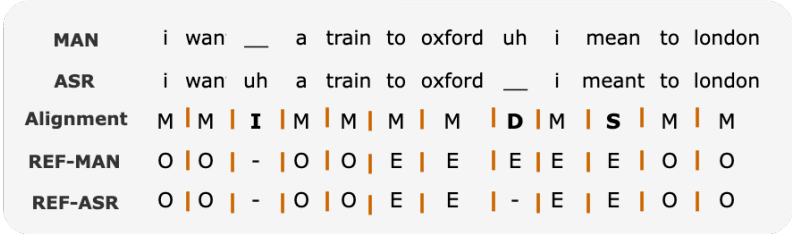


Fig. 5.5 Illustration of the pseudo reference tag generation on ASR transcriptions

Word error rate (WER) is a standard metric commonly used for speech recognition, which assesses the quality of the transcription hypotheses:

$$\text{WER} = \frac{\text{S+I+D}}{\text{M+S+I}} = \frac{\text{S+I+D}}{\text{Total \#words in hypothesis}} \tag{5.16}$$

Here, WER is used to assess the quality of the fluent sequence, which is an indication of how clean the sentences are post disfluency removal. The manual fluent sequence $w^f_{1:N}$ is used as the reference, and the hypothesised fluent sequence is generated by removing words tagged as disfluencies from the ASR transcripts $w_{1:L}$:

$$\hat{w}^f_{1:N} = \text{Removal}(\hat{w}_{1:L}, \hat{d}_{1:L}) \quad \rightarrow \text{WER}(w^f_{1:N}, \hat{w}^f_{1:N}) \tag{5.17}$$

5.3.2 Corpora

Switchboard [62] is used as the main corpus for the spoken disfluency detection task, following previous works on transcription disfluency detection. It consists of approximately 260 hours of telephone conversations of native English speakers, sampled at 8 kHz. There are multiple versions of transcription annotations in the speech community. The Treebank3 corpus [200] provides manual transcripts together with disfluency annotations. Switchboard-300 (Mississippi State transcriptions) [98] provides higher quality manual transcriptions than the Treebank3 corpus, and is commonly used for speech recognition. For spoken disfluency removal, the transcription quality and disfluency annotations are both essential. Therefore, in this work, the Switchboard disfluency annotation were obtained by aligning the Treebank-3 transcriptions with the Switchboard-300 corpus, and the disfluency annotations were mapped from Treebank-3 to Switchboard-300. The alignment was conducted at the per speaker level due to segmentation mismatches between the two corpora, and the utterance segmentations were recovered after the alignment process.

Switchboard provides data triplets of raw speech, manual transcriptions and disfluency annotations. There is few auxiliary corpora for text based disfluency detection, since dis-

fluency detection naturally operates on speech transcripts, i.e. written text does not contain disfluencies. Therefore, the experiments in this chapter were directly conducted on the end-to-end Switchboard corpus, without pre-training on data from auxiliary domains. For the individual modules and the embedding passing experiments, the same dev and test sets conventionally used for disfluency detection [99] were adopted. Table 5.1 summarises the statistics of the data splits used for the following experiments. Filler words (e.g. uh, huh) can be easily removed with rule based approaches, and were not counted as disfluencies for evaluation. All the results quoted in the following sections were evaluated on the Switchboard test set unless specified otherwise.

Split	#Sent.	#Words	%Disfluency
Train	173.9K	1.3M	5.70
Dev	10,172	86.4K	6.52
Test	8,039	65.8K	6.58

Table 5.1 Switchboard corpus statistics

5.3.3 Model training

Two spoken disfluency detection systems were trained: a cascaded system and an integrated system with embedding passing. The vocabulary used for model training was generated with the Switchboard corpus excluding rare words with less than two occurrences. All neural models were trained using the Adam optimiser [105] with a batch size of 256, and a learning rate of 0.001 with gradient clipping. Dropout was set at 0.2 if not specified.

For the cascaded system, a hybrid ASR module and a disfluency detection module were separately trained. The hybrid ASR module is a factorised time-delay neural network (TDNN-F) [166, 165] model with a trigram lattice generation, followed by rescoring with a 4-gram language model trained on the Fisher Corpus [37]. Word-level timestamps were generated alongside the ASR decoding. For manual transcriptions, timestamps were obtained through force alignment. The disfluency detection module consists of a 200D word embedding initialised using GloVe [161], a 2-layer 300D BLSTM followed by a binary classifier. The dropout rate was set at 0.5. Training was run for 50 epochs, with early stop if the performance on the development set stops improving for over 5 epochs. The performance of the individual modules in the cascaded system are listed in Table 5.2.

The integrated system was trained with both hard and soft acoustic embedding extractions. For the hard embedding extraction, a 2-layer 256D BLSTM was used to extract the

Module (Metric)	Result
ASR (WER ↓)	15.60
DD (F_1 ↑)	81.71

Table 5.2 Individual module (ASR and DD) performance evaluated on Switchboard test. The ASR module was evaluated against manual transcriptions as the reference, and the DD module was evaluated by feeding manual transcriptions as the module input.

bidirectional speech context, followed with a bilinear attention over per-word time windows. The hard embedding extraction layers were trained together with its downstream disfluency detection objective. For the soft embedding extraction, the alignment pipeline was initialised with an auxiliary attention-based encoder decoder (AED) ASR. Following the listen-attend-and-spell style model [27], the AED ASR was trained with three components: an acoustic encoder with a 1-layer 256D BLSTM and a 3-layer 256D pyramidal BLSTM (pBLSTM), a bilinear attention, and a decoder with a 4-layer 200D LSTM. After training with the ASR objective, the encoder and the attention mechanism combined were fixed, and used for the soft embedding extraction in the downstream disfluency detection training. The downstream disfluency detection module stays the same as the one in the cascaded system, with both word and acoustic embeddings as inputs. 40-dimensional filter banks were used as the acoustic features, and speaker level normalisation and spec augmentation [160] were both adopted.

5.3.4 Embedding passing

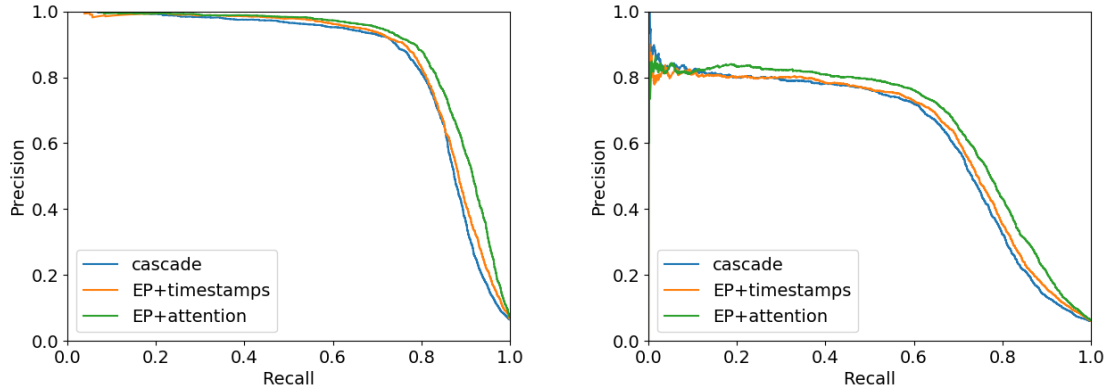
The baseline cascaded system is compared with the embedding passing (EP) approach discussed in Section 5.2 with both hard and soft acoustic feature extraction. Table 5.3 lists the system performance on both manual and ASR transcriptions, with the F_1 score analysing the disfluency tagging accuracy and WER assessing the quality of the output fluent sentences.

Compared with the cascaded system, adding acoustic information through embedding passing consistently reduced WER and improved F_1 scores on both manual and ASR transcriptions. Embedding passing gave slightly larger gain on F_1 scores of ASR transcripts compared with manual transcripts, which shows that acoustic information can potentially make up for transcription errors during tagging. Between the soft and hard embedding extraction approaches, the hard alignment using explicit timestamps yielded smaller gain compared with the soft alignment through the global attention trained with the speech recognition objective. The reason behind is that the timestamps generated from hybrid ASR decoding tend to cut-off sharply at the word boundaries, whereas the soft attention is more flexible

Models	MAN		ASR	
	$F_1 \uparrow$	WER $\downarrow$	$F_1 \uparrow$	WER $\downarrow$
Cascade	81.71	2.22	70.37	15.32
EP+timestamps	82.23	2.23	71.15	15.28
EP+attention	83.74	2.00	73.13	15.10

Table 5.3 Comparing vanilla cascade and embedding passing (EP) based spoken disfluency detection evaluated on the Switchboard test set. The disfluency detection operating threshold was chosen at 0.50. All models were trained on manual transcriptions, and F_1 / WER were evaluated against the manually annotated fluent transcriptions. MAN: hypothesised fluent sequence generated by feeding manual transcriptions as the DD module input; ASR: hypothesised fluent sequence generated with ASR transcriptions.

in attending to cross-over regions. Prosodic cues that are particularly informative to the disfluency detection task often lie across word boundaries, i.e. long pauses or hesitations tend to indicate disfluencies around their neighbouring regions. Figure 5.6 plots the precision recall curves of each system. It shows that the embedding passing approach with soft acoustic extraction achieved higher accuracies throughout the sweep over different thresholds, on both manual and ASR transcriptions.



(a) MAN (b) ASR

Fig. 5.6 Precision-recall curves of the vanilla cascade and the embedding passing (EP) systems, evaluated on manual and ASR transcriptions

5.4 Summary

This chapter investigated spoken disfluency detection as an example spoken language processing task. The main focus is to compare the cascaded system and the integrated system with embedding passing under the same amount of end-to-end training data. F_1 scores and WER were both adopted to measure the tagging accuracy and the output quality respectively. It is shown that extracting acoustic information via embedding passing effectively improved the downstream disfluency tagging accuracy as well as the output fluent sequences. As shown in Table 5.3, soft embedding passing with global attention worked better than hard embedding passing with timestamps, since the global attention accounts for the prosodic cues across word boundaries that are important for disfluency detection. The experiments confirmed that more tightly integrated systems with richer information passing across modules help mitigate the information loss issue in the cascaded formulation.

Chapter 6

Spoken Language Translation

This chapter takes spoken language translation (SLT) as an example spoken language processing task, and investigates the impact of various module combination approaches discussed in Chapter 4. Section 6.1 gives an overview of the task and discusses the cascaded SLT pipeline. Section 6.2 describes both supervised and semi-supervised error mitigation approach. Section 6.3 discusses an integrated SLT system with reranking, which takes into account rich information across modular connections via an external reranker. In Section 6.4, an embedding passing approach is proposed to encourage tighter modular connection in integrated SLT systems. Detailed evaluation metrics, experimental setups and analyses are reported in Section 6.5.

6.1 Task descriptions

Spoken language translation (SLT) is a translation task that converts speech in one language into text in a foreign language. It is a complex task that brings together both automatic speech recognition (ASR) and neural machine translation (NMT). The application areas include automatic video subtitling [179], simultaneous interpreting [56], and travel assistant [211] etc. End-to-end SLT corpora require speech utterances to be translated, and its high annotation costs lead to limited availability of speech translation data [195]. The central challenge is therefore to overcome the constraints on data availability, and leverage the modular nature to strike a balance between modelling power and data efficiency.

Traditional cascaded SLT systems directly concatenate an ASR module and an NMT module that are separately trained in their corresponding domains. Given the input speech $\boldsymbol{x}_{1:T}$, the speech transcript $w_{1:L}$ and the output translation $y_{1:N}$, the cascaded SLT process

(shown in Figure 6.1) can be formulated as:

$$\hat{w}_{1:L} = \text{argmax}_{w_{1:L} \in \mathcal{W}} P(w_{1:L} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}_{\text{ASR}}) \tag{6.1}$$

$$P(y_{1:N} | \boldsymbol{x}_{1:T}; \boldsymbol{\theta}) \approx P(y_{1:N} | \hat{w}_{1:L}; \boldsymbol{\theta}_{\text{NMT}}) \tag{6.2}$$

where the most likely hypothesis from the ASR module $\hat{w}_{1:L}$ is used as the input of the subsequent NMT module. As discussed in Chapter 4, traditional cascaded systems face challenges such as information loss, domain mismatches and erroneous early decisions.

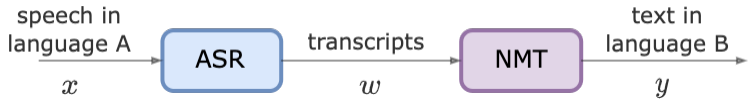


Fig. 6.1 Illustration of a cascaded spoken language translation system

Past literature has explored richer modular connections through N-best lists [206, 119] and lattices [182, 238]. Error propagation can be mitigated to some extent with an increasing level of complexity involved in the connection point. However, maintaining a rich search space for transcriptions is computationally expensive. With recent advances in attention-based encoder-decoder models, more work has been done on integrating ASR and NMT into a single model. Direct end-to-end model [9, 199] does not rely on intermediate speech recognition, but it requires significant quantities of in-domain, end-to-end data to reach good performance. Another line of approaches relies on explicit speech recognition, and adopts end-to-end trainable triangle structures [1, 194, 207]. The following sections will discuss various module combination approaches proposed in Section 4.2 and 4.3 under the context of multimodular spoken language translation systems. Error mitigation approaches directly operate on the cascaded structure, whereas the reranking and embedding passing approaches modify the system structure to allow richer information propagation across modules.

6.2 Error mitigation

Error propagation is one of the inherent deficiencies in cascaded multimodular systems. As discussed in Section 4.2.2, error mitigation is a straight-forward technique to tackle such issue without altering the cascaded structure. Given a cascaded SLT system with separately trained ASR and NMT modules, either supervised or semi-supervised error mitigation training can be adopted depending on the data availabilities. With end-to-end annotations containing the input speech and its corresponding translation $\{\boldsymbol{x}_{1:T}, y_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$[1], the NMT module

[1]The dataset notation follows the definition in Section 4.1.

can be trained using the ASR transcripts $\hat{w}_{1:L}$ from Equation 6.1 and the reference translation $y_{1:N}$:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{NMT}}^{\text{EM}}) = -\log P(y_{1:N}|\hat{w}_{1:L};\boldsymbol{\theta}_{\text{NMT}}^{\text{EM}}) \quad \{\boldsymbol{x}_{1:T}, y_{1:N}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y) \tag{6.3}$$

In cases where only speech data with its paired transcripts $\{\boldsymbol{x}_{1:T}, w_{1:L}\}$ are available from the target domain, the NMT module is trained with semi-supervised error mitigation:

$$y_{1:N}^{p} = \text{argmax}_{y_{1:N}\in\mathcal{Y}} P(y_{1:N}|w_{1:L};\boldsymbol{\theta}_{\text{NMT}}) \tag{6.4}$$

$$\mathcal{L}(\boldsymbol{\theta}_{\text{NMT}}^{\text{EMsemi}}) = -\log P(y_{1:N}^{p}|\hat{w}_{1:L};\boldsymbol{\theta}_{\text{NMT}}^{\text{EMsemi}}) \quad \{\boldsymbol{x}_{1:T}, w_{1:L}\} \in \mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w) \tag{6.5}$$

where the semi-supervision relies on the pseudo reference $y_{1:N}^{p}$ generated from $w_{1:L}$. Semi-supervised error mitigation familiarises the NMT module with ASR transcripts, and also indirectly exposes the NMT module under the target domain by training with generated pseudo references. Further semi-supervised self-distillation can be applied to guide the model with a richer distribution rather than the one-hot pseudo reference:

$$\mathcal{L}_{\text{KL}}(\boldsymbol{\theta}_{\text{NMT}}^{\text{EMdistil}}) = \sum_{n=2}^{N} \text{KL}\{P(y_{n}^{p}|y_{1:n-1}^{p}, \hat{w}_{1:L};\boldsymbol{\theta}_{\text{NMT}}^{\text{EMsemi}})||P(\boldsymbol{y}_{n}^{p}|y_{1:n-1}^{p}, \hat{w}_{1:L};\boldsymbol{\theta}_{\text{NMT}}^{\text{EMdistil}})\} \tag{6.6}$$

6.3 Reranking

To allow richer information propagation across modules, an external reranker can be adopted to directly access the transcription and translation hypotheses of the ASR and NMT modules. As introduced in Section 4.3.1.2, the reranker leverages the rich transcription search space, and then selects the best translation candidate among all. Without having to re-train the individual modules, reranking is especially useful when individual components of the cascaded pipeline are costly to build or to modify.

When feeding a sequence of speech input $\boldsymbol{x}_{1:T}$ into a cascaded SLT system, a transcription set $\mathcal{W} = \{\hat{w}_{1:L}^{(1)}, ..., \hat{w}_{1:L}^{(M_w)}\}$ and a corresponding translation set $\mathcal{Y} = \{\mathcal{Y}^{(1)}, ..., \mathcal{Y}^{(M_w)}\}$ with $\mathcal{Y}^{(i)} = \{\hat{y}_{1:N}^{(i,1)}, ..., \hat{y}_{1:N}^{(i,M_y)}\}$ can be obtained. For each translation hypothesis $\hat{y}_{1:N}^{(i,j)} \in \mathcal{Y}$, the reranker assigns a scalar score to indicate the quality of the candidate. The score is then normalised over the rest of the candidate scores, and the highest scoring candidate is chosen as the optimal output according to the reranker posterior:

$$\hat{y}_{1:N} = \text{argmax}_{y_{1:N}\in\mathcal{Y}} P(y_{1:N}|\mathcal{W};\boldsymbol{\theta}_{r}) \tag{6.7}$$

Figure 6.2 illustrates a reranking pipeline operating on the cascaded SLT system. In the centre of the reranker formulation, there are two design choices to be made: the metric μ (in Equation 4.30) used for the reference rank ordering generation; and the form of the feature extraction function f (in Equation 4.27). This section mainly addresses the design choices and practical consideration when applying reranking to the spoken language translation task.

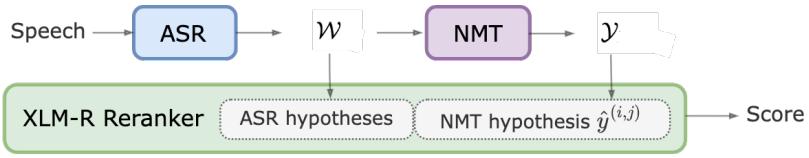


Fig. 6.2 Illustration of the SLT reranking pipeline

The rank ordering of the translation candidates is modelled as a score distribution indicating the probability of each hypothesis $\hat{y}_{1:N}^{(i,j)} \in \mathcal{Y}$ being the best candidate. The reference score is defined using BLEU, which is the standard metric used for speech translation evaluation. The utility function μ is therefore set to be BLEU, where the BLEU scores are computed between each translation hypothesis $\hat{y}_{1:N}^{(i,j)}$ and the reference translation $y_{1:N}$, and then converted into probabilities by normalising over the entire candidate set $\mathcal{Y}$.

The feature extraction function f captures the correlations between the transcription hypotheses and the translation candidate, and then generates a hidden representation used for score prediction, i.e. $f(\hat{y}_{1:N}^{(i,j)}$. Large pre-trained Transformer-based models are commonly used as feature extractors. XLM-R [40] (described in Section 2.4) is used here to initialise the bilingual feature extractor in the reranker, and then jointly fine-tuned with the reranking objective. The hidden state corresponding to the initial begin-of-sentence '<s>' token is used as the feature representation $\boldsymbol{h}$. Figure 6.3 shows a basic reranker structure, which only accounts for the paired transcription $\hat{w}_{1:L}^{(i)}$ and translation hypothesis $\hat{y}_{1:N}^{(i,j)} \in \mathcal{Y}^{(i)}$.

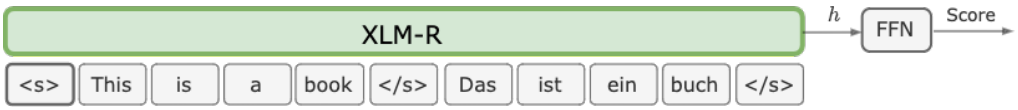


Fig. 6.3 Illustration of a basic SLT reranker with the XLM-R feature extractor [122]

The feature extraction process can also account for more than one transcription hypotheses. The same XLM-R structure can be adopted, with additional concatenation or attention mechanisms to propagate extra information from the ASR module. For the concatenation approach, multiple ASR hypotheses are concatenated via the end-of-sentence '</s>' token. For the attention approach, a basic dot product attention is adopted to attend over multiple feature vectors. The feature vectors are extracted by pairing different transcription hypotheses

with the same translation candidate to be scored, and they are further combined into a single hidden vector $\boldsymbol{h}$ using the attention weights. Figure 6.4 shows an illustration of the SLT reranker with an attention over multiple transcription to translation pairs.

Fig. 6.4 Illustration of the reranker with an attention mechanism over 3 ASR hypotheses paired with the translation candidate.

Another practical consideration is on the diversity of the translation candidates. The translation set $\mathcal{Y}$ to be ranked is usually generated using beam search in order to ensure high quality translations, and therefore the diversity among the candidates is often quite low. When the candidate diversity reduces to a certain level, the XLM-R feature extractor tends to neglect the differences, and consequently the reranker will fail to converge. Therefore, measures need to be taken to diversify the candidates without compromising the translation quality. Detailed diversification approaches will be introduced in the experiments section.

6.4 Embedding passing

Cascaded systems pass information through discrete words, which incur issues such as error propagation and loss of acoustic information. Vanilla end-to-end systems drop the notion of modules, and thus suffer from low data efficiency. As described in Section 4.3.2, the embedding passing approach aims to strike a balance between modelling power and data efficiency via tightly integrated modules:

$$P(y_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(y_{1:N}|\boldsymbol{e}_{1:L};\boldsymbol{\theta}^{\text{EP}}_{\text{NMT}}) \quad \boldsymbol{e}_{1:L} = f(\boldsymbol{x}_{1:T};\boldsymbol{\theta}_e) \tag{6.8}$$

where $\boldsymbol{\theta}^{\text{EP}}_{\text{NMT}}$ denotes the downstream translation module that takes embeddings $\boldsymbol{e}_{1:L}$ as inputs, and $\boldsymbol{\theta}_e$ parametrises the embedding extraction function. Figure 6.5 contrasts the three forms of the SLT systems with increasingly tighter integration: vanilla cascade, integration with embedding passing (EP), and end-to-end (E2E) systems. The ASR and NMT modules both adopt the attention-based encoder decoder (AED) formulation.

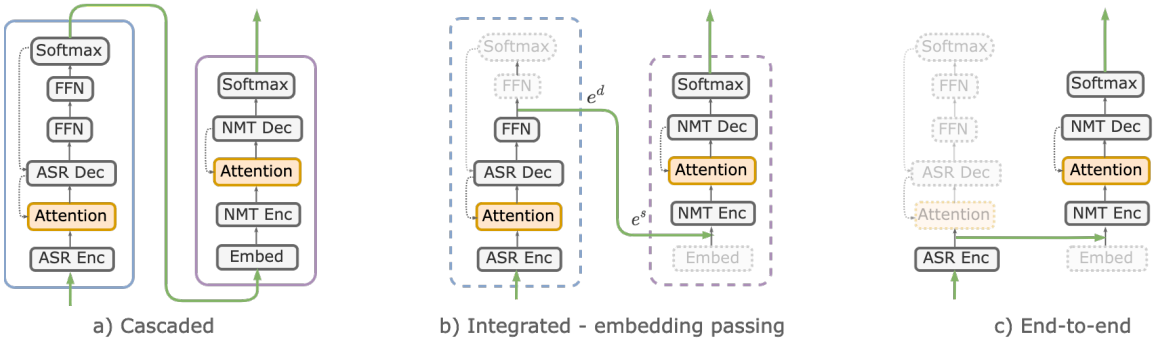


Fig. 6.5 Comparing cascade, embedding passing integration, and end-to-end SLT systems. The active components are coloured and the inactive components are shaded.

For the embedding passing SLT system, an attention-based soft embedding extraction is adopted, as opposed to the hard embedding extraction with timestamps. The global attention allows more flexibility in attending over distance context, which provides richer information for the downstream translation module. The acoustically derived dynamic embeddings $\boldsymbol{e}^d_{1:L}$ are matched with the static word embeddings $\boldsymbol{e}^s_{1:L}$, which are used as the modular connection:

$$\boldsymbol{e}_{1:L} = \boldsymbol{e}^s_{1:L} = \boldsymbol{e}^d_{1:L} \tag{6.9}$$

Figure 6.5b shows an illustration of the integrated SLT system with embedding passing. The embedding connection maintains a rich information flow, and is particularly useful during the in-domain training with end-to-end speech translation data.

Static embeddings are derived using a one-to-one mapping from words, whereas dynamic embeddings follow a many-to-one mapping since the same word can be pronounced in different ways. The embedding matching process reaches a compromise between richness from acoustics and robustness from transcripts. However, the many-to-one nature of the dynamic embeddings causes a systematic mismatch from the static embeddings. To strike a balance between rich information and regularisation through words, joint embedding passing (EP-J) can be adopted, which uses both embeddings as the modular connection:

$$\boldsymbol{e}_{1:L} = \boldsymbol{W}[\boldsymbol{e}^s_{1:L}, \boldsymbol{e}^d_{1:L}] \tag{6.10}$$

where $\boldsymbol{W}$ is a transformation matrix setting the dimension of the joint embedding. EP-J aims to loosen the constraints of embedding matching by decoupling the static and dynamic embeddings, and simply concatenates the two to yield a joint embedding for translation. Figure 6.6a shows an illustration of the integrated SLT system with joint embedding passing.

The downstream translation module can still be initialised with auxiliary translation corpora by fixing $\boldsymbol{e}^{d}_{1:L}$ to be an averaged dynamic embedding derived from the auxiliary ASR corpora.

Under a limited amount of training data, AED ASR is not as robust as hybrid ASR. Embedding passing models, which rely on the transcripts from the AED ASR, tend to suffer from poor speech recognition, and consequently lead to inferior speech translation quality. Compared with direct end-to-end models, one of the advantages of having an explicit speech recognition module in integrated systems is the flexibility of incorporating external sources of information. In embedding passing systems, it is possible to modify the AED ASR generated hypotheses during inference, and propagate its impact through the ASR attention with the modified back history. One way of improving the transcriptions is via shallow fusion, i.e. decode the AED ASR module with an external language model. Another more effective approach is to directly replace the AED generated ASR hypotheses with higher quality transcriptions. Figure 6.6b illustrates the process of incorporating external transcripts to the joint embedding passing system. The predicted transcription history $w_{1:l-1}$ is replaced with a higher quality transcription history $w'_{1:l-1}$, which indirectly improves the static and dynamic embeddings to be passed on to the downstream translation module.

Fig. 6.6 Joint embedding passing system: decode without / with external transcripts

6.5 Experiments

The experiments in this chapter mainly analyse the impact of the module combination approaches discussed from Section 6.2 to 6.4. This section first discusses the evaluation metrics and the corpora of the spoken language translation task, followed with the details on model training. Three module combination approaches are then analysed, namely error mitigation, reranking and embedding passing.

6.5.1 Evaluation metrics

Bilingual Evaluation Understudy Score (BLEU) [158] is an edit based evaluation metric conventionally used for machine translation tasks. BLEU analyses the translation quality by considering the distance between the machine predicted output and the gold standard manual annotations. Given a translation candidate C with a reference R, the score is calculated as the geometric average over the n-gram matching precision over the entire test corpus:

$$\text{BLEU}(C,R) = \text{BP} \cdot \exp(\textstyle\sum_{n=1}^{N} w_n \log p_n) \tag{6.11}$$

$$\text{BP} = \min(1, e^{1-r/c}) \tag{6.12}$$

$$p_n = \frac{\sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}(R)}(\text{n-gram})}{\sum_{\text{n-gram}' \in C} \text{Count}(\text{n-gram}')} \tag{6.13}$$

where p_n is the n-gram precision, w_n is the weight coefficient, and BP is the brevity factor with c being the candidate translation length and r being the reference length. $\text{Count}_{\text{clip}(R)}$ clips the total count of each candidate n-gram by its maximum count in reference R. This thesis adopts the conventional combination where $n = 4$ with uniform weights $w_n = 1/4$. Another line of metrics adopts large language models such as BERT to measures the semantic similarity between the generated text and the reference text, e.g. BERTScore [239]. In this thesis, BLEU score is adopted to be consistent with past literature on the SLT task.

Another important aspect is the diversity of the generated candidates. A translation model can produce a group of hypotheses via beam search or sampling during decoding. Cross-BLEU [188] measures candidate diversity by averaging over the pair-wise BLEUs. Given a group of hypotheses $\{\hat{y}^{(m)}\}_{m=1}^{M}$, cross-BLEU can be calculated as:

$$\text{crossBLEU} = \tfrac{1}{M(M-1)} \textstyle\sum_{m_1=1}^{M} \sum_{m_2=1}^{M} \text{BLEU}(\hat{y}^{(m_1)}, \hat{y}^{(m_2)})_{m_1 \neq m_2} \tag{6.14}$$

Lower cross-BLEU indicates higher diversity in the hypotheses set. For a translation model, it is usually ideal to have a high quality and diverse hypotheses set. Both BLEU and cross-BLEU scores can be directly applied to the spoken translation task, since they do not rely on the transcriptions and only compare the output translation sequences.

6.5.2 Corpora

The experiments were conducted on the English-to-German (En-De) speech translation task. Three main corpora used here are summarised in Table 6.1. **MuST-C** [47] is a multilingual speech translation corpus translating English TED Talks into 8 different languages. The En-De direction is used as the target task in this thesis. The corpus provides data triplets

consisting of speech, its corresponding English transcription and German translation. **TED-LIUM3** [78] is collected from TED conference videos and dedicated to speech recognition tasks in English. It is used in this work to initialise the ASR module. **WMT17** [19] refers to a public release of machine translation shared task, and the En-De subset is adopted for the speech translation task. WMT17 En-De has over 4M sentences in total, but only 10% is used here for pre-training purposes to save computational time. All results are evaluated on the MuST-C tst-COMMON test set with case-sensitive BLEU.

On the audio side, 40 dimensional filter bank features were extracted at 10ms frame rate. For text pre-processing, English transcriptions were lower-cased, punctuation-normalised using Moses toolkit [108], and German translations were kept punctuated and true-cased. Byte Pair Encoding (BPE) [77] tokenisation was adopted for the English transcripts, which splits words into subword units and helps alleviate the issue of out-of-vocabulary words.

Corpus	Domain	#Hour	#Sent.	#Words
MuST-C En-De (MuSTC)	SLT - $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$	408h	229.7K	4.3M
TED-LIUM3 (TED)	ASR - $\mathcal{D}_{\text{up}}(\boldsymbol{x}, w)$	452h	268.3K	4.5M
WMT17-P En-De (WMT)	NMT - $\mathcal{D}_{\text{dn}}(w, y)$	-	454.4K	11.5M

Table 6.1 Corpora used for training spoken language translation systems. Word count is calculated on the source English text.

6.5.3 Model training

There are a total of four individual models being separately trained[2]. For the upstream ASR module, an attention-based encoder decoder (AED) ASR and a hybrid ASR model were trained. For the downstream NMT module, a vanilla Transformer-based NMT model was trained from scratch, and a T5-based pre-trained NMT model was also adopted. The error mitigation and reranker experiments were conducted on the cascaded system consisting of the hybrid ASR and the T5-based NMT. The embedding passing experiment adopted the concatenation of the AED ASR and the Transformer-based NMT as the baseline. It further investigated the data efficiency of different forms of modular integration. All models were trained using Adam optimiser [105] with a batch size of 256, dropout 0.2, and a learning rate of 0.001 with gradient clipping. The final model parameters were calculated using checkpoint averaging [94], which took the average over the 5 best checkpoints.

[2]Different combinations of the individual models were adopted for different experiments, since some of the more advanced models were introduced during the course of the research development.

The AED-based ASR model has an encoder of 1x256D bidirectional LSTM and 3x256D pyramidal bidirectional LSTM layers, reducing the acoustic sequence lengths by 8. The decoder then uses bilinear attention, followed by 3x512D unidirectional LSTM layers. Speaker level normalisation and spec augmentation [160] were enabled during training. The target transcriptions were tokenised using Byte Pair Encoding (BPE) [189] with a 40K vocabulary trained on transcriptions of MuSTC. The hybrid ASR model adopts a lattice-free maximum mutual information (LF-MMI) factorised time-delay neural network (TDNN-F) acoustic model [166, 165] followed by a trigram decoding. The decoding language model (LM) was trained on TED, and the LM scale factor was set at 10. Table 6.2 contrasts the RNN-based AED and hybrid ASR modules trained on the in-domain MuSTC corpus. Under limited training data, the hybrid ASR model achieved much lower WER compared with the recurrent network based AED ASR model[3].

Module (Metric)	Model	Result
ASR (WER↓)	RNN AED	20.99
	Hybrid	7.32
	Transformer AED from corpus paper [47]	27.00
	Transformer AED from ESPnet [212]	12.70
	Transformer AED from ESPnet-ST [92]	7.00
	Conformer AED from ESPnet-ST [92]	5.60
NMT (BLEU↑)	Transformer	27.07
	T5	31.55
	Transformer from corpus paper [47]	28.09
	Transformer from ESPnet [212]	30.16
	Transformer from ESPnet-ST [92]	35.50

Table 6.2 Individual model performances trained on MuSTC, evaluated on MuSTC tst-COMMON split. Manual transcripts were used both for NMT training and evaluation. For the ASR module, the RNN AED ASR is compared with the hybrid ASR, as well as the literature. For the NMT module, the vanilla Transformer NMT trained from scratch is compared with T5-based pre-trained NMT, as well as the literature.

The Transformer NMT is a standard base-sized model with 512D hidden states, 6 encoder layers, and 6 decoder layers. The translation source sequences adopted the same BPE tokenisation that is consistent with the vocabulary used for the AED-based ASR, and the

[3]This work was done before the current state-of-the-art Conformer-based ASR [70], which achieves a much lower WER compared with the recurrent network based AED ASR.

target sequences were tokenised into character level units. The T5-based NMT [174] follows the standard Transformer structure, but is pre-trained with large quantities of data under a unified multitask framework. The Huggingface pre-trained T5 implementation[4] was adopted for this work, and the model was further fine-tuned to the in-domain MuSTC data. Table 6.2 contrasts the vanilla Transformer and the T5 NMT models, both trained on the in-domain MuSTC corpus. Translation decoding was conducted via greedy search unless specified otherwise. The T5-based pre-trained NMT model outperformed the vanilla Transformer model trained from scratch by 4.5 BLEU scores.

In Table 6.2, the ASR and NMT performances were also compared with results in literature. The original paper [47] that describes the corpus were quoted, as well as the results from the ESPnet toolkit [212] and the more recent ESPnet-ST paper [92][5]. The ASR performance of ESPnet-ST improved significantly from the original paper and ESTnet, and there is also a huge gap between our RNN-based AED to ESPnet-ST's Transformer and Conformer based AED. This may be due to different choices of tokenisation, audio pre-processing, model structure, as well as more datasets being used for ESPnet-ST training. The NMT performance from ESPnet-ST beats the rest, which may be benefited from the significantly larger amount of training corpora. It also beats the T5 model fine-tuned with MuST-C dataset. This may be because they also used TED training data, which is in a very similar domain as the target domain MuSTC set. The focus of this work is to explore the integration of the ASR and NMT modules, and thus no further effort was made to improve the individual module performances.

6.5.4 Error mitigation

This section analyses the impact of the error mitigation approaches discussed in Section 6.2. The cascaded system is composed of the hybrid ASR and the T5-based NMT modules: the ASR module was trained on the in-domain MuSTC data and fixed from further update; the downstream NMT module was initialised with the pre-trained T5 model. Various error mitigation approaches were adopted to adapt the NMT, and the results are listed in Table 6.3.

The baseline model (Base) is the pre-trained T5 model without in-domain training. Under partially annotated data $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w)$, where there is only speech with its paired manual transcripts from the target MuSTC domain, semi-supervised error mitigation (EMsemi) trained the NMT module with the ASR transcripts as the inputs, and used the manual

[4]https://huggingface.co/t5-base

[5]Our work discussed in this thesis was conducted in between the developemnt of the ESPnet and the ESPnet-ST systems.

Model	In-domain Data	BLEU ↑
Base	-	20.65
EMsemi	$\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w)$	22.11
EMdistil	$\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w)$	22.41
EM	$\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$	28.89

Table 6.3 The impact of error mitigation on the NMT module, with different availabilities of data pairs under the target domain. BLEU scores were computed against the manual translations, and the NMT module inputs were ASR transcripts. The BLEU scores therefore directly reflect the speech translation performance.

transcripts translated hypotheses as the pseudo reference. Semi-supervised self-distillation (EMdistil) further distiled the EMsemi model, by setting the EMsemi model as both the teacher and the student. Under fully annotated in-domain data, where there is speech paired with its corresponding translations, the NMT module can be directly fine-tuned under manual supervision (EM). Semi-supervised error mitigation improved the speech translation BLEU by 1.46, which confirms that familiarising the NMT module with in-domain ASR transcripts effectively mitigates ASR error propagation, and semi-supervised self-distillation further improved the speech translation performance to 22.41. Under end-to-end data, supervised error mitigation gave the largest gain, pushing the speech translation performance to 28.89.

α_{KL}	BLEU ↑
0.0	22.11
0.5	22.21
1.0	22.41

Table 6.4 The impact of adding negative log likelihood loss to the semi-supervised self-distillation training. The KL coefficient α_{KL} is the weight of the KL divergence loss, and the weight of the likelihood loss is (1.0-α_{KL}).

A common practice in self-distillation is to combine the KL divergence loss (Equation 6.6) with the standard negative log likelihood loss (Equation 6.5). It aims to reach a balance between learning the distribution and learning the one-hot prediction. Table 6.4 shows the impact of incorporating the log likelihood objective with different KL loss coefficients. Setting the KL coefficient α_{KL} to 0.0 turns off the KL loss and the model falls back to the semi-supervised EMsemi model; setting α_{KL} to 1.0 effectively turns off the log likelihood objective and the model becomes the fully distilled EMdistil model; and setting α_{KL} to 0.5

weighs the two losses equally during training. It is shown that adding the log likelihood loss degraded the translation performance. This confirms that under semi-supervised scenario where the quality of the pseudo reference is not good enough, the KL loss over the distribution leaves the model with more freedom to explore and reach a better candidate.

6.5.5 Reranking

This section investigates the reranking approach discussed in Section 6.3. The baseline system is a cascaded concatenation of the hybrid ASR and the T5-based NMT, both trained on the in-domain MuSTC data[6]. The following sections will discuss the candidate generation process from the ASR and NMT modules, followed with discussions on reranker training and further analyses on the reranker performance.

Candidate generation

In modular systems, if each individual module generates a fixed number of candidates for each input, the total number of the candidates increases exponentially with the number of the modules. With a larger search space and more candidates to choose from, it is more likely that the best candidate in the hypotheses set gets closer to the ground truth translation. However, it is computationally expensive to keep expanding the search space. Table 6.5 contrasts the impact of branching out the search spaces with varying sizes for the ASR and NMT modules. To compare across different candidate sets, several non-neural based ranking approaches are used in the table: oracle ranking directly uses BLEU scores calculated against the manual translation to select the best candidate, which sets the upper bound of any reranking algorithm; P_{ASR}, P_{NMT}, S_{comb} respectively ranks the translation hypotheses according to the sentence level confidence scores using the ASR posterior, NMT posterior, and a combination of both the ASR and NMT posteriors. When calculating S_{comb}, since the dynamic range of P_{ASR} and P_{NMT} are different, the ASR confidence was first softened using a temperature factor and then combined with the NMT posterior. The optimal temperature factor and P_{ASR} weighting were chosen as 30 and 0.2 respectively.

To ensure fair comparison, the total number of the translation candidates was fixed at 50. Keeping the 5-best ASR candidates and then branching out to 10 NMT hypotheses per ASR candidate ($N_a = 5$, $N_n = 10$) led to the highest BLEU score of 30.29 with the probabilistic ranker S_{comb}. For the following experiments, this setup was adopted for candidate generation, and 30.29 is considered as the baseline reranker performance. Further expanding the search space to a total of 2500 candidates with ($N_a = 50$, $N_n = 50$) gave an oracle performance of

[6]It is not required that the ASR and NMT modules are trained on the in-domain corpora, and the reranking approach applies to any pre-trained cascaded SLT system that is able to generate reasonable hypotheses.

Cand. Generation		Ranker (BLEU↑)			
ASR (N_a)	NMT (N_n)	Oracle	P_{ASR}	P_{NMT}	S_{comb}
1	1	28.89	28.89	28.89	28.89
1	50	42.09	29.95	29.95	29.95
5	10	40.26	29.29	30.14	**30.29**
10	5	39.51	28.59	30.01	30.12
50	1	37.50	27.34	28.89	29.05
50	50	49.30	27.60	29.95	29.96

Table 6.5 Speech translation performance with varying size search spaces for the ASR and NMT modules. ASR: keep the top N_a ASR hypotheses; NMT: for each ASR hypothesis, the NMT decoding generates N_n candidates with a beam width of N_n. For each speech input, there are a total of $N_a N_n$ candidates.

49.30, yet the top candidate selected using the probabilistic rankers did not reach higher scores in spite of the better oracle candidate. This is due to the mismatch between the posterior confidence and the reference BLEU score of the candidates. One of the advantages of adopting an external reranker is to address such mismatch between the model predicted posterior and the true distribution of the scores.

Sampling	aveBLEU	crossBLEU
✗	17.21	43.49
✓	18.33	30.58

Table 6.6 The average BLEU and cross BLEU scores of the candidate set ($N_a = 5$, $N_n = 10$) before and after the diversification process with sampling.

Translations generated via beam search are not as diverse as those generated with sampling [84]. In order for the neural based rerankers to train, the candidate set needs to reach a certain level of diversity. Selective sampling can be used to diversify the hypothesis set without compromising the translation quality: a set of candidates is first generated with beam search; the top half are selected according to the reference BLEU scores, and the other half are replaced with candidates generated using sampling. Table 6.6 compares the average BLEU (aveBLEU) and crossBLEU scores before and after the diversification process, where the chosen candidate set ($N_a = 5$, $N_n = 10$) is described in Table 6.5. It is shown that adding sampling reduced crossBLEU by 12.91, with the average translation BLEU improving by

1.12. Candidate diversification is only required for training, and the trained reranker can be used to rank the full candidate set generated from beam search during inference.

Reranker training and evaluation

With the translation candidates generated as described, the neural based rerankers can be trained on the MuSTC data. The rerankers extract correlations between the ASR transcripts and the translation hypotheses, and predict a score for each hypothesis. The pre-trained XLM-R model was adopted for feature extraction, and different neural rerankers were trained with an increasing number of ASR hypotheses and different scoring functions. The vanilla scoring function operated on a matched translation pair $\{w_{1:L}^{(i)}, y_{1:N}^{(i,j)}\}$. To take into account multiple ASR hypotheses, a concatenation based and an attention-based feature extraction functions were adopted. Table 6.7 compares the reranker performance with different feature extraction functions. All reranker posteriors were interpolated with the ASR and NMT posteriors for the final results. The baseline ranker (Base) selects the best hypotheses according to the combined posterior of ASR and NMT module, and the Oracle ranker uses the reference translation score to choose the best candidate, which sets the upper-bound for the rerankers.

Ranker	N_{ASR}	Mode	BLEU↑
Base	-	-	30.29
Oracle	-	-	40.26
XLM-R	1	-	31.26
	2	cat	31.23
	2	att	31.47
	3	cat	31.37
	3	att	**31.50**
Corpus paper (2019) [47]			18.50
ESPnet (2020) [212]			22.91
KIT (2020) [201]			30.60
RWTH (2021) [5]			26.50
ESPnet-ST (2021) [92]			35.50

Table 6.7 SLT performance with neural reranking. N_{ASR}: the top-N_{ASR} ASR hypotheses taken into account for feature extraction; Mode: approaches to incorporate multiple ASR hypotheses into feature representation - cat: concatenation, att: attention. Base and Oracle performances are quoting the $N_a = 5, N_n = 10$ case listed in Table 6.5. The model performances from the literature were also included for reference. The best performing reranker reaches 31.50 BLEU with an attention over $N_{\text{ASR}} = 3$. It outperforms most recent works, yet still lagging compared with ESPnet-ST.

As shown in Table 6.7, the vanilla neural ranker operating on the matched translation pair improved the translation performance to 31.26 from the baseline 30.29. This verifies that directly optimising the reranker to learn the reference score distribution helps adjust the model posterior to better align with the evaluation metric. Further incorporating more ASR hypotheses gave additional gains. The attention combination was more effective than simply concatenating the ASR transcripts, since the attention mechanism is more flexible in assigning larger weights to the most relevant context during feature combination. The best performing reranker reaches 31.50 with an attention feature combination over $N_{\text{ASR}} = 3$, outperforming the baseline by 1.21. This confirms that adopting an external reranker to access multiple ASR hypotheses is an effective approach of addressing information loss and error mitigation issues in cascaded SLT systems. However, it is not feasible to keep increasing the number of ASR hypotheses being used. Running large pre-trained models, such as XLM-R, on long sequences requires large memory usage, and the limiting factor is usually the GPU memory.

6.5.6 Embedding passing

This section investigates integrated systems with embedding passing as discussed in Section 6.4. Three SLT systems are considered here with an increasing level of integration: the vanilla cascaded (Casc) system is connected through discrete word sequences, the integrated system is connected through word-level continuous embeddings (EP), and the direct end-to-end (E2E) system directly uses acoustic-level hidden states for downstream translation. The system development details are first discussed, and the analyses mainly focus on the following aspects: the impact of different levels of in-domain data availability, data efficiency, as well as the output translation performance.

System development

The cascaded system is a simple concatenation of the separately trained ASR and NMT modules. Table 6.8 contrasts the AED and hybrid ASR models, and their corresponding cascaded SLT performance when connected with the downstream NMT module. For the ASR module, the AED and hybrid models were respectively trained on the out-of-domain TED corpus $\mathcal{D}_{\text{up}}(\boldsymbol{x}, w)$, and the MuSTC corpus from the target domain $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w)$. The NMT module adopts a standard Transformer structure. The baseline model (Base) was trained on the out-of-domain WMT corpus $\mathcal{D}_{\text{dn}}(w, y)$, and the fine-tuned model (Tuned) was further trained on the MuSTC data from the target domain $\mathcal{D}_{\text{tgt}}(w, y)$. The hybrid ASR model largely outperformed AED ASR, and they consequently led to improved speech translation

performance when connected with the NMT models. The highest BLEU score 25.45 was achieved by connecting the in-domain trained hybrid ASR model with the Tuned NMT.

	ASR		NMT	
Model	Corpus	WER↓	Base	Tuned
Hybrid	TED	10.58	13.58	23.85
	MuSTC	7.32	14.41	**25.45**
AED	TED	35.20	9.60	15.97
	MuSTC	20.99	12.25	20.54

Table 6.8 The cascaded SLT system performance with different combinations of ASR and NMT models (BLEU ↑). Base: the baseline NMT trained on WMT from $\mathcal{D}_{\text{dn}}(w,y)$, Tuned: NMT trained on MuSTC from the target domain $\mathcal{D}_{\text{tgt}}(w,y)$. All results were evaluated on the MuSTC tst-COMMON set.

In the direct E2E system, the acoustic encoder adopts a similar structure as the AED ASR, which consists of 1x256D bidirectional LSTM and 3x256D pyramidal bidirectional LSTM layers. The translation decoder is a standard Transformer decoder but with acoustic level hidden states as the inputs. The E2E training requires end-to-end data from the target domain $\mathcal{D}_{\text{tgt}}(\boldsymbol{x},y)$, yet it is difficult for the E2E system to converge under limited quantities of data. Therefore, the acoustic encoder was initialised using the auxiliary speech recognition objective with $\mathcal{D}_{\text{tgt}}(\boldsymbol{x},w)$, and then fixed when trained for the translation objective.

As discussed in Section 6.4, two integrated systems were developed: an embedding passing (EP) system that matches the static and dynamic embeddings, and a joint embedding passing (EP-J) system that passes both the static and dynamic embeddings. Both the static and dynamic embeddings were set to be 512D. For both EP and EP-J, the upstream AED ASR module was initialised with the auxiliary ASR corpus from $\mathcal{D}_{\text{up}}(\boldsymbol{x},w)$, and the downstream NMT module was initialised with the auxiliary NMT data from $\mathcal{D}_{\text{dn}}(w,y)$. At the in-domain training stage, all parameters were fine-tuned with the SLT data: if the target domain corpus only contains speech with its paired translation $\mathcal{D}_{\text{tgt}}(\boldsymbol{x},y)$, the acoustic encoder is fixed, and the translation objective is imposed during fine-tuning; when manual transcriptions are available in $\mathcal{D}_{\text{tgt}}(\boldsymbol{x},w,y)$, fine-tuning adopts both the auxiliary ASR objective and the SLT objective. All results in this section were decoded with a beam width of 5.

In-domain data availability

This section compares the SLT systems under different levels of in-domain data availability. The key difference that distinguishes the SLT systems is the information flow passed from

the acoustic side to the translation side. Cascade uses discrete words, EP uses continuous embeddings, and E2E uses acoustic level hidden states. To directly contrast the impact of different information flows, AED ASR was used across all systems, and hybrid ASR will be considered in later sections. Three levels of data availabilities were considered: (1) no in-domain data - None; (2) in-domain speech paired with its translation, without manual transcription - $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$; (3) in-domain data triplet containing speech, transcription and translation - $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$. The Cascade, EP and EP-J systems were initialised under the auxiliary ASR domain $\mathcal{D}_{\text{up}}(\boldsymbol{x}, w)$ and NMT domain $\mathcal{D}_{\text{dn}}(w, y)$, and further fine-tuned under the target SLT domain[7], whereas the E2E system was directly trained under the target domain.

Models	None	$\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$	$\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$
E2E	-	-	19.29
Cascade	9.60	16.56	20.54
EP	7.97	**18.84**	22.56
EP-J	9.60	16.24	**23.25**

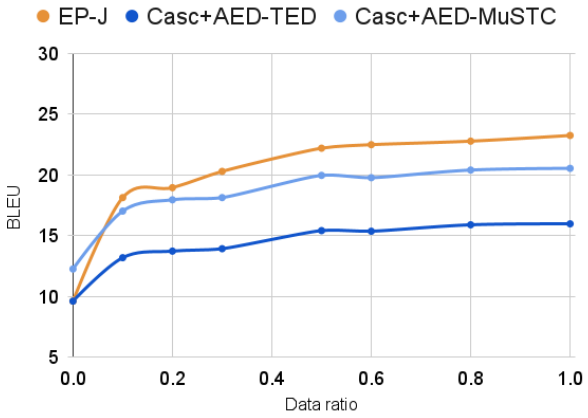
Table 6.9 Comparing the Cascade, EP, EP-J and E2E systems under three levels of in-domain data availabilities (BLEU↑). None: no in-domain data; $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$: in-domain speech paired with its translation, without manual transcription; $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$: in-domain data triplet containing speech, transcription and translation.

Table 6.9 shows that Cascade and EP-J performed similarly under zero in-domain data. EP-J performed the best with fully annotated target domain data $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$, whereas EP performed the best in cases when manual transcriptions are not available in $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$. When there is no in-domain data, EP fell short mainly because of the imperfect matching between the static and dynamic embeddings, and embedding mismatch is a systematic issue before any in-domain training takes place. In comparison, EP-J loosens the restriction and makes use of both the dynamic and static embeddings. The downstream NMT module in EP-J was initialised using the averaged dynamic embeddings over all training instances. Before in-domain training (None), the EP-J system was not better than Cascade, since it is yet to benefit from the rich acoustic context. After training with fully annotated target domain data $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$, EP-J achieved the best performance among all. It obtained robustness through regularisation provided by the static embeddings, while retaining richness of the

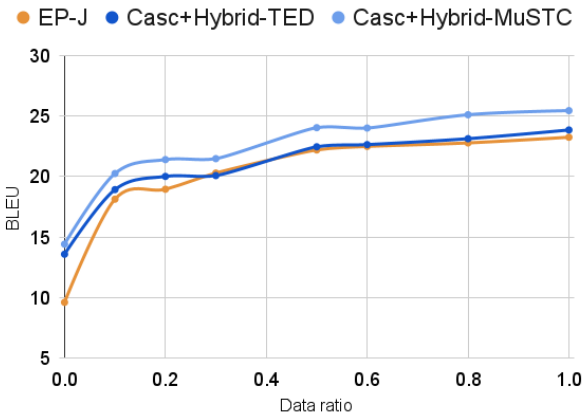
[7]For the ASR module in the Cascade system, the TED trained AED was adopted for condition (1) and (2), and the MuSTC trained AED was adopted for condition (3). When fine-tuning the NMT module in the Cascade system, the AED-TED model predicted transcripts were adopted for condition (2).

acoustic features from the dynamic embeddings. However, when manual transcriptions are not available in $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, y)$, i.e. the ASR errors were passed down to the NMT module during in-domain training, EP outperformed Cascade and EP-J by over 2 BLEU points. This confirms that having a soft dynamic modular connection does help mitigate the error propagation issue. In the following sections, the focus is laid on comparing the Cascade and EP-J systems, since they are the most competitive systems when trained with fully annotated SLT corpus.

Data efficiency



(a) Cascaded system with AED ASR versus EP-J system



(b) Cascaded system with Hybrid ASR versus EP-J system

Fig. 6.7 Data efficiency: BLEU↑ versus Data ratio (from left to right: systems were fine-tuned with an increasing amount of end-to-end SLT data using manual transcriptions)

The previous section analysed two ends of the data spectrum, with either none or fully end-to-end SLT data. This section further explores how the Cascade and EP-J systems behave under different quantities of target domain data $\mathcal{D}_{\text{tgt}}(\boldsymbol{x}, w, y)$. To analyse the data efficiency, both systems were initialised with the ASR corpus $\mathcal{D}_{\text{up}}(\boldsymbol{x}, w)$ and NMT corpus $\mathcal{D}_{\text{dn}}(w, y)$, and then fine-tuned with a sweep through the in-domain SLT data (from 0% to 100%). Their respective performances were recorded for each data ratio. Figure 6.7a compares the Cascade and EP-J systems with the AED ASR module. For Cascade, both AED-TED and AED-MuSTC transcriptions were used, setting the lower bound (out-of-domain trained AED) and the upper bound (in-domain trained AED) respectively. EP-J and Cascade behaved similarly in the zero data region. With an increasing amount of in-domain SLT data, EP-J adapted to the target domain much more quickly, starting to outperform the Cascade upper bound with only 10% of end-to-end SLT data. The main difference between Cascade and EP-J is the additional dynamic embeddings used at the modular connection. The observation confirms that dynamic embeddings do provide downstream modules with a richer acoustic context, and thus allow more efficient domain adaptation.

However, it is not fair to only compare EP-J with Cascade systems that use the inferior AED transcriptions. One of the benefits of cascaded structures is the flexibility in improving each individual module. To construct a stronger cascaded baseline, two hybrid ASR models were used to provide better speech transcriptions: Hybrid-TED and Hybrid-MuSTC respectively set the lower and upper bounds. Figure 6.7b shows that when the Cascade system adopts hybrid ASR transcriptions, its lower bound achieved similar performances as the EP-J system, and the upper-bound outperformed EP-J. The inferior performance of EP-J is mainly due to the performance gap between the AED and the hybrid ASR.

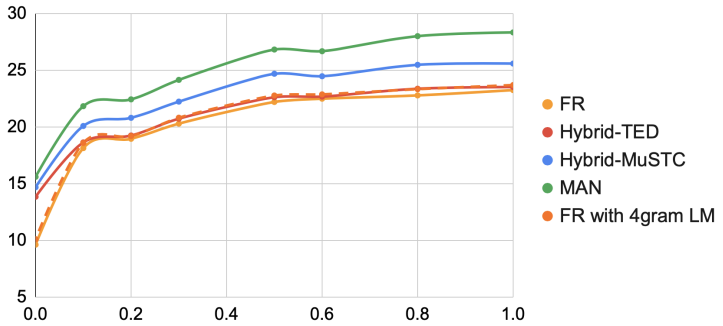
Improved AED back history

As discussed above, the EP-J system suffered from poor AED performance, and consequently led to inferior speech translation quality. With an explicit speech recognition module, the EP-J system is able to incorporate external sources of information as discussed in Section 6.4. The AED generated hypotheses can be improved via two approaches: (1) adding an external language model with shallow fusion - here a 4-gram language model trained on TED was adopted; (2) directly replace the AED hypotheses with external ASR transcriptions - here the hybrid ASR transcripts were used. Figure 6.8 shows the impact of incorporating external information, with the same sweep through the data ratio as in the previous section.

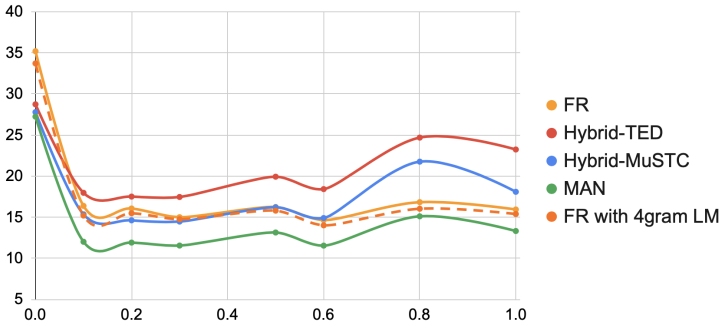
In general, adopting a better AED back history led to lower AED WER, and higher SLT BLEU scores[8]. Shallow fusion with a 4-gram language model reduced AED WER, and led

[8]Figure 6.8b shows some mismatches between the back history quality and the AED performance. The back history quality follows the order of FR < Hybrid-TED < Hybrid-MuSTC < MAN, yet the generated AED

to slightly improved BLEU scores (dotted orange lines). Adopting the higher quality hybrid ASR transcripts gave more significant gains (blue lines). With the in-domain Hybrid-MuSTC transcripts, EP-J improved by around 2 BLEU points throughout the sweep. It is also worth notice that, even when manual transcriptions are used as the AED back history (green lines), the WER of AED hypotheses were still above 10% (3 points higher than Hybrid-MuSTC). The suboptimal AED transcripts have led to 28.34 BLEU at 100% data ratio, which is higher than the Cascade highest 25.45 BLEU.



(a) SLT BLEU scores versus Data ratio



(b) AED WER versus Data ratio

Fig. 6.8 EP-J decoded with different AED ASR back histories. From left to right, SLT systems were fine-tuned with an increasing amount of end-to-end data. (FR: back history generated under free running, i.e. AED transcripts are used; FR with 4gram LM: shallow fusion with 4gram LM; Hybrid-TED/MuSTC: back history adopts transcripts from hybrid ASR trained with TED/MuSTC; MAN: back history adopts manual transcripts.)

Figure 6.9 compares EP-J with Cascade, both making use of the hybrid ASR transcriptions. The plot only shows the lower and upper bounds of the system performance. Under zero in-domain data, the true performance is closer to the lower bound (Hybrid-TED);

hypotheses gave a different order of Hybrid-TED < Hybrid-MuSTC < FR < MAN (from the highest to the lowest WER). The inversion between FR and Hybrid-TED/MuSTC is mainly due to the degradation in AED caused by the mismatch between the AED hypotheses and the hybrid ASR transcripts.

whereas full data leads to the upper bound (Hybrid-MuSTC). Compared with the vanilla Cascade, it is more challenging for EP-J to indirectly propagate ASR transcripts through the AED back history. In the low data region, Cascade performed better, suggesting that regularisation via words is still effective in handling out-of-domain corpus. After training with 40% of the in-domain data or more, EP-J started to outperform the Cascade system even though the AED performance was still not as good as Hybrid-MuSTC. This confirms that joint embedding passing helps make the system more robust against poor AED performance, and further supports efficient adaptation towards the target domain.

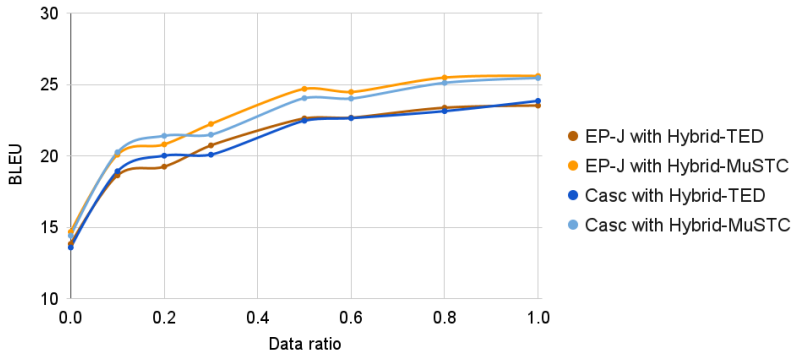


Fig. 6.9 Cascade versus EP-J with Hybrid ASR transcripts

6.6 Summary

This chapter investigated various modular combination approaches for the spoken language translation task. The error mitigation approach operates on a cascaded SLT system. As shown in Table 6.3, under partially annotated data, semi-supervised error mitigation effectively mitigated ASR error propagation by guiding the downstream NMT module with pseudo references, and semi-supervised self-distillation led to further improvement by learning richer distributions as opposed to one-hot references. When fully annotated end-to-end data is available, supervised error mitigation gave the largest gain. The reranking approach adopts an external reranker to access the hypotheses space of both the ASR and NMT modules. The reranker was trained under the target domain, and its posterior was directly optimised to align with the evaluation metric. As shown in Table 6.7, the optimal candidate chosen by the reranker outperformed the maximum a posterior candidate of the vanilla cascaded system, and further incorporating more ASR hypotheses led to additional gains. This confirms that accessing multiple ASR hypotheses via reranking is an effective approach to mitigate information loss in cascaded SLT systems. In the embedding passing experiments in Section 6.5.6,

the static embeddings carry textual information and pose strong regularisation, whereas the dynamic embeddings carry richer acoustic information across modular connections. The combination of the static and dynamic embeddings were adopted as the modular connection to strike a balance between regularisation and richness, thus allowing efficient adaptation towards the target domain under limited end-to-end data.

Chapter 7

Spoken Grammatical Error Correction

This chapter investigates the spoken grammatical error correction (SGEC) task to further analyse multimodular combination under extremely limited end-to-end data. The SGEC task builds on top of the spoken disfluency detection (SDD) task discussed in Chapter 5. It adds an additional error correction module following the speech recognition and disfluecy detection modules. Section 7.1 gives an overview of the task and discusses the cascaded pipeline of the SGEC system. Section 7.2 describes the semi-supervised error mitigation approach that aims to improve the cascaded SGEC system under partially annotated data. Section 7.3 discusses the reranking approach that directly optimises towards the evaluation metric without training on in-domain corpora. In Section 7.4, the embedding passing approach is proposed to encourage tighter modular combination in an integrated SGEC system. An important aspect in computer assisted language learning (CALL) tasks is to give feedback to learners. Section 7.5 describes the feedback extraction process and discusses approaches to improve feedback quality for SGEC systems. Detailed experimental results and analyses are further reported in Section 7.6.

7.1 Task descriptions

Grammatical construction is one of the key elements in second language acquisition, and text-based grammatical error correction (GEC) has been widely studied over the past decade [44, 153, 23]. With speaking skills playing a big part in language learning, it has become increasingly important to analyse spoken grammars. Although no strict rules are followed in free speaking, there are nonetheless phrases that a native speaker is highly unlikely to say. The focus of this work is therefore to give feedback on these 'grammatical errors' to help learners reflect on their spoken language. This chapter discusses the spoken grammatical error correction (SGEC) task, which converts non-native disfluent spoken language into

grammatically correct fluent text. There are several challenges facing SGEC: running automatic speech recognition (ASR) on learner English is harder than native speech due to potential pronunciation and grammatical errors; spoken language often comes with disfluent speech events such as repetitions and false starts, which are disruptive to downstream tasks; there is very little end-to-end speech to correction data that can be used for training.

Constrained by the limited end-to-end data, this work considers a cascaded SGEC system that consists of three modules (shown in Figure 7.1): an ASR module that converts speech into transcripts; a disfluency detection (DD) [234] module that recovers a fluent text flow; and a conventional machine translation style GEC [233] module that conducts error corrections.

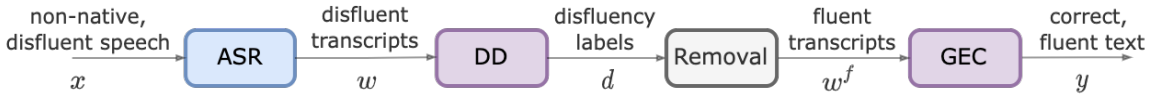


Fig. 7.1 Illustration of a cascaded spoken grammatical error correction system. Three trainable modules are shown in colours. The removal step will be omitted in later diagrams.

Given input speech $\boldsymbol{x}_{1:T}$, disfluent transcription $w_{1:L}$, its corresponding disfluency tags $d_{1:L}$, fluent transcription $w^f_{1:L}$ post disfluency removal[1], and the grammatically correct output sequence $y_{1:N}$, the cascaded SGEC process can be formulated as:

$$\hat{w}_{1:L} = \text{argmax}_{w_{1:L}\in\mathcal{W}} P(w_{1:L}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}_{\text{ASR}}) \tag{7.1}$$

$$\hat{d}_l = \text{argmax}_{d_l} P(d_l|\hat{w}_{1:L};\boldsymbol{\theta}_{\text{DD}}) \quad \hat{w}^f_{1:L} = \text{Removal}(\hat{w}_{1:L},\hat{d}_{1:L}) \tag{7.2}$$

$$P(y_{1:N}|\boldsymbol{x}_{1:T};\boldsymbol{\theta}) \approx P(y_{1:N}|\hat{w}^f_{1:L};\boldsymbol{\theta}_{\text{GEC}}) \tag{7.3}$$

The most likely hypothesis from the ASR module $\hat{w}_{1:L}$ is used as the input to the subsequent DD module, and the hypothesised fluent text $\hat{w}^f_{1:L}$ is further used as the input to its downstream GEC module. Under limited quantities of end-to-end corpora, individual modules in the cascaded SGEC system are separately trained to allow efficient use of data. As discussed in Section 4.1, cascaded systems suffer from issues such as error propagation and information loss. This chapter therefore explores various module combination approaches for the SGEC task, aiming to improve the quality of the corrected sentences as well as the feedback given to learners.

[1]To simplify the notation, L is used to represent the sequence length of both the disfluent and fluent transcription sequence. In practice, the length of the fluent sequence $w^f_{1:L}$ is usually different from the disfluent transcription $w_{1:L}$.

7.2 Error mitigation

The cascaded SGEC system suffers from error propagation due to mismatches between the training and evaluation domains. For example, the GEC module is usually trained on written text corpora due to limited spoken corpora. At the inference stage, the ASR errors in the input to the GEC module would potentially disrupt its performance. Ideally, training on non-native spoken corpora from the target domain would most effectively mitigate error propagation, yet there is usually very little end-to-end data available for training. Semi-supervised error mitigation approaches (discussed in Section 4.2.2) are therefore adopted here to fine-tune the cascaded SGEC system with partially annotated target domain data.

Here a non-native ASR corpus $\{\boldsymbol{x}_{1:T}, w_{1:L}\} \in \mathcal{D}_{\theta_T}(\boldsymbol{x}, w)$ is adopted as the main training set, which is comparatively abundant and less costly to obtain than the end-to-end SGEC annotation. It consists of non-native audio sequences $\boldsymbol{x}_{1:T}$ and the corresponding manual transcriptions $w_{1:L}$, without references for fluent text $w^f_{1:L}$ or grammatical error corrections $y_{1:N}$. Figure 7.2 illustrates the semi-supervised error mitigation pipeline. The pseudo references are generated by feeding manual transcriptions through the cascaded SGEC system, and the hypotheses are generated by feeding the non-native audio sequences through the pipeline. The pseudo references derived from manual transcripts are not impacted by ASR errors, and therefore minimising the distance between the references and hypotheses helps mitigate ASR error propagation.

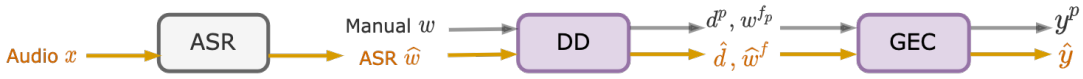


Fig. 7.2 Semi-supervised error mitigation pipeline. Grey arrows denote reference generation, and orange arrows denote hypotheses generation. The ASR block is fixed during semi-supervised training, and the DD and GEC modules are separately updated.

The pseudo references $d^p_{1:L}$ and $w^{f_p}_{1:L}$ for disfluency tagging can be generated by feeding the manual transcripts $w_{1:L}$ through the DD module:

$$d^p_{1:L} = \text{argmax}_{d_{1:L}} P(d_{1:L}|w_{1:L}; \boldsymbol{\theta}_{\text{DD}}) \quad w^{f_p}_{1:L} = \text{Removal}(w_{1:L}, d^p_{1:L}) \tag{7.4}$$

For sequence tagging tasks, reference tags have a one-to-one correspondence with the input tokens, and thus the pseudo reference tags $d^p_{1:L}$ of the manual transcripts cannot be directly applied to the ASR transcripts. Alternatively, reference tags on ASR transcripts $d^{p'}_{1:L}$ can be derived by aligning the reference fluent text $w^{f_p}_{1:L}$ with the ASR transcripts $\hat{w}_{1:L}$: all insertions in $\hat{w}_{1:L}$ are considered as disfluencies; substitutions and matched words are tagged as fluent; deletions are ignored (illustrated in Figure 7.3). With the pseudo reference $d^{p'}_{1:L}$, the binary

cross entropy loss can be applied:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{DD}}^{\text{EMsemi}}) = -\log P(d_{1:L}^{p'}|\hat{w}_{1:L};\boldsymbol{\theta}_{\text{DD}}^{\text{EMsemi}}) \tag{7.5}$$

where $\hat{w}_{1:L}$ denotes the ASR transcripts from Equation 7.1, and minimising the cross entropy loss is equivalent to directly optimising for higher quality fluent transcripts $w_{1:L}^{f}$.

w^{f_p}	when i was a child i dreamed _ to work like a doctor
$\hat{w}$	when i was a child i dream to to work like _ doctor
Align	M | M | M | M | M | M | S | **I** | M | M | **D** | M
$d^{p'}$	O | O | O | O | O | O | O | **E** | O | O | - | O

Fig. 7.3 An example of generating reference tags $d_{1:L}^{p'}$ by aligning the pseudo reference fluent text $w_{1:L}^{f_p}$, and the ASR transcript $\hat{w}_{1:L}$. M:match, D:deletion, S:substitution, I:insertion; E:disfluent, O:fluent, '-': no label for deleted tokens.

The pseudo reference $y_{1:N}^{p}$ for the correction output can be generated by feeding $w_{1:L}^{f_p}$ through the GEC module, and the standard cross entropy loss can be used:

$$y_{1:N}^{p} = \text{argmax}_{y_{1:N}\in\mathcal{Y}} P(y_{1:N}|w_{1:L}^{f_p};\boldsymbol{\theta}_{\text{GEC}}) \tag{7.6}$$

$$\mathcal{L}(\boldsymbol{\theta}_{\text{GEC}}^{\text{EMsemi}}) = -\log P(y_{1:N}^{p}|\hat{w}_{1:L}^{f};\boldsymbol{\theta}_{\text{GEC}}^{\text{EMsemi}}) \tag{7.7}$$

where $\hat{w}_{1:L}^{f}$ denotes the hypothesised fluent transcripts from Equation 7.2. Minimising the cross entropy loss is equivalent to maximising the sentence-level probabilities, which leads to higher quality correction outputs $y_{1:N}$.

Semi-supervised error mitigation relies on the pseudo references generated from the manual transcripts, the quality of which largely depends on the baseline SGEC system. To alleviate the potential degradation caused by erroneous pseudo references, semi-supervised self-distillation can be further applied to the GEC module:

$$\mathcal{L}_{\text{KL}}(\boldsymbol{\theta}_{\text{GEC}}^{\text{EMdistil}}) = \Sigma_{n=2}^{N}\,\text{KL}\{P(y_n^p|y_{1:n-1}^p,\hat{w}_{1:L}^f;\boldsymbol{\theta}_{\text{GEC}}^{\text{EMsemi}})||P(\boldsymbol{y}_n^p|y_{1:n-1}^p,\hat{w}_{1:L}^f;\boldsymbol{\theta}_{\text{GEC}}^{\text{EMdistil}})\} \tag{7.8}$$

7.3 Reranking

Reranking (discussed in Section 4.3.1.2) is an alternative approach to improve the cascaded SGEC system by directly optimising towards the metric of interest. Due to extremely limited

end-to-end SGEC data, the reranker in this chapter can only be trained on written GEC data $\mathcal{D}_{\theta_y}(w^f, y)$, and further evaluated on the spoken GEC corpus $\mathcal{D}_{\theta_T}(\boldsymbol{x}, w, w^f, y)$. When feeding a sequence of fluent text input $w^f_{1:L}$ into the GEC module, a hypothesis set $\mathcal{Y}$ can be generated for the correction sequence $y_{1:N}$:

$$\hat{y}^{(j)}_{1:N} \sim P(y_{1:N}|w^f_{1:L}; \boldsymbol{\theta}_{\text{GEC}}) \quad \mathcal{Y} = \{\hat{y}^{(1)}_{1:N}, ..., \hat{y}^{(M_y)}_{1:N}\} \tag{7.9}$$

The reranker assigns a scalar score to each correction hypothesis $\hat{y}^{(j)}_{1:N} \in \mathcal{Y}$, and the highest scoring candidate is chosen as the optimal output:

$$\hat{y}_{1:N} = \text{argmax}_{y_{1:N} \in \mathcal{Y}} P(y_{1:N}|w^f_{1:L}; \boldsymbol{\theta}_r) \tag{7.10}$$

Constrained by data availability, the GEC reranker only accounts for the single input sequence $w^f_{1:L}$, and adopts the basic scoring function defined in Equation 4.32. Figure 7.4 shows an illustration of the text based GEC reranker.

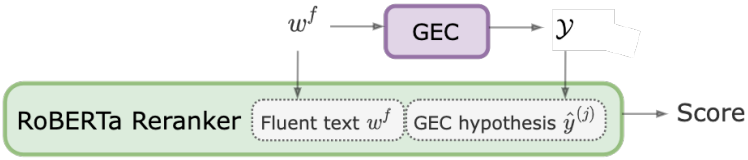


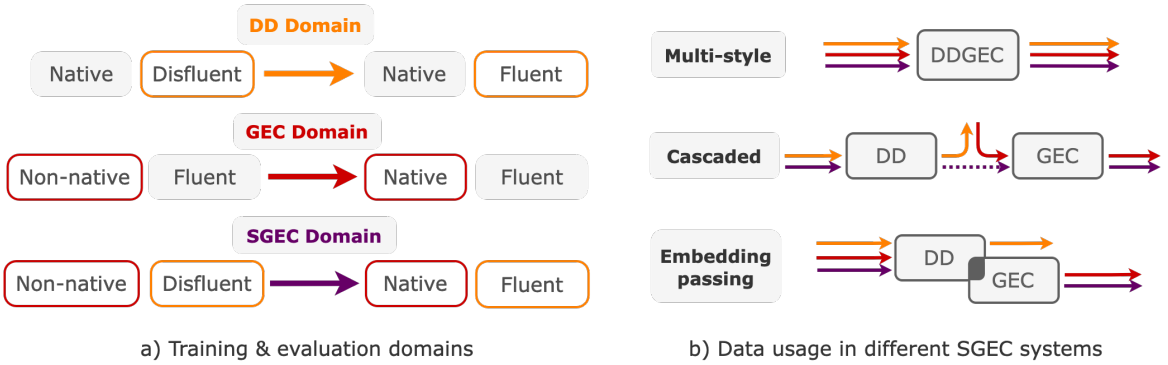
Fig. 7.4 Illustration of the GEC reranking pipeline

The reference scores are defined using the BLEU [158] metric, which a standard metric for translation tasks. Ideally, the reranker can be optimised towards the task specific evaluation metric, i.e. sentence error rate (SER) or translation edit rate (TER) of the correction sequences (see Section 7.6.1). SER calculates the sentence level matches, and TER gives the token level edit distance. Neither of the two provides the level of granularity required to differentiate the correction candidates. The n-gram based BLEU score, being an average over multiple levels of n-gram matches, provides a much higher granularity, and is therefore adopted here for reference calculation. For the feature extraction function in the reranker, the pre-trained RoBERTa [134] model is adopted. RoBERTa follows the basic BERT [104] architecture, but is trained with a robustly optimised training recipe (discussed in Section 2.4). Being a large pre-trained unilingual model, RoBERTa is capable of extracting correlations between sentences before and after the error correction process, which suits well with the GEC task.

7.4 Embedding passing

Loosely cascaded systems adopt discrete sequences to pass information, which leads to error propagation and information loss. This section focuses on the embedding passing approach (introduced in Section 4.3.2), which encourages tighter integration by propagating richer information between the DD and GEC modules[2]. While reserving a sequential modular structure, the embedding passing system allows gradients to back propagate through the modular connection, and thus simultaneously optimises for both modules.

Three data domains[3] are considered for the SGEC task (Figure 7.5a): the auxiliary DD domain $\mathcal{D}_{\text{DD}}$ converts disfluent native transcripts into fluent native transcripts; the auxiliary GEC domain $\mathcal{D}_{\text{GEC}}$ converts fluent non-native text into fluent native text; and the target SGEC domain $\mathcal{D}_{\text{SGEC}}$ converts disfluent non-native transcripts $w_{1:L}$ into fluent native transcripts $y_{1:N}$. There are readily available corpora for both DD and GEC domains, yet there is little data for the target SGEC domain. The challenge is therefore to make use of the auxiliary DD and GEC data and optimise for the unseen SGEC task. As shown in Figure 7.5b, three system architectures are considered here, each using the auxiliary corpora differently.



a) Training & evaluation domains

b) Data usage in different SGEC systems

Fig. 7.5 a) The DD, GEC and SGEC domains. b) The information flows from the three domains in the multi-style, cascaded and embedding passing systems. Note: 'native' implies grammatically correct text, whereas 'non-native' implies grammatically incorrect text.

Multi-style The multi-style system simply blends together the data from the DD and GEC domains, and trains a single sequence-to-sequence model that simultaneously does disfluency removal and error correction:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{multi}}) = -\log P(y_{1:N}|w_{1:L};\boldsymbol{\theta}_{\text{multi}}) \quad \{w_{1:L}, y_{1:N}\} \in \{\mathcal{D}_{\text{DD}}, \mathcal{D}_{\text{GEC}}\} \tag{7.11}$$

[2]This section only considers the combination of DD and GEC, with the upstream ASR module fixed.

[3]Ideally, the DD and GEC modules should be trained on non-native spoken data. However, there is little annotated in-domain corpora available, and therefore the auxiliary DD and GEC domains are adopted.

where the input $w_{1:L}$ and output $y_{1:N}$ pairs are from both the DD and GEC domains. At the inference stage, the multi-style system is directly applied to the SGEC domain $\mathcal{D}_{\text{SGEC}}$. The idea of multi-style training is borrowed from the multilingual machine translation [100]. The multilingual training enables zero-shot translation, where the unseen task benefits from interlingua representation through diversifying the source and target domains. Here the multi-style data forces the system to simultaneously remove disfluencies and correct errors, which to some extends enables the system to handle disfluent non-native transcripts. The multi-style system drops the notion of modules, and the stand-alone structure is not trained to separate the DD and GEC domains.

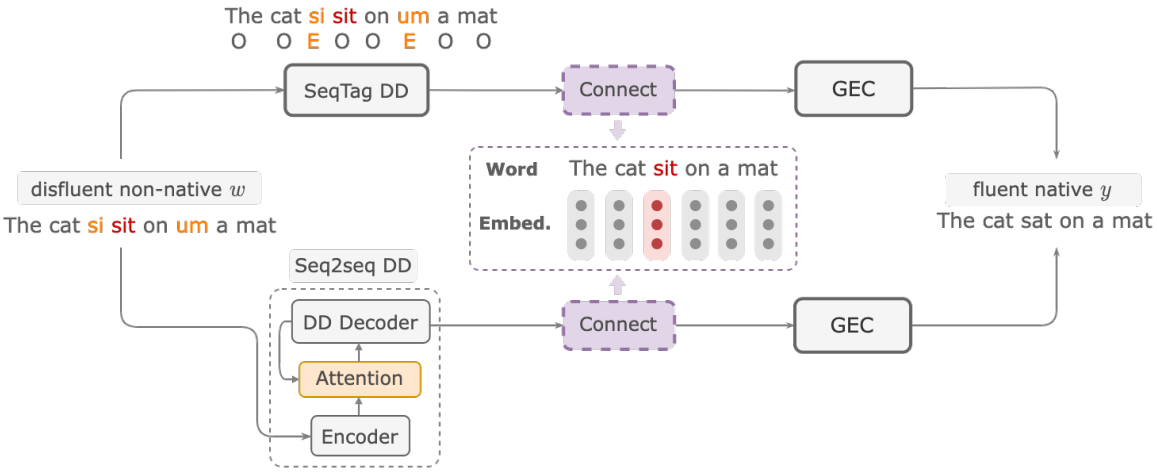


Fig. 7.6 Illustration of the cascaded and embedding passing SGEC systems (omitting the ASR module). Purple blocks represent modular connections: the cascaded system is connected via words, and the embedding passing system is connected via embeddings. The upper half shows the system with SeqTag DD, and the lower half shows the system with Seq2seq DD. Text in orange denotes disfluencies, and text in red denotes grammatical errors.

Cascaded Unlike the implicit modelling of disfluency removal in the multi-style system, the cascaded system adopts an explicit DD module and a text-based GEC module. As shown in Figure 7.5b, the DD and GEC modules are trained in their corresponding domains, and then concatenated together for evaluation in the SGEC domain:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{DD}}) = -\log P(w^f_{1:L}|w_{1:L};\boldsymbol{\theta}_{\text{DD}}) \quad \{w_{1:L}, w^f_{1:L}\} \in \mathcal{D}_{\text{DD}} \tag{7.12}$$

$$\mathcal{L}(\boldsymbol{\theta}_{\text{GEC}}) = -\log P(y_{1:N}|w^f_{1:L};\boldsymbol{\theta}_{\text{GEC}}) \quad \{w^f_{1:L}, y_{1:N}\} \in \mathcal{D}_{\text{GEC}} \tag{7.13}$$

The cascaded system uses word tokens $w^f_{1:L}$ as the modular connection (Figure 7.6), which can be interpreted as the non-native fluent transcripts. Two different DD formulations are considered here: sequence tagging (Seqtag) and sequence-to-sequence (Seq2seq). SeqTag

DD assigns a binary tag to each input token (discussed in Section 3.3), and Seq2seq DD models a translation process that removes speech disfluencies. During training, the downstream GEC module will not be disrupted by disfluencies, yet at the evaluation stage, errors in disfluency removal will potentially propagate through and degrade the SGEC performance.

Embedding passing The embedding passing system keeps the modular structure, and adopts embeddings as the modular connection:

$$P(y_{1:N}|w_{1:L};\boldsymbol{\theta}) \approx P(y_{1:N}|\boldsymbol{e}_{1:L};\boldsymbol{\theta}^{\text{EP}}_{\text{GEC}}) \quad \boldsymbol{e}_{1:L} = f(w_{1:L};\boldsymbol{\theta}^{\text{EP}}_{\text{DD}}) \tag{7.14}$$

where the embeddings $\boldsymbol{e}_{1:L}$ can be viewed as the continuous representations of the modular connection $w^f_{1:L}$ used in the cascaded system. In order to allow gradients to flow through the modular connection, the embedding passing approach is used with the Seq2seq style DD structure (see Figure 7.6). The embeddings contain the information of the complete back history of the previous context, and allows a much richer information flow through the modular connection between the DD and GEC modules. The embedding passing system can be initialised under the auxiliary domains $\mathcal{D}_{\text{DD}}$ and $\mathcal{D}_{\text{GEC}}$. The auxiliary DD training is formulated similarly as the cascaded system:

$$\mathcal{L}(\boldsymbol{\theta}_{\text{DD}}) = -\log P(w^f_{1:L}|w_{1:L};\boldsymbol{\theta}_{\text{DD}}) = -\log P(w^f_{1:L}|\boldsymbol{e}_{1:L};\boldsymbol{\theta}_{\text{aux}}) \tag{7.15}$$

$$\boldsymbol{\theta}_{\text{DD}} = \{\boldsymbol{\theta}^{\text{EP}}_{\text{DD}}, \boldsymbol{\theta}_{\text{aux}}\} \quad \{w_{1:L}, w^f_{1:L}\} \in \mathcal{D}_{\text{DD}} \tag{7.16}$$

where $\boldsymbol{\theta}_{\text{aux}}$ denotes the auxiliary parameters that project the embeddings $\boldsymbol{e}_{1:L}$ to the intermediate fluent sequences $w^f_{1:L}$. The embeddings $\boldsymbol{e}_{1:L}$ are later used as the modular connection for integrated training. When training under $\mathcal{D}_{\text{GEC}}$, the non-native fluent text $w^f_{1:L}$ is piped through both the DD and GEC modules[4]:

$$\mathcal{L}(\boldsymbol{\theta}) = -\log P(y_{1:N}|w^f_{1:L};\boldsymbol{\theta}) \quad \{w^f_{1:L}, y_{1:N}\} \in \mathcal{D}_{\text{GEC}} \tag{7.17}$$

The rationale behind exposing the DD module under $\mathcal{D}_{\text{GEC}}$ is to encourage better generalisation to the non-native disfluent data in the target SGEC domain. At the fine-tuning stage, the embedding passing system can be further adapted to the target domain $\mathcal{D}_{\text{SGEC}}$:

$$\mathcal{L}(\boldsymbol{\theta}) = -\log P(y_{1:N}|w_{1:L};\boldsymbol{\theta}) \quad \{w_{1:L}, y_{1:N}\} \in \mathcal{D}_{\text{SGEC}} \tag{7.18}$$

[4]Note that this is different from the general embedding passing approach described in Section 4.3.2. In general, the upstream module is not trained on the auxiliary domain associated with the downstream module. Here, an ideal DD module will make no change to the input fluent text $w^f_{1:L}$, and the embedding connection $\boldsymbol{e}_{1:L}$ should directly correspond to the input sequence $w^f_{1:L}$.

7.5 Feedback

The previous sections discussed approaches that aim to improve the output quality of the SGEC system. Another important aspect of language learning applications is feedback, which directly impacts learner's progression in learning. This section first describes how feedback is extracted and assessed for SGEC systems, and then introduces confidence-based filtering that aims to improve the feedback precision.

For GEC tasks, feedback usually suggests where the error occurred and how to make the correction. Feedback is therefore extracted by comparing the sequences before and after error correction using MaxMatch (M^2) [43], which is a phrase-level edit extraction algorithm. Given a source sequence $w^f_{1:L}$, a reference correction $y_{1:N}$ and a hypothesised correction $\hat{y}_{1:N}$, the reference and hypothesised edits can be extracted as:

$$E = M^2(w^f_{1:L}, y_{1:N}) \quad \hat{E} = M^2(w^f_{1:L}, \hat{y}_{1:N}) \tag{7.19}$$

Each edit E is defined using a triplet $\{p_1, p_2, c\}$, where p_1 and p_2 are the start and end positions with respect to the source sentence, and c denotes the correction phrase. Figure 7.7 gives an example of the extracted edits. $F_{0.5}$ scores can then be calculated by comparing the reference and hypothesised edits, following the general F-score formulation (Section 5.3.1):

$$F_{0.5}(E, \hat{E}) \tag{7.20}$$

When computing $F_{0.5}$, true positives are defined as correctly predicted edits, false positives are incorrectly predicted edits, and false negatives are reference edits that are not successfully predicted. False negatives are not defined in GEC tasks. $F_{0.5}$ assesses GEC performance in terms of edit accuracy, which suits well with feedback oriented applications.

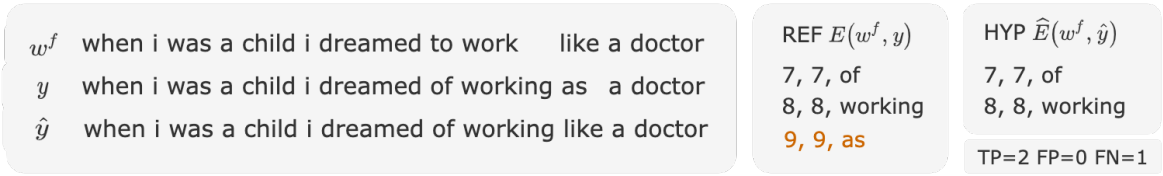


Fig. 7.7 An example of the M^2 edit extraction process for $F_{0.5}$ calculation. The edit coloured in orange is a false negative feedback.

However, Equation 7.19 is not directly applicable to SGEC, since the reference edits change with the upstream ASR transcriptions, are consequently $F_{0.5}$ scores are not comparable

across systems. To adapt to SGEC systems, the reference and hypothesised edits are modified:

$$E = M^2(w^f_{1:L}, y_{1:N}) \quad \hat{E} = M^2(\hat{w}^f_{1:L}, \hat{y}_{1:N}) \tag{7.21}$$

where the reference edits E are generated using manual fluent transcripts $w^f_{1:L}$ as source sequences, and the hypothesised edits $\hat{E}$ use hypothesised fluent transcriptions $\hat{w}^f_{1:L}$ as source sequences. With the reference edits E defined independently of the ASR and DD modules, feedback $F_{0.5}$ can be compared across systems. The hypothesised edits $\hat{E}$ account for errors from all ASR, DD and GEC modules, and reflects the true feedback given to users when the system is deployed. Note that the mismatched source sequences in E and $\hat{E}$ put extra penalties on $F_{0.5}$, and Figure 7.8 illustrates this issue with an example: even when the system output $\hat{y}_{1:N}$ matches exactly with the reference $y_{1:N}$, differences in $w^f_{1:L}$ and $\hat{w}^f_{1:L}$ still resulted in differences in E and $\hat{E}$, leading to degraded $F_{0.5}$ scores.

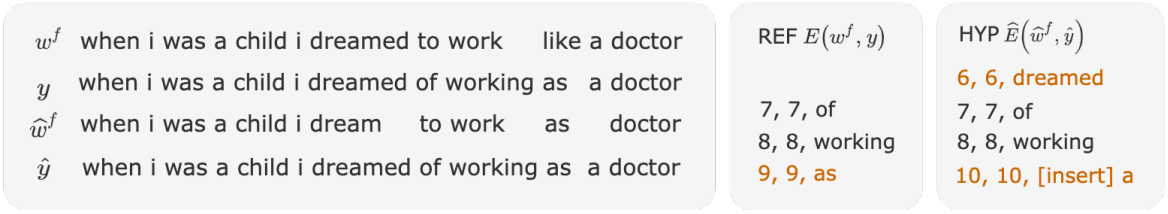


Fig. 7.8 An example of reference and hypothesis feedback extraction with mismatched $w^f_{1:L}$ and $\hat{w}^f_{1:L}$. The edits coloured in orange indicate the artificial mismatches in feedback due to the ASR transcription error.

The SGEC system faces potential errors arising from the transcripts, disfluencies as well as the correction process, and it is important not to give erroneous feedback to language learners. A confidence filtering approach can be adopted to filter out edits that the system has little confidence in, assuming lower confidence indicates lower accuracy. To conduct confidence filtering, a confidence measure needs to be defined. In a cascaded SGEC pipeline, each module produces a token-level confidence score associated with its prediction, and the sentence-level confidence can be defined for each module as the lowest token probability over the entire sentence. Sentence-level filtering can be conducted by rejecting sentences with low confidence. An alternative is to adopt edit-level confidence, where the confidence scores are calculated over each edit instead of the sentence[5]. The overall confidence score of the SGEC system can be calculated using a weighted sum over all three modules:

$$\log P = \alpha \log P_{\text{ASR}} + \beta \log P_{\text{DD}} + \gamma \log P_{\text{GEC}} \tag{7.22}$$

[5]For the ASR module, the edit-level confidence remains as the lowest token probability over the sentence, in order to mitigate a known issue of ASR error propagation.

where P_{ASR}, P_{DD}, P_{GEC} represent sentence-level or edit-level confidence scores of each module, and α, β, γ are their respective weight coefficients.

7.6 Experiments

The experiments in this chapter investigate the module combination approaches discussed from Section 7.2 to 7.4. This section first discusses the evaluation metrics and the corpora, followed with the model training details. The error mitigation and reranking approaches directly operate on the cascaded SGEC system, both of which are trained with out-of-domain corpora. The embedding passing approach builds a tightly integrated multimodular system under extremely limited end-to-end data. In addition to output quality, emphasis is also laid on feedback quality. Confidence based filtering approaches discussed in Section 7.5 are adopted to improve the feedback precision.

7.6.1 Evaluation metrics

Multimodular systems can be evaluated both in terms of individual module performance, and combinations of multiple modules. When evaluating individual modules of the SGEC system, standard metrics can be used: word error rate (WER) is used to assess the ASR module; F_1 score is adopted to account for the tagging accuracy of the DD module (described in Section 5.3.1); GLUE and M^2 $F_{0.5}$ can be used to assess the GEC module. The feedback based metric M^2 $F_{0.5}$ was introduced in Section 7.5, and the GLUE [149] metric, short for Generalized Language Evaluation Understanding, is a BLEU [158] inspired metric that accounts for a weighted n-gram precision. $M^2F_{0.5}$ is adopted in this thesis.

For individual module assessment, the input to each module is fixed as the reference transcripts rather than the hypotheses. However, when evaluating combinations of modules, the input to each module depends on the upstream module output, and therefore metrics that rely on a fixed input would no longer work. When evaluating the ASR and DD modules combined, the standard DD metric F_1 score no longer applies. As discussed in Section 5.3.1, the reference tags $d_{1:L}$ have a one-to-one correspondence with input word tokens $w_{1:L}$, and a different set of reference is needed for the ASR transcriptions. Thus, an alternative is proposed, WER, to directly analyse the quality of the fluent text after disfluency removal. When evaluating the ASR, DD and GEC modules combined, i.e. the overall SGEC system, the standard GEC metric $M^2F_{0.5}$ cannot be used, since it does not allow comparison across systems. $M^2F_{0.5}$ requires the input sequence $w^f_{1:L}$ to be fixed for edit extraction, and changes in the upstream ASR and DD modules will lead to a different set of reference edits E.

Therefore, the focus is laid on the quality of the system outputs. Sentence error rate (SER) is used to analyse sentence-level matches between the reference and hypothesised correction sequences. To achieve greater granularity, translation edit rate (TER) [193] is also adopted to assess the word-level distance. Given a set of hypothesis $\hat{y}_{1:N}$ and reference $y_{1:N}$ sentences, SER and TER can be calculated:

$$\text{SER} = \frac{\text{\#matched sentences}}{\text{\#sentences}} \tag{7.23}$$

$$\text{TER} = \frac{\text{S+I+D}}{\text{M+S+D}} = \frac{\text{S+I+D}}{\text{\#words in reference}} \tag{7.24}$$

where S, I, D, M respectively stands for the number of substitution, insertion, deletion and matched words. When evaluating the DD and GEC combined, the GEC metrics still apply, since the system input is fixed as the disfluent manual transcripts $w_{1:L}$.

Modules	Metrics	Args
ASR	WER	$\hat{w}, w$
DD	F_1	$\hat{d}, d, w$
GEC	GLEU, M^2 $F_{0.5}$	$\hat{y}, y, w^f$
DD + GEC	GLEU, M^2 $F_{0.5}$	$\hat{y}, y, w$
ASR + DD	WER	$\hat{w}^f, w^f$
ASR + DD + GEC	TER, SER	$\hat{y}, y$

Table 7.1 Evaluation metrics and their corresponding arguments for assessing individual modules and combinations of modules in the SGEC system. w: disfluent transcripts, w^f: fluent transcripts post disfluency removal, d: disfluency tags, y: grammar correction output.

Table 7.1 summarises the assessment metrics for individual and combinations of modules. Individual module evaluation helps develop each module separately. System-level metrics on ASR+DD and ASR+DD+GEC both emphasise the output quality, which enables comparison across systems even when the upstream modules change. They also help guide further tuning and development of the SGEC system as a whole.

7.6.2 Corpora

This work focuses on spoken grammatical error correction for the English language. The corpora used in the experiments are grouped in terms of their respective domains, namely ASR, DD, GEC and SGEC. Relevant corpora statistics are summarised in Table 7.2.

Corpus	Spoken	Usage	#Sent	#Word	%Disfluency
ASRtrn	✓	Train - ASR	62K	2.5M*	-
SWBD	✓	Train - DD	174K	1.3M	5.7†
CLC	✗	Train - GEC	1.9M	25.2M	-
BEA	✗	Train - GEC	1M	11.5M	-
SWBDtst	✓	Eval - DD	8,039	66K	6.58
FCEtst	✗	Eval - GEC	2,681	37K	-
BULATS	✓	Eval - SGEC	3,650	64K	3.4(+5.8‡)
LIN	✓	Eval - SGEC	3,361	38K	5.0

Table 7.2 Corpora statistics. Spoken: whether it is derived from speech; *: approximated value, no manual transcriptions available; †: fillers are not counted as disfluencies; ‡: percentage of words that are annotated as unnecessary, most of which are disfluencies.

ASRtrn is used for ASR training, as well as the experiments on semi-supervised error mitigation. It consists of 334 hours of an online English speaking test data, which mainly covers 28 L1s[6] and the 5 CEFR [42] grades ranging from A1 to C2. Different from usual ASR training corpora, it only provides crowd source transcriptions, the quality of which is not as good as manual transcriptions. A remedy for this is to bootstrap multiple ASR systems with the crowd source data, and use an ensemble to generate higher quality transcriptions as discussed in [209]. Such lightly supervised approach is often used to produce higher quality transcriptions given the low quality original transcriptions [115, 118]. For the following experiments, manual transcriptions always refer to the higher quality transcriptions generated using the ensemble. **Switchboard (SWBD)** [142] is used for DD training. It consists of 260 hours of telephone conversations of Native American English speakers. The Treebank-3 annotation [200] provides manual transcripts and disfluency annotations, and Switchboard-300 [98] provides higher quality manual transcriptions. The two corpora were aligned as described in Section 5.3.2 to obtain the paired transcription and disfluency annotation. **Cambridge Learner Corpus (CLC)** [155] is used for GEC training. It is a collection of written exams of candidates from 86 L1s at different proficiency levels. The corpus was annotated with grammatical errors. **BEA** [22] is also used for GEC training. It is a collection of text-based grammatical error correction corpora, including Write & Improve, LOCNESS, Lang-8 and NUCLE (FCE train split excluded, since it overlaps with CLC). **SWBDtst** is a held out test set from the Switchboard corpus, and it is used to evaluate the DD models. **FCEtst** [230] is a held out subset of the CLC corpus used for GEC evaluation.

[6] L1 stands for first language. Similarly, L2 stands for second language.

Punctuation and capitalisation were removed from all corpora derived from written text, to make the text closer to speech transcriptions. **BULATS** [26] and **Linguaskill (LIN)** are the evaluation datasets from the SGEC domain. BULATS is a free speaking business English test consisting of prompted responses of up to 1 minute. The 225 learners are from 6 L1s and have an even distribution across all speaking CEFR grades. BULATS manual transcriptions were annotated with metadata, error types and corrections, with some ambiguous words annotated as unknown. LIN is from the same domain as the BULATS set, but with better annotation quality. It consists of 340 learners from over 30 L1s. The manual transcriptions were segmented at the phrase level, with incomplete or ambiguous phrases rejected. The remaining set was annotated with disfluencies and grammatical errors. Experiments in this section were mostly evaluated on the LIN dataset, except for the embedding passing experiments, which involved fine-tuning to BULATS before LIN became available.

7.6.3 Model training

Three individual modules are considered: ASR, DD and GEC. For the ASR module, a hybrid based ASR model was adopted. For the DD module, both an RNN-based model and a BERT-based pre-trained model were used. For the GEC module, an RNN-based model and a Gramformer-based pre-trained GEC model were used. The RNN models use a 200D word embedding with GloVe [161] initialisation, and the vocabulary was generated from the Switchboard and CLC corpora excluding rare words with less than four occurrences. The learning rate was set to 0.001 with gradient clipping. The Transformer-based models were initialised from the pre-trained weights, and further training was conducted under a learning rate of 5e-4 with warm up. Unless specified otherwise, all neural models were trained using the Adam optimiser [105] with a batch size of 256 and a dropout rate of 0.2, and the maximum sentence length was set at 64. The model parameters were calculated using checkpoint averaging [94], which took the average over the 5 best checkpoints.

Individual modules

For the hybrid ASR model, a lattice-free maximum mutual information (LF-MMI) [166] TDNN-F [165] acoustic model was trained on the ASRtrn corpus, and evaluated on BULATS and LIN (Table 7.3). The model adopted sequence-level teacher student training [209], which trains the student to produce the same decoding output as the teacher ensemble. The decoding stage adopted a trigram lattice generation, followed with a succeeding word RNNLM [29] rescoring. Confidence scores used for the feedback filtering experiments were returned by the ASR engines, and the scores were rescaled with a piece-wise linear mapping [53].

WER↓	BULATS	LIN
Hybrid	19.50	19.97

Table 7.3 The hybrid ASR model evaluated on BULATS and LIN corpora. The language model scale factor was set at 10.

The DD module is modelled as a binary classification task. The RNN-based DD consists of 2x300D BLSTM followed by a binary classifier, and the dropout rate was set at 0.5. The BERT-based [104] DD adopts the 'bert-base-uncased' implementation provided by the HuggingFace Library [221]. The BERT layer is connected to a 768x128 dense layer, followed with an output layer of size 2. The model was trained with a learning rate of 1e-06 and dropout 0.1. Table 7.4 contrasts the RNN and BERT-based DD, both trained on the native SWBD corpus, and evaluated on the SWBDtst and the non-native LIN dataset. The BERT initialisation gave significant improvements on both the in-domain and out-of-domain evaluation sets compared with the vanilla RNN-based DD.

$F_1 \uparrow$	SWBDtst	LIN
RNN	81.87	62.97
BERT	89.63	79.29

Table 7.4 Comparing RNN-based and BERT-based DD models. Manual transcripts were used for evaluation. The operating thresholds were chosen at 0.40 for RNN and 0.60 for BERT taggers.

The GEC module is modelled as a sequence-to-sequence translation task. The RNN-based GEC consists of an encoder with 2x200D bidirectional LSTM layers, a decoder with a 4x200D unidirectional LSTM layers, and a bilinear attention connecting the two. The RNN model was trained on the CLC corpus. The Gramformer-based GEC (described in Section 3.5) adopts the basic Transformer structure. In this thesis, the pre-trained Gramformer model was further fine-tuned on both the CLC and BEA corpora, and then used for the SGEC task. Table 7.5 contrasts the RNN-based and Gramformer-based GEC models, and evaluates their performance on both the in-domain FCEtst set and the spoken style LIN dataset. The Gramformer model significantly outperformed the RNN model. The additional BEA training data used for the Gramformer model helps, and the Transformer-based structure inherently has stronger modelling power. The general purpose T5 initialisation for the Gramformer model also improves the implicit language model, and thus Gramformer generalises better to the unseen LIN dataset from the spoken domain.

M^2 $F_{0.5}$ ↑	FCEtst	LIN
RNN	48.90	36.52
Gramformer	56.60	53.57

Table 7.5 Comparing the RNN-based and Gramformer-based GEC models. Fluent manual transcripts were used for evaluation, and the hypotheses were generated using greedy search.

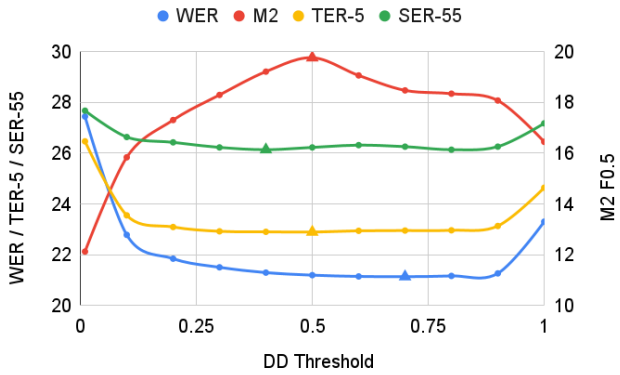
The error mitigation and reranker experiments are conducted on the cascaded system consisting of the hybrid ASR, BERT-based DD and Gramformer-based GEC. The analyses on feedback also adopt this cascaded pipeline consisting of the pre-trained models. The embedding passing experiment uses the concatenation of the RNN-based DD and RNN-based GEC as the baseline, and it further investigates the impact of the embedding connection.

Metrics and tuning

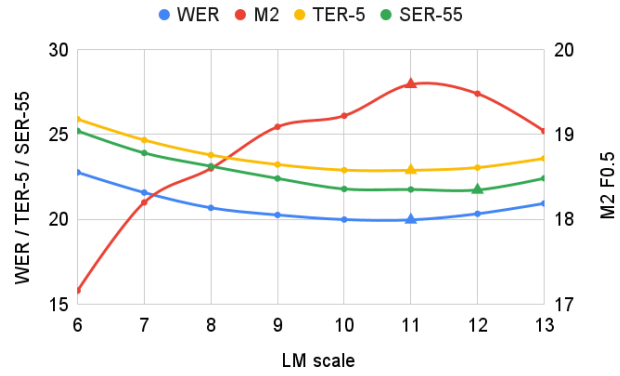
With the system level metrics defined in Section 7.6.1, hyperparameters of individual modules in the cascaded SGEC system (hybrid ASR, BERT-based DD and Gramformer-based GEC) can be jointly tuned towards a better quality output.

Two tunable variables are considered here: the ASR language model (LM) scale factor, and the disfluency removal threshold, above which the words are classified as disfluencies. A two-dimensional grid search was conducted over a range of LM factors (6-13) and DD thresholds (0.0-1.0). Figure 7.9a shows a sweep over the disfluency removal thresholds at the chosen LM scale factor, and Figure 7.9b shows a sweep over the LM factor at the chosen DD threshold. It can be seen that all edit distance based metrics are relatively insensitive to the sweep, whereas the feedback $M^2F_{0.5}$ shows a stronger preference (feedback quality will be further analysed in Section 7.6.7). According to the best WER for the intermediate DD output, the optimal DD threshold is at 0.7, whereas the best SER/TER for the final GEC output led to a threshold of 0.4. Although the differences are insignificant, this shows that an intermediate optima can be different from the overall optima. The final operating point was chosen according to system SER/TER, which is at a LM scale of 11 and a disfluency threshold of 0.4.

Table 7.6 summarises the system level performance of the cascaded SGEC system at the chosen operating point (this tuned system is used as the baseline for future experiments), and the evaluation was conducted on both manual and ASR transcripts. It can be seen that, going from manual to ASR transcripts, there was a significant increase in the overall SER and TER, which motivates further system development in mitigating ASR error propagation.



(a) A sweep over DD thresholds at the chosen LM scale = 11



(b) A sweep over LM scale factor at the chosen DD threshold = 0.4

Fig. 7.9 Sweeping over DD thresholds and LM scale factors. WER assesses the ASR+DD output, SER and TER assess the ASR+DD+GEC output, and $M^2F_{0.5}$ assesses the SGEC feedback quality. The optimal point for each metric are marked in triangles in the plot, and the operating point was chosen according to the system level metrics SER and TER, with a LM scale of 11 and a disfluency threshold of 0.4.

Modules	Metric	ASR	MAN
ASR+DD	WER ↓	21.20	1.96
ASR+DD+GEC	WER ↓	27.22	9.00
	SER ↓	76.76	43.26
	TER ↓	27.89	8.27

Table 7.6 Evaluating the cascaded SGEC system on LIN corpus. The ASR column shows the performance on ASR transcriptions with automatic disfluency removal, with the ASR and DD modules at their respective operating points. The MAN column shows performance on fluent manual transcriptions.

7.6.4 Error mitigation

This section analyses the impact of error mitigation (discussed in Section 7.2) on the cascaded SGEC pipeline, and all experimental results are reported on the LIN dataset from the SGEC domain. The cascaded system is composed of the hybrid ASR, the BERT-based DD and the Gramformer-based GEC modules, with its baseline results listed in Table 7.6. It is shown that ASR transcription errors tend to induce large performance drop, which motivates the error mitigation experiments.

ASR errors resulted in large performance degradation, since the DD and GEC modules have not encountered any non-native spoken data during training. Directly fine-tuning on non-native spoken corpora is the most efficient way to mitigate ASR error propagation. However, constrained by data availability, semi-supervised error mitigation was adopted instead. The ASR training data (ASRtrn) was fed through the cascaded SGEC pipeline for pseudo reference generation. The system performance on the manual transcripts of LIN in Table 7.6 gives an approximation of how much the semi-supervised approach will fall behind the supervised error mitigation. Table 7.7 shows the impact of error mitigation: the Base model is the vanilla cascaded SGEC system, and EMsemi-DD, EMsemi-GEC are the systems with the respectively fine-tuned DD and GEC modules.

Semi-supervised fine-tuning of the DD module (EMsemi-DD) improved the WER of the DD output, yet it did not further improve the SER/TER of the GEC output. When generating the pseudo reference tags, the objective is to minimise the edit distance between reference and hypothesised fluent transcripts, which leads to direct optimisation for lower WER. However, optimising for the intermediate output does not always help improve the overall output, and here the improved DD WER did not lead to further gain in its downstream GEC. On the other hand, tuning the GEC module (EMsemi-GEC) improved both SER and TER, since the fine-tuning process maximises the sentence-level probabilities, which helps to achieve

Model	ASR+DD WER↓	ASR+DD+GEC SER↓	ASR+DD+GEC TER↓
Base	21.20	76.76	27.89
EMsemi-DD	21.06	76.79	27.83
EMsemi-GEC	-	76.35	27.47
EMdistil-GEC	-	76.35	27.51

Table 7.7 Impact of the error mitigation training on the DD and GEC modules. Base: the vanilla cascaded SGEC system with performance shown in Table 7.6.

lower SER/TER[7]. To further improve the SGEC system, semi-supervised self-distillation was adopted to train the GEC model (EMdistil-GEC) to learn a probability distribution at each time step, as opposed to predicting a one-hot token. The rationale is that a probability distribution potentially offers richer information than a single prediction, especially when the reference is synthetically generated. The EMsemi-GEC system was adopted as the teacher model, and the student model was also initialised from the teacher. It is shown that SER/TER are quite insensitive to self-distillation, and further experiments in Section 7.6.7 showed that semi-supervised self-distillation helps improve the feedback performance.

7.6.5 Reranking

This section investigates the reranking approach discussed in Section 7.3. The baseline system is a cascaded concatenation of the hybrid ASR, BERT DD and Gramformer-based GEC, each trained in their corresponding domains. The idea is to train an external ranker that scores each correction candidate according to the reference metric, without having to retrain individual modules of the cascaded pipeline. Since the in-domain SGEC annotation is extremely limited, the reranker was trained purely on the written GEC corpora (CLC and BEA). A set of hypotheses was generated from the GEC module, and the reranker was trained to select the best correction candidate among all. All reranker results were interpolated with the GEC posterior to reach the final prediction unless specified otherwise.

Candidate generation

For each input, correction candidates can be generated from the GEC module using beam search decoding (beam width equals to the number of candidates N). Table 7.8 shows the impact of the number of candidates on the correction quality. Increasing N from 1 to 50 did

[7]Combining the DD and GEC fine-tuning did not yield better performance, and thus the EMsemi-GEC system was used for future experiments.

not give much gain to the top candidate with the highest model posterior (P_{GEC}), yet the top candidate according to the oracle score (Oracle) largely outperforms the MAP hypothesis.

N	Ranker	M^2 $F_{0.5}$ ↑	BLEU ↑
1	-	56.60	85.38
50	P_{GEC}	56.64	85.30
50	Oracle	85.08	94.85

Table 7.8 GEC performance with different numbers of candidates, evaluated on the FCEtst set. FCEtst set is used here since it is the closest in nature to the training data (both are non-native written English). P_{GEC}: the maximum-a-posterior (MAP) candidate according to GEC posterior, Oracle: using BLEU scores computed against reference corrections to choose the best candidate.

Candidates generated via beam search tend to be very similar, which will cause convergence failure when the reranker fails to extract meaningful differences among the candidates. Selective sampling and corruption approaches can be used to diversify the candidate set. Both approaches keep the higher quality half according to the reference scores, and replace the other half with alternative hypotheses. The sampling approach generates candidates by sampling from the posterior distribution at each decoding step of the sequence generation. The corruption approach selects a random number of words, and replaces them with their conjugate forms, which essentially adds artificial grammatical errors to the hypotheses. Table 7.9 compares the two diversification processing in terms of the average BLEU (aveBLEU) and crossBLEU scores (discussed in Section 6.5.1) of the final candidate set. The ideal scenario is to improve diversity (decrease crossBLEU) without affecting the average candidate quality (aveBLEU). Sampling increases both aveBLEU and crossBLEU. The Gramformer GEC module is a strong pre-trained language model with a highly biased posterior distribution, thus yielding very similar sampled candidates. The corruption process reduced crossBLEU by 13.1 without losing much on the average performance. The candidate set used for the following experiments was generated with a fixed beam width of 50, with the corruption approach adopted to diversify the candidate set for reranker training[8].

Reranker training and evaluation

For translation tasks, the model posterior does not usually align well with the reference ranking, whereas for GEC tasks, high quality hypotheses typically concentrate in the higher ranked region according to the model posterior.

[8]Candidate diversification was only adopted for training, and the trained reranker can be used to rank the full candidate set generated from beam search during inference.

Diversification	aveBLEU	crossBLEU
None	58.06	52.39
Sampling	74.28	80.01
Corruption	50.59	39.28

Table 7.9 Comparing the aveBLEU and crossBLEU before and after the diversification process of the candidate set ($N = 50$) on FCEtst.

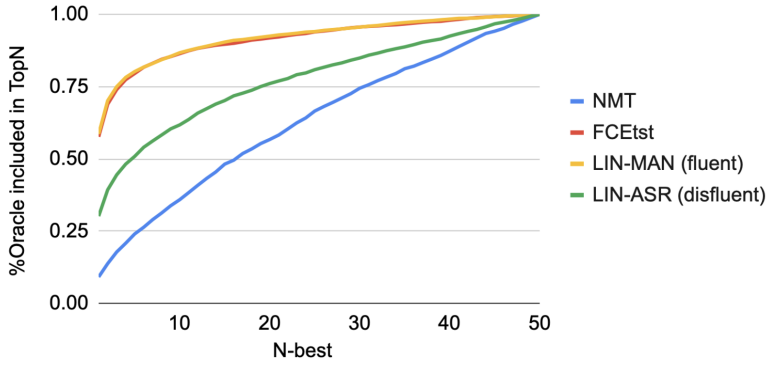


Fig. 7.10 Probability of the Oracle Top1 hypothesis appearing in the TopN hypotheses, i.e. Oracle error rate as a function of N-best depth. NMT: MuSTC En-De tst-COMMON set, with candidates generated using T5-based NMT (details described in Section 6.5). LIN-MAN: fluent manual transcriptions of LIN; LIN-ASR: disfluent transcriptions generated with the hybrid ASR module. The GEC candidates were generated using the Gramformer fine-tuned on CLC+BEA.

Given a candidate set with its corresponding reference sequence, the oracle candidate is the highest scored candidate when compared against the reference. Figure 7.10 shows the percentage of oracle candidates being included in the top N hypotheses, with N increasing along the x-axis. The oracle candidates for the NMT task spread quite evenly across the top N candidates, whereas for GEC, over 50-75% of the oracle candidates were included within the top 5 hypotheses. Rerankers rely on large pre-trained models, and increasing the number of hypotheses directly leads to larger memory usage. The distribution of the oracle hypotheses suggests that GEC rerankers can be trained more efficiently with fewer candidates.

The reranker assigns a scalar score to each candidate independently, thus the number of candidates can be set differently for training and evaluation. Table 7.10 shows how the number of candidates impacts the reranker performance. The rerankers adopted the RoBERTa

model for feature extraction, and were trained on the CLC+BEA corpora with the candidates generated using a beam width of 50. It is shown that training with the top 5 candidates gave the best performance, and expanding the candidate set to 50 at the inference stage further improved the $F_{0.5}$ score. The best performing system adopts $N_{\text{trn}} = 5$ and $N_{\text{eval}} = 50$, achieving a final $F_{0.5}$ score of 57.55. This confirms that it is not necessary to expose the reranker to a large candidate set during training, which helps save computational costs. Once the model is trained, it has the flexibility to be applied to a larger candidate set.

	N_{eval}		
N_{trn}	50	5	3
50	57.29	57.07	57.27
5	**57.55**	57.35	57.39
3	57.25	57.15	57.05

Table 7.10 RoBERTa reranker performance (in M^2 $F_{0.5}$) with a varying number of candidates during training N_{trn} and evaluation N_{eval}. Results are reported on the FCEtst set, with the baseline MAP candidate scored at 56.64 (shown in Table 7.8).

Adopting the best configuration with $N_{\text{trn}} = 5$ and $N_{\text{eval}} = 50$, Table 7.11 further evaluates the reranker performance on the out-of-domain spoken style LIN dataset. The fluent manual transcripts LIN-MAN is from a similar domain as the FCEtst set, and the reranker improved the $F_{0.5}$ score by 2.07 points. The LIN-ASR set was evaluated through the complete cascaded SGEC pipeline with reranking: the ASR transcripts were generated from the hybrid ASR, followed with BERT-based disfluency removal, and the correction candidates were generated with Gramformer-based GEC. The reranker reduced the TER by 0.27 from the MAP prediction. This shows that the reranker knowledge trained from the written domain is transferable to improve the cascaded system in the spoken domain.

7.6.6 Embedding passing

This section investigates three different modes of integration between the DD and GEC modules, namely multi-style (Multi), cascaded (Casc), and embedding passing (EP). The multi-style system trains a single sequence-to-sequence model using data from both DD and GEC domains. The main difference between the cascaded and the embedding passing systems lies in the modular connection: the Cascade system use words to connect two modules, whereas the EP system allows richer information flow with embeddings as the

Ranker	FCEtst M^2 $F_{0.5}$ ↑	LIN-MAN M^2 $F_{0.5}$ ↑	LIN-MAN TER ↓	LIN-ASR TER ↓
Baseline	56.60	53.57	8.27	27.89
P_{GEC}	56.64	54.35	8.18	27.81
RoBERTa	57.55	56.42	7.86	27.54
Oracle	85.08	87.07	2.15	18.83

Table 7.11 RoBERTa reranker performance on SGEC data. The reranker adopted $N_{\text{trn}} = 5$ and $N_{\text{eval}} = 50$, with the hypotheses generated from Gramformer-based GEC under a beam width of 50. LIN-MAN: fluent manual transcriptions of LIN, LIN-ASR: hybrid ASR transcriptions of LIN with BERT-based disfluency removal. Baseline: only consider 1 hypothesis from greedy search.

connection. RNN-based models are adopted for both the DD and GEC modules[9]. All results are reported on the BULATS dataset[10], and manual transcriptions are used throughout since the focus is on tighter integration between DD and GEC. As discussed in Section 7.6.1, the standard GEC metric M^2 $F_{0.5}$ are used to assess the DD and GEC modules combined, and the system input is fixed as the disfluent manual transcriptions.

Base systems

The base systems were trained on the SWBD and CLC corpora from the auxiliary DD and GEC domains respectively. The multi-style system was trained on the randomly shuffled combination of the two datasets. For the cascaded system, the DD and GEC modules were separately trained under their respective domains, and the DD module can take either of the sequence tagging (SeqTag) or sequence-to-sequence (Seq2seq) form. The embedding passing system adopts an embedding modular connection, which is constrained to use the Seq2seq style DD. As discussed in Section 7.4, the DD module in the EP system was exposed to both the disfluent native SWBD data and the fluent non-native CLC data, and the GEC module was trained on CLC. The performance of the base systems are listed in Table 7.12.

The multi-style system does not generate any intermediate fluent transcripts, and can only be assessed in terms of the output correction sentences. The cascaded system scored 3.95 $F_{0.5}$ higher than the multi-style system, which confirms that having a structured pipeline benefits the SGEC training under auxiliary data domains. Between the cascaded systems

[9]The embedding passing work was done before the introduction of large pre-trained Transformer-based models such as BERT and Gramformer, and therefore RNN-based models were used.

[10]This work was done before the LIN dataset becomes available, and thus BULATS was used for evaluation as well as cross-validation fine-tuning.

Model	DD style	DD $F_1 \uparrow$	DD+GEC M^2 $F_{0.5} \uparrow$
Multi	-	-	27.61
Cascade	SeqTag	53.94	31.56
	Seq2seq	44.63	24.86
EP	Seq2seq	37.13	29.30

Table 7.12 Base systems trained on out-of-domain SWBD and CLC corpora

with two different DD styles, the Seq2seq DD performed much worse than the SeqTag DD, and consequently degraded its downstream GEC performance. It is because the nature of the DD task requires only deletion, and the Seq2seq structure is inherently disadvantaged due to its flexibility in substitution and insertion operations. The deficiency in using SeqTag DD is also observed in the embedding passing framework, where the DD F_1 fell behind by 16.81 points compared with the SeqTag DD in the cascaded system. The correction outputs from the EP system scored 4.44 points higher than the cascaded system with the same Seq2seq DD, leaving a 2.66 gap from the best performing cascaded system with SeqTag DD. This result confirms that by allowing a richer information flow, the embedding connection helps compensate for the disadvantages posed by the Seq2seq DD, and potentially mitigates the errors made at earlier stages of the pipeline.

Fine-tuned systems

Without training under the target domain, the cascaded system with SeqTag DD performed the best. The word connection offers strong regularisation against domain mismatch, yet the information flow passed across modules is restricted to be discrete sequences. Although the embedding passing system suffers from trade-offs between the DD and GEC objectives, the continuous embedding connection shows potential benefit in mitigating error propagation. With a small amount of annotated target domain SGEC data (BULATS), it is possible to run fine-tuning with 10-fold cross-validation. There are two levels of annotation for each source sequence, one being disfluency tags, and the other being the target grammatically correct fluent sentence. For learner speech, it is often more difficult to tag each word with its error type, than simply generating a correction sequence. Therefore, fine-tuning was carried out without using reference disfluency tags, and the cascaded system was only fine-tuned with the best performing SeqTag DD. Table 7.13 shows the fine-tuning results on BULATS.

The embedding passing system outperformed the multi-style and cascaded systems by 3.13 and 1.20 $F_{0.5}$ respectively. Although trained end-to-end, the multi-style system was

Model	DD style	Base	Fine-tune
Multi	-	27.61	31.90
Cascade	SeqTag	**31.56**	33.83
EP	Seq2seq	29.30	**35.03**

Table 7.13 Comparing the Multi, Cascade and EP systems after 10-fold cross-validation fine-tuning on BULATS. Manual transcriptions were used both for fine-tuning and evaluation, and the DD+GEC results are reported in M^2 $F_{0.5}$ ↑.

constraint by the limited amount of parallel data due to lack of explicit DD modelling. In comparison, the EP system provides a more structured pipeline, which uses two separate attention mechanisms, each focusing on the DD and GEC tasks respectively. Fine-tuning individual modules in the cascaded system is difficult without reference for the intermediate disfluency tags, and only the GEC module was fine-tuned to the target domain[11]. On the contrary, the EP system can be easily adapted with different levels of annotations by switching the DD objective on or off. The embedding connection in the EP system encapsulates the full back history of the DD decoder, and it is confirmed that the richer information flow helps the system to swiftly adapt to the target domain under limited target domain data.

7.6.7 Feedback

Experiments in previous sections mainly focused on improving the correction output, and this section shifts the focus to analyse the feedback quality. The cascaded system consists of the hybrid ASR, the BERT-based DD and the Gramformer-based GEC modules discussed in Section 7.6.3. A slightly different operating point was chosen to optimise for feedback $F_{0.5}$ (according to Figure 7.9), which sets the LM scale factor at 11 and the DD threshold at 0.5. Table 7.14 tabulates the impact of the error mitigation approaches (discussed in Section 7.2) on the feedback $F_{0.5}$ scores. The semi-supervised fine-tuning (EMsemi) and self-distillation (EMdistil) approaches on the GEC module both benefited the feedback, respectively improving the $F_{0.5}$ score by 1.66 and 2.31. The optimal $F_{0.5}$ score of 22.57 is adopted as the baseline for the following experiments.

Excluding ambiguous errors

The GEC feedback usually suggests the error location, error type and the correction phrase. To give high quality feedback to language learners, it is important to pass on a clear and

[11]The inputs to the GEC module during fine-tuning was generated using DD from the base system.

Models	Base	EMsemi-GEC	EMdistil-GEC
Feedback $F_{0.5}\uparrow$	20.26	21.92	22.57

Table 7.14 Impact of the error mitigation approaches on feedback $F_{0.5}$. EMsemi-GEC: semi-supervised fine-tuning on GEC, EMdistil-GEC: semi-supervised self-distillation on GEC

accurate message in terms of the error type and suggested correction. Feedback edits are automatically typed using a rule-based framework ERRANT [24, 54] in this thesis. To give some examples of the error types[12]: 'R:PREP' means replacement of preposition, 'M:PREP' means missing preposition, 'U:DET' means unnecessary determiner. If the edits do not fall into any other category, they will be typed as 'OTHER', and a large part of the 'OTHER' errors are paraphrases. To reduce ambiguity and give a clear feedback to learners, the edits typed as 'OTHER' are excluded from the feedback.

	$F_{0.5}\uparrow$		%Edits Excluded	
Eval	Include	Exclude	REF	HYP
FCEtst	56.60	59.73	13.87	9.65
LIN	22.57	24.30	14.21	12.54

Table 7.15 Feedback $F_{0.5}$ before and after excluding 'OTHER', and percentage edits being excluded from the reference and hypothesis by excluding the 'OTHER' type.

Table 7.15 show the impact of removing the 'OTHER' errors. Approximately 10-15% edits were excluded from the reference and hypothesis edits. Removing 'OTHER' in reference reduced the total number of edits, and made it much easier for the model to achieve a higher $F_{0.5}$, since most rejected edits are ambiguous and difficult to predict. Both the precision and recall scores increased, thus improving the baseline $F_{0.5}$. All 'OTHER' errors were excluded from scoring for the following confidence filtering experiments.

Confidence filtering

To further improve the feedback precision, sentence-level and edit-level confidence filtering were applied to reject ill-conditioned edits (discussed in Section 7.5). When conducting the filtering, both true positives (TP: correctly predicted edits) and false positives (FP: incorrectly

[12]There are three prefixes, namely R: replacement, U: unnecessary, M: missing. The error types are defined using part-of-speech (POS) tags. More details on edit types can be found in Bryant et al. 24.

predicted edits) will reduce. Assuming that there are more FPs than TPs in the low confidence region, it is expected that filtering will improve the precision score, and will consequently boost up $F_{0.5}$. Figure 7.11 shows the change in feedback $F_{0.5}$ score with an increasing number of edits being filtered out under an increasing confidence threshold. Both sentence-level and edit-level filtering peaked midway (peak: operating point), and drop back as more edits got filtered out. The sentence-level filtering worked better than the edit-level filtering. This can be explained by the nature of the grammatical corrections being intertwined within one sentence, i.e. removing one edit will potentially result in inconsistencies with other corrections made to the sentence.

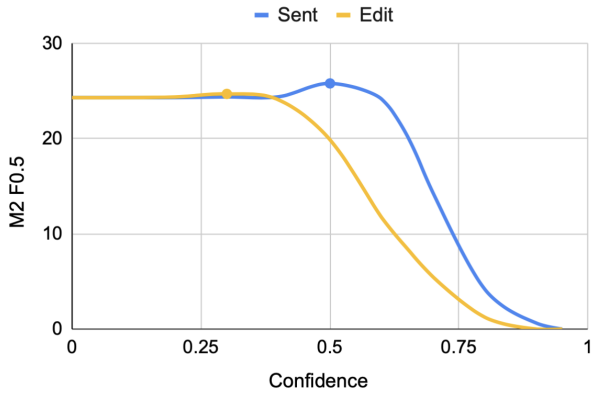


Fig. 7.11 Comparing the sentence-level and edit-level confidence filtering. Moving from left to right, more edits get filtered out with an increasing confidence threshold.

Table 7.16 shows the feedback performance at the operating points. Removing 33.8% of the edits using sentence-level confidence led to significant gains in both the precision and $F_{0.5}$ scores; whereas edit-level filtering gave mild improvement with 3.7% removal. When the SGEC system gets deployed, the confidence threshold can be adjusted according to desired level of percentage edit removal and the precision score.

Filter	P	R	$F_{0.5}$	%Remove
None	27.75	16.24	24.30	0
Sent	33.96	13.15	25.80	33.8
Edit	28.53	16.05	24.69	3.7

Table 7.16 The operating points of confidence filtering. P: precision, R: recall, %Remove: percentage edits being removed, None: no filtering.

The system confidence combines the confidence scores from the ASR, DD and GEC modules (see Equation 7.22). The coefficients for confidence combination were selected using the optimal feedback $F_{0.5}$ score, which gives $\alpha = 0.3, \beta = 0.4, \gamma = 0.3$. Figure 7.12 contrasts the filtering results with the sentence-level confidence of each individual module. With an increasing number of edits gets filtered out, the precision-recall curve moves from the bottom right to the top left corner, and having a larger area under the curve indicates higher $F_{0.5}$ scores throughout the sweep. Filtering with P_a outperforms both P_d and P_g, and filtering with the combined confidence P_{comb} led to the highest $F_{0.5}$. This observation suggests that the ASR confidence is quite indicative of the feedback quality, and also confirms that filtering with the combined confidence score is an effective approach in improving the feedback quality of the cascaded SGEC system.

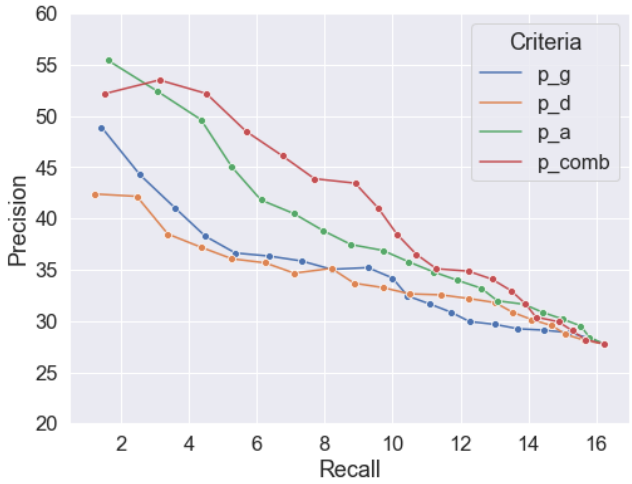


Fig. 7.12 Precision and recall curves with feedback filtering using sentence-level confidences of individual modules. P_{comb}: combined confidence, P_a: ASR, P_d: DD, P_g: GEC. From right to left, an increasing number of edits are filtered out.

Table 7.17 further analyses the impact of the sentence-level filtering on different edit types, showing the change in precision, recall and $F_{0.5}$ scores before and after filtering. Confidence filtering improved feedback $F_{0.5}$ on most edit types, among which the most significant ones are 'R:PREP', 'U:DET', 'M:PREP'. The two degraded edit types are 'R:VERB:TENSE', 'R:VERB:FORM', both of which often have more than one feasible corrections. Confidence-based filtering tends to remove edits with diverse solutions, due to the high entropy and low confidence in the hypotheses. This rejection pattern will lead to a significant drop in recall, and thus reduced $F_{0.5}$ of edits with diverse corrections. On the other hand, for edits like 'R:PREP', 'U:DET', 'M:PREP', there usually exists a single, definite fix. Baseline $F_{0.5}$ scores on those edits are in general quite high, and confidence filtering helps to further improve

the performance. This observation suggests that applying confidence filtering helps reduce feedback on ambiguous edits, and further boosts precision on more deterministic corrections.

	No filtering			Sentence-level filtering		
Edit Type	P	R	$F_{0.5}$	P	R	$F_{0.5}$
M:DET	30.18	27.39	29.57	35.86	22.61	**32.10**
R:PREP	37.86	18.47	31.29	46.88	15.68	**33.53**
R:NOUN:NUM	37.88	20.66	32.47	44.68	17.36	**33.98**
R:VERB:TENSE	35.63	13.60	**26.91**	35.00	9.21	22.44
U:DET	23.20	16.20	21.36	29.49	12.85	**23.42**
R:VERB	27.27	11.69	21.53	33.33	10.39	**23.12**
R:NOUN	11.77	4.29	8.72	18.52	3.57	**10.08**
M:PREP	23.29	13.39	20.29	35.56	12.60	**26.06**
R:VERB:FORM	31.17	20.00	**28.04**	38.10	13.33	27.78
R:VERB:SVA	31.92	27.78	30.99	37.31	23.15	**33.25**

Table 7.17 Comparing P, R, $F_{0.5}$ scores before and after sentence-level confidence filtering, with breakdown in terms of edit types.

7.7 Summary

This chapter investigated multimodular combination for the spoken grammatical error correction task. The cascaded SGEC system consists of the ASR, DD and GEC modules, and different metrics were discussed to evaluate different combinations of modules. As shown in Table 7.7, with partially annotated non-native spoken corpora, semi-supervised error mitigation effectively improved the quality of the output correction sentences. The reranking approach adopts an external reranker trained from the auxiliary written GEC domain. As seen in Table 7.10, reranking is helpful in further improving the cascaded system in the unseen SGEC domain. The embedding passing approach was adopted to encourage tighter integration between the DD and GEC modules. As seen in Table 7.12, the rich information flow via the continuous embedding connection helped the system more efficiently adapt to the target domain, under limited quantities of target domain data. Furthermore, confidence filtering was adopted to improve the feedback quality of the cascaded SGEC system. With the combined confidence score from individual modules, Table 7.16 showed that the sentence-level confidence filtering was beneficial in reducing ambiguous feedback, boosting precision on more deterministic corrections (see Table 7.17), and thus improving the overall feedback quality.

Chapter 8

Conclusion

This thesis investigated multimodular combination of spoken language processing tasks. Due to the lack of end-to-end training corpora, spoken language tasks are often formulated in a cascaded multimodular fashion. Loosely cascaded structure allows individual modules to be trained in their corresponding domains, yet the simplified structure gives rise to several challenges: loss of information due to discrete modular connections; error propagation due to early decisions; as well as domain mismatches across modules. This thesis has examined the general concept of multimodular combination, and explored approaches to overcome the limitations of the vanilla cascaded structure under limited end-to-end data.

Chapter 1 introduced the scope of this thesis and the research questions. Fundamentals of deep neural networks were reviewed in Chapter 2, and individual module formulations of spoken language tasks were introduced in Chapter 3. Chapter 4 discussed the challenges in cascaded spoken language systems, and discussed several multimodular combination approaches. Chapter 5, 6 and 7 respectively investigated multimodular combination on spoken disfluency detection (SDD), spoken language translation (SLT), and spoken grammatical error correction (SGEC) tasks. This chapter provides a brief summary of the contributions of this work, and presents several potential directions for future research.

8.1 Summary

The first contribution of this thesis is to examine the general concept of multimodular combination. Three forms of spoken language systems are considered in Chapter 4 with an increasing level of module integration: namely cascaded, integrated and end-to-end systems. More tightly integrated systems generally require larger amount of end-to-end training data, and different approaches are discussed aiming to strike a balance between modelling power and data efficiency. Cascaded systems train individual modules in their corresponding

domains, and do not require any end-to-end corpora. Loosely coupled modules suffer from information loss, domain mismatch and error propagation issues, and thus domain adaptation and error mitigation approaches are discussed to address these issues. Integrated systems seek tighter modular integration, and require a small amount of end-to-end data to adapt to the target domain. Discrete information passing and embedding passing approaches are proposed to propagate richer information in different forms across the modular connection. End-to-end systems do not rely on intermediate variables to aid training, and require large amount of end-to-end data to reach convergence. Data augmentation and meta-learning approaches are discussed to overcome the low data efficiency issue.

The second contribution of this thesis is to propose a general framework of reranking for multimodular systems, addressing the error propagation and information loss issues in the vanilla cascaded system. Rerankers are commonly used for single module systems such as speech recognition and machine translation. In Section 4.3.1.2, rerankers are generalised to multimodular systems, where they directly access the hypothesis space of the modular connection. Taking into account multiple hypotheses of the modular connection allows a richer information flow and helps reduce error propagation.

The spoken language translation (SLT) experiments in Section 6.5.5 applied an external reranker to a cascaded SLT system. The ASR and NMT modules were initialised in their respective domains, and the reranker was trained using an end-to-end corpus. Experiments have shown that adopting an external reranker helped adjust the model posterior to better align with the evaluation metric, and further incorporating multiple ASR hypotheses led to additional gains. When combining multiple ASR hypotheses into reranking, using an attention mechanism tends to be more effective than simple concatenation. The attention mechanism assigns larger weights to the most relevant context, and thus extracts better feature representation for the reranking process. The spoken grammatical error correction (SGEC) experiments in Section 7.6.5 applied reranking to the GEC module of the cascaded SGEC system. The ASR, DD and GEC modules of the cascaded pipeline were initialised with their corresponding corpora, and the reranker was trained on a written GEC corpus due to the limited target domain SGEC data. It is shown that training the reranker under an auxiliary domain also helps improve the performance in the target domain.

The third contribution of this thesis is to propose embedding passing. Introduced in Section 4.3.2, the idea of embedding passing is to extract continuous feature representations of the upstream context, and use them as the modular connection to pass richer information to the downstream modules. Another advantage is that embeddings allow gradient backpropagation across modules, thus enabling joint optimisation of the multimodular system.

The spoken disfluency detection (SDD) experiments in Section 5.3.4 extracted acoustic embeddings from the upstream speech inputs to aid the downstream disfluency tagging process. It is shown that propagating acoustic information through embedding passing consistently improved the disfluency removal performance. The hard embeddings extracted from the ASR timestamps yielded smaller gains compared with the soft embeddings extracted using a global attention mechanism over the speech input. It is because the soft attention is more flexible in attending to cross-over regions between neighbouring words, which are particularly informative to the disfluency detection task. The spoken language translation (SLT) experiments in Section 6.5.6 contrasted the data efficiency of the embedding passing system with the vanilla cascaded system. By providing a richer acoustic context, embedding passing helped mitigate error propagation, and also allowed more efficient domain adaptation under limited amount of end-to-end data. The embedding passing performance can be further improved by incorporating an external language model or high quality ASR transcriptions. The spoken grammatical error correction (SGEC) experiments in Section 7.6.6 adopted embedding passing to achieve tighter integration between the DD and GEC modules. When trained under auxiliary domains, the embedding passing system helped mitigate the errors made at the disadvantaged DD module. After fine-tuned under the target domain, the embedding passing system outperformed both the end-to-end trained multi-style system and the vanilla cascaded system. It shows that the richer information flow via the embedding connection allows efficient adaptation under limited target domain data.

8.2 Future work

This thesis has mainly experimented with the module combination approaches on cascaded and integrated systems. Future experiments may further explore the data augmentation and meta-learning approaches on end-to-end systems, as discussed in Section 4.4. The spoken grammatical error correction experiments were conducted on extremely limited target domain data, and it will be interesting to extend the current analyses when more target domain annotations become available.

The reranking approach in Section 4.3.1.2 only considered text sequences for reranker feature extraction. Future research may explore the more general form, where the scoring function can be extended to account for the input speech signals, and further extracts the correlations between the speech input and its associated downstream textual information. The current reranking approach adopts large pre-trained language models for feature extraction. The number of candidates being considered is largely restricted by the high memory usage.

Future work may adopt model compression techniques [31, 164] to compress the pre-trained language models, and further account for a larger candidate set during reranking.

With the recent advances on self-supervised speech representation learning [4, 88], it is possible to extract higher quality speech representations, as well as train a more powerful speech recognition module. Future research may explore the interactions between the pre-trained acoustic models with the proposed multimodular combination approaches. It is not known whether a better speech recognition module would reduce the gap between the vanilla cascade and the more tightly integrated systems. It will also be interesting to see to which extent the higher quality speech representations would benefit the spoken language tasks.

References

[1] Anastasopoulos, A. and Chiang, D. (2018). Tied multitask learning for neural speech translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 82–91.

[2] Awasthi, A., Sarawagi, S., Goyal, R., Ghosh, S., and Piratla, V. (2019). Parallel iterative edit models for local sequence transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4259–4269, Hong Kong, China. Association for Computational Linguistics.

[3] Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *ArXiv preprint arXiv:1607.06450*.

[4] Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460.

[5] Bahar, P., Bieschke, T., Schlüter, R., and Ney, H. (2021). Tight integrated end-to-end training for cascaded speech translation. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 950–957. IEEE.

[6] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[7] Bahl, L. R., Jelinek, F., and Mercer, R. L. (1983). A maximum likelihood approach to continuous speech recognition. *IEEE transactions on pattern analysis and machine intelligence*, (2):179–190.

[8] Baker, J. (1975). The DRAGON system – An overview. *IEEE Transactions on Acoustics, speech, and signal Processing*, 23(1):24–29.

[9] Bansal, S., Kamper, H., Livescu, K., Lopez, A., and Goldwater, S. (2018). Low-resource speech-to-text translation. *Proc. Interspeech 2018*, pages 1298–1302.

[10] Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. (2010). A theory of learning from different domains. *Machine learning*, 79(1):151–175.

[11] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in neural information processing systems*, 28.

[12] Bengio, Y. (2009). Learning deep architectures for AI. *Machine Learning*, 2(1):1–127.

[13] Bengio, Y., Courville, A., and Vincent, P. (2013a). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[14] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

[15] Bengio, Y., Léonard, N., and Courville, A. (2013b). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

[16] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[17] Bentivogli, L., Cettolo, M., Gaido, M., Karakanta, A., Martinelli, A., Negri, M., and Turchi, M. (2021). Cascade versus direct speech translation: Do the differences still make a difference? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2873–2887.

[18] Bhattacharyya, S., Rooshenas, A., Naskar, S., Sun, S., Iyyer, M., and McCallum, A. (2021). Energy-based reranking: Improving neural machine translation using energy-based models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4528–4537.

[19] Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huang, S., Huck, M., Koehn, P., Liu, Q., Logacheva, V., et al. (2017). Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214.

[20] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

[21] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[22] Bryant, C., Felice, M., Andersen, Ø. E., and Briscoe, T. (2019). The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75.

[23] Bryant, C., Felice, M., and Briscoe, E. (2017a). Automatic annotation and evaluation of error types for grammatical error correction. In *Association for Computational Linguistics*.

[24] Bryant, C., Felice, M., and Briscoe, T. (2017b). Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 793–805, Vancouver, Canada. Association for Computational Linguistics.

[25] Caruana, R., Lawrence, S., and Giles, C. (2000). Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 13.

[26] Chambers, L. and Ingham, K. (2011). The BULATS online speaking test. *Research Notes*, 43(1):21–25.

[27] Chan, W., Jaitly, N., Le, Q., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *2016 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4960–4964. IEEE.

[28] Charniak, E. and Johnson, M. (2001). Edit detection and parsing for transcribed speech. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

[29] Chen, X., Liu, X., Ragni, A., Wang, Y., and Gales, M. J. (2017). Future word contexts in neural network language models. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 97–103. IEEE.

[30] Cheng, J., Dong, L., and Lapata, M. (2016). Long Short-Term Memory-networks for machine reading. In *EMNLP*.

[31] Cheng, Y., Wang, D., Zhou, P., and Zhang, T. (2018). Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136.

[32] Chiu, C.-C. and Raffel, C. (2018). Monotonic chunkwise attention. In *International Conference on Learning Representations*.

[33] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

[34] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*.

[35] Chollampatt, S., Wang, W., and Ng, H. T. (2019). Cross-sentence grammatical error correction. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 435–445.

[36] Chorowski, J. and Jaitly, N. (2017). Towards better decoding and language model integration in sequence to sequence models. *Proc. Interspeech 2017*, pages 523–527.

[37] Cieri, C., Miller, D., and Walker, K. (2004). The Fisher corpus: A resource for the next generations of speech-to-text. In *LREC*, volume 4, pages 69–71.

[38] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

[39] Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–70.

[40] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, É., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020). Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451.

[41] Conneau, A. and Lample, G. (2019). Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32.

[42] Council of Europe (2001). *Common European framework of reference for languages: Learning, teaching, assessment*. Cambridge University Press.

[43] Dahlmeier, D. and Ng, H. T. (2012). Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572.

[44] Dale, R. and Kilgarriff, A. (2011). Helping our own: The HOO 2011 pilot shared task. In *Proceedings of the 13th European Workshop on Natural Language Generation*, pages 242–249.

[45] Damodaran, P. (2021). Gramformer. *URL: https://github.com/PrithivirajDamodaran/Gramformer*.

[46] De Mori, R., Bechet, F., Hakkani-Tur, D., McTear, M., Riccardi, G., and Tur, G. (2008). Spoken language understanding. *IEEE Signal Processing Magazine*, 25(3):50–58.

[47] Di Gangi, M. A., Cattoni, R., Bentivogli, L., Negri, M., and Turchi, M. (2019). Must-c: a multilingual speech translation corpus. In *2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2012–2017. Association for Computational Linguistics.

[48] Dong, Y., Li, Z., Rezagholizadeh, M., and Cheung, J. C. K. (2019). EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3393–3402.

[49] Dou, Q., Efiong, J., and Gales, M. J. (2020). Attention forcing for speech synthesis. In *Interspeech*, pages 4014–4018.

[50] Dou, Q., Lu, Y., Manakul, P., Wu, X., and Gales, M. J. (2021). Attention forcing for machine translation. *arXiv preprint arXiv:2104.01264*.

[51] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

[52] Duckworth, D., Neelakantan, A., Goodrich, B., Kaiser, L., and Bengio, S. (2019). Parallel scheduled sampling. *arXiv preprint arXiv:1906.04331*.

[53] Evermann, G., Chan, H. Y., Gales, M. J., Jia, B., Mrva, D., Woodland, P. C., and Yu, K. (2005). Training LVCSR systems on thousands of hours of data. In *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 1, pages I–209. IEEE.

[54] Felice, M., Bryant, C., and Briscoe, T. (2016). Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 825–835, Osaka, Japan. The COLING 2016 Organizing Committee.

[55] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

[56] Fügen, C. (2009). *A system for simultaneous translation of lectures and speeches*. PhD thesis, University of Karlsruhe.

[57] Gales, M., Young, S., et al. (2008). The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304.

[58] Gales, M. J. (1998). Maximum likelihood linear transformations for HMM-based speech recognition. *Computer speech & language*, 12(2):75–98.

[59] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR.

[60] Gibson, M. and Hain, T. (2006). Hypothesis spaces for minimum Bayes risk training in large vocabulary speech recognition. In *Interspeech*, volume 6, pages 2406–2409. Citeseer.

[61] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

[62] Godfrey, J. J., Holliman, E. C., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, volume 1, pages 517–520. IEEE Computer Society.

[63] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[64] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch SGD: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*.

[65] Graves, A. (2012). Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*.

[66] Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376.

[67] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772. PMLR.

[68] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *ArXiv*, abs/1410.5401.

[69] Grundkiewicz, R. and Junczys-Dowmunt, M. (2014). The wiked error corpus: A corpus of corrective wikipedia edits and its application to grammatical error correction. In *International Conference on Natural Language Processing*, pages 478–490. Springer.

[70] Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., Wu, Y., et al. (2020). Conformer: Convolution-augmented transformer for speech recognition. *Proc. Interspeech 2020*, pages 5036–5040.

[71] Guo, Q., Qiu, X., Liu, P., Shao, Y., Xue, X., and Zhang, Z. (2019). Star-transformer. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1315–1325.

[72] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

[73] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[74] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[75] He, Y., Sainath, T. N., Prabhavalkar, R., McGraw, I., Alvarez, R., Zhao, D., Rybach, D., Kannan, A., Wu, Y., Pang, R., et al. (2019). Streaming end-to-end speech recognition for mobile devices. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6381–6385. IEEE.

[76] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

[77] Heinzerling, B. and Strube, M. (2018). BPEmb: Tokenization-free pre-trained subword embeddings in 275 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

[78] Hernandez, F., Nguyen, V., Ghannay, S., Tomashenko, N., and Esteve, Y. (2018). TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker adaptation. In *International conference on speech and computer*, pages 198–208. Springer.

[79] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a local Nash equilibrium. *Advances in neural information processing systems*, 30.

[80] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97.

[81] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *stat*, 1050:9.

[82] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: The difficulty of learning long-term dependencies.

[83] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[84] Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. (2019). The curious case of neural text degeneration. In *International Conference on Learning Representations*.

[85] Honnibal, M. and Johnson, M. (2014). Joint incremental disfluency detection and dependency parsing. *Transactions of the Association for Computational Linguistics*, 2:131–142.

[86] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[87] Hough, J. and Schlangen, D. (2015). Recurrent neural networks for incremental disfluency detection. In *INTERSPEECH*.

[88] Hsu, W.-N., Bolte, B., Tsai, Y.-H. 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*, 29:3451–3460.

[89] Huang, X., Acero, A., Hon, H.-W., and Reddy, R. (2001). *Spoken language processing: A guide to theory, algorithm, and system development*. Prentice hall PTR.

[90] Hutchins, W. J. (1995). Machine translation: A brief history. In *Concise history of the language sciences*, pages 431–445. Elsevier.

[91] Imamura, K. and Sumita, E. (2017). Ensemble and reranking: Using multiple models in the nict-2 neural machine translation system at wat2017. In *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pages 127–134.

[92] Inaguma, H., Yan, B., Dalmia, S., Guo, P., Shi, J., Duh, K., and Watanabe, S. (2021). ESPnet-ST IWSLT 2021 offline speech translation system. In *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*, pages 100–109.

[93] Indurthi, S., Han, H., Lakumarapu, N. K., Lee, B., Chung, I., Kim, S., and Kim, C. (2020). End-end speech-to-text translation with modality agnostic meta-learning. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7904–7908. IEEE.

[94] Izmailov, P., Wilson, A., Podoprikhin, D., Vetrov, D., and Garipov, T. (2018). Averaging weights leads to wider optima and better generalization. In *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, pages 876–885.

[95] Jan, N., Cattoni, R., Sebastian, S., Negri, M., Turchi, M., Elizabeth, S., Ramon, S., Loic, B., Lucia, S., and Federico, M. (2019). The IWSLT 2019 evaluation campaign. In *16th International Workshop on Spoken Language Translation 2019*.

[96] Jang, E., Gu, S. S., and Poole, B. (2017). Categorical reparameterization with Gumbel-Softmax. *In International Conference on Learning Representations, ICLR 2017*.

[97] Jia, Y., Johnson, M., Macherey, W., Weiss, R. J., Cao, Y., Chiu, C.-C., Ari, N., Laurenzo, S., and Wu, Y. (2019). Leveraging weakly supervised data to improve end-to-end speech-to-text translation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7180–7184. IEEE.

[98] John J. Godfrey, E. H. (1993). Switchboard-1 Release 2. *Philadelphia: Linguistic Data Consortium. URL: https://catalog.ldc.upenn.edu/LDC97S62*.

[99] Johnson, M. and Charniak, E. (2004). A TAG-based noisy-channel model of speech repairs. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 33–39.

[100] Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F., Wattenberg, M., Corrado, G., et al. (2017). Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

[101] Junczys-Dowmunt, M. and Grundkiewicz, R. (2016). Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1546–1556.

[102] Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. *URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/*.

[103] Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401.

[104] Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

[105] Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *ICLR (Poster)*.

[106] Kneser, R. and Ney, H. (1995). Improved backing-off for M-gram language modeling. *1995 International Conference on Acoustics, Speech, and Signal Processing*, 1:181–184 vol.1.

[107] Koehn, P. (2009). *Statistical machine translation*. Cambridge University Press.

[108] Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., et al. (2007). Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180.

[109] Kombrink, S., Mikolov, T., Karafiát, M., and Burget, L. (2011). Recurrent neural network based language modeling in meeting recognition. In *Interspeech*, volume 11, pages 2877–2880.

[110] Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

[111] Kumar, S. and Byrne, W. (2004). Minimum Bayes-risk decoding for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 169–176.

[112] Kurata, G., Audhkhasi, K., and Kingsbury, B. (2019). IBM Research advances in end-to-end speech recognition at INTERSPEECH 2019. *URL: https://www.ibm.com/blogs/research/2019/10/end-to-end-speech-recognition/*.

[113] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[114] Lamb, A. M., ALIAS PARTH GOYAL, A. G., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29.

[115] Lamel, L., Gauvain, J.-L., and Adda, G. (2002). Lightly supervised and unsupervised acoustic model training. *Computer Speech & Language*, 16(1):115–129.

[116] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. In *Proceedings of NAACL-HLT*, pages 260–270.

[117] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). ALBERT: A lite bert for self-supervised learning of language representations.

[118] Lanchantin, P., Gales, M. J., Karanasou, P., Liu, X., Qian, Y., Wang, L., Woodland, P. C., and Zhang, C. (2016). Selection of multi-genre broadcast data for the training of automatic speech recognition systems. In *Interspeech*, volume 2016, pages 3057–3061.

[119] Lavie, A., Gates, D., Gavalda, M., Tomokiyo, L. M., Waibel, A., and Levin, L. (1996). Multi-lingual translation of spontaneously spoken language in a limited domain. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

[120] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[121] LeCun, Y., Kanter, I., and Solla, S. (1990). Second order properties of error surfaces: Learning time and generalization. *Advances in neural information processing systems*, 3.

[122] Lee, A., Auli, M., and Ranzato, M. (2021). Discriminative reranking for neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7250–7264.

[123] Leng, Y., Tan, X., Wang, R., Zhu, L., Xu, J., Liu, W., Liu, L., Li, X.-Y., Qin, T., Lin, E., et al. (2021). FastCorrect 2: Fast error correction on multiple candidates for automatic speech recognition. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4328–4337.

[124] Leshno, M., Lin, V. Y., Pinkus, A., and Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867.

[125] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.

[126] Li, J. (2021). Recent advances in end-to-end automatic speech recognition. *arXiv preprint arXiv:2111.01690*.

[127] Li, J., Zhao, R., Meng, Z., Liu, Y., Wei, W., Parthasarathy, S., Mazalov, V., Wang, Z., He, L., Zhao, S., et al. (2020). Developing RNN-T models surpassing high-performance hybrid models with customization capability.

[128] Li, Y., Wei, C., and Ma, T. (2019). Towards explaining the regularization effect of initial large learning rate in training neural networks. *Advances in Neural Information Processing Systems*, 32.

[129] Lichtarge, J., Alberti, C., Kumar, S., Shazeer, N., Parmar, N., and Tong, S. (2019). Corpora generation for grammatical error correction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3291–3301.

[130] Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luís, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530.

[131] Litman, D. J. and Pan, S. (2002). Designing and evaluating an adaptive spoken dialogue system. *User Modeling and User-Adapted Interaction*, 12(2):111–137.

[132] Liu, X., Chen, X., Wang, Y., Gales, M. J., and Woodland, P. C. (2016). Two efficient lattice rescoring methods using recurrent neural network language models. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(8):1438–1449.

[133] Liu, X., Li, M., Chen, L., Wanigasekara, P., Ruan, W., Khan, H., Hamza, W., and Su, C. (2021). ASR n-best fusion nets. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7618–7622. IEEE.

[134] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[135] Liu, Y., Zhou, L., Wang, Y., Zhao, Y., Zhang, J., and Zong, C. (2018). A comparable study on model averaging, ensembling and reranking in nmt. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 299–308. Springer.

[136] Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *EMNLP*.

[137] Maas, A., Xie, Z., Jurafsky, D., and Ng, A. Y. (2015). Lexicon-free conversational speech recognition with neural networks. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 345–354.

[138] Maas, A. L., Hannun, A. Y., Ng, A. Y., et al. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Citeseer.

[139] Mallinson, J., Severyn, A., Malmi, E., and Garrido, G. (2020). FELIX: Flexible text editing through tagging and insertion. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1244–1255.

[140] McCann, B., Bradbury, J., Xiong, C., and Socher, R. (2017). Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30.

[141] McCann, B., Keskar, N. S., Xiong, C., and Socher, R. (2018). The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

[142] Meteer, M., Taylor, A., MacIntyre, R., and Iyer, R. (1995). Disfluency Annotation Stylebook for the Switchboard Corpus. Technical report, Linguistic Data Consortium. updated June 1995 by Ann Taylor.

[143] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *ICLR*.

[144] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari.

[145] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

[146] Mohammadi, S. H. and Kain, A. (2017). An overview of voice conversion systems. *Speech Communication*, 88:65–82.

[147] Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT press.

[148] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *ICML*.

[149] Napoles, C., Sakaguchi, K., Post, M., and Tetreault, J. (2015). Ground truth for grammatical error correction metrics. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 588–593.

[150] Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., and Shaalan, K. (2019). Speech recognition using deep neural networks: A systematic review. *IEEE access*, 7:19143–19165.

[151] Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

[152] Ney, H. (1999). Speech translation: Coupling of recognition and translation. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, volume 1, pages 517–520. IEEE.

[153] Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., and Bryant, C. (2014). The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14.

[154] Ng, N., Yee, K., Baevski, A., Ott, M., Auli, M., and Edunov, S. (2019). Facebook FAIR's WMT19 news translation task submission. *WMT 2019*, page 314.

[155] Nicholls, D. (2003). The Cambridge Learner Corpus: Error coding and analysis for lexicography and ELT. In *Proc. of the Corpus Linguistics 2003 conference; UCREL technical paper number 16.*

[156] Normandin, Y., Lacouture, R., and Cardin, R. (1994). MMIE training for large vocabulary continuous speech recognition. In *ICSLP*, pages 1367–1370.

[157] Omelianchuk, K., Atrasevych, V., Chernodub, A., and Skurzhanskyi, O. (2020). GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170.

[158] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

[159] Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *EMNLP*.

[160] 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. *Proc. Interspeech 2019*, pages 2613–2617.

[161] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[162] Pino, J., Puzon, L., Gu, J., Ma, X., McCarthy, A. D., and Gopinath, D. (2019a). Harnessing indirect training data for end-to-end automatic speech translation: Tricks of the trade. *In proceedings of IWSLT 2019*.

[163] Pino, J. M., Puzon, L., Gu, J., Ma, X., McCarthy, A. D., and Gopinath, D. P. (2019b). Harnessing indirect training data for end-to-end automatic speech translation: Tricks of the trade. *International Workshop on Spoken Language Translation (IWSLT)*.

[164] Polino, A., Pascanu, R., and Alistarh, D. (2018). Model compression via distillation and quantization. In *International Conference on Learning Representations*.

[165] Povey, D., Cheng, G., Wang, Y., Li, K., Xu, H., Yarmohammadi, M., and Khudanpur, S. (2018). Semi-orthogonal low-rank matrix factorization for deep neural networks. In *Interspeech*, pages 3743–3747.

[166] Povey, D., Peddinti, V., Galvez, D., Ghahremani, P., Manohar, V., Na, X., Wang, Y., and Khudanpur, S. (2016). Purely sequence-trained neural networks for ASR based on lattice-free MMI. In *Interspeech*, pages 2751–2755.

[167] Powers, D. (2011). Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2:37–63.

[168] 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 *Interspeech*, pages 939–943.

[169] Qian, X. and Liu, Y. (2013). Disfluency detection using multi-step stacked learning. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 820–825.

[170] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[171] Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

[172] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[173] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020a). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

[174] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020b). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.

[175] Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2016). Sequence level training with recurrent neural networks. *CoRR*, abs/1511.06732.

[176] Redko, I., Morvant, E., Habrard, A., Sebban, M., and Bennani, Y. (2019). *Advances in domain adaptation theory*. Elsevier.

[177] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[178] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[179] Saboo, A. and Baumann, T. (2019). Integration of dubbing constraints into machine translation. In *Proceedings of the Fourth Conference on Machine Translation (Volume 1: Research Papers)*, pages 94–101.

[180] Sainath, T. N., He, Y., Li, B., Narayanan, A., Pang, R., Bruguier, A., Chang, S.-y., Li, W., Alvarez, R., Chen, Z., et al. (2020). A streaming on-device end-to-end model surpassing server-side conventional model quality and latency. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6059–6063. IEEE.

[181] Salazar, J., Liang, D., Nguyen, T. Q., and Kirchhoff, K. (2020). Masked language model scoring. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2699–2712.

[182] Saleem, S., Jou, S.-C., Vogel, S., and Schultz, T. (2004). Using word lattice information for a tighter coupling in speech translation systems. In *Proc. Int. Conf. on Spoken Language Processing*, pages 41–44.

[183] Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanowska, J., Stefaniak, I., Jarkiewicz, M., and Okruszek, L. (2021). Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135.

[184] Schmaltz, A., Kim, Y., Rush, A. M., and Shieber, S. M. (2017). Adapting sequence models for sentence correction. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2807–2813.

[185] Schuster, M. and Nakajima, K. (2012). Japanese and Korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE.

[186] Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

[187] Shen, S., Cheng, Y., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. (2016). Minimum risk training for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1683–1692.

[188] Shen, T., Ott, M., Auli, M., and Ranzato, M. (2019). Mixture models for diverse machine translation: Tricks of the trade. In *International conference on machine learning*, pages 5719–5728. PMLR.

[189] Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., and Arikawa, S. (1999). Byte Pair encoding: A text compression scheme that accelerates pattern matching.

[190] Shriberg, E. and Stolcke, A. (2004). Direct modeling of prosody: An overview of applications in automatic speech processing. In *Speech Prosody 2004, International Conference*.

[191] Shriberg, E. E. (1994). *Preliminaries to a theory of speech disfluencies*. PhD thesis, Citeseer.

[192] Siegelmann, H. T. and Sontag, E. D. (1995). On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150.

[193] Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231.

[194] Sperber, M., Neubig, G., Niehues, J., and Waibel, A. (2019). Attention-passing models for robust and data-efficient end-to-end speech translation. *Transactions of the Association for Computational Linguistics*, 7:313–325.

[195] Sperber, M. and Paulik, M. (2020). Speech translation and the end-to-end promise: Taking stock of where we are. In *ACL*.

[196] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[197] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

[198] Stahlberg, F. and Kumar, S. (2021). Synthetic data generation for grammatical error correction with tagged corruption models. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–47, Online. Association for Computational Linguistics.

[199] Stoian, M. C., Bansal, S., and Goldwater, S. (2020). Analyzing ASR pretraining for low-resource speech-to-text translation. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7909–7913. IEEE.

[200] Taylor, A., Marcus, M., and Santorini, B. (2003). The Penn treebank: An overview. *Treebanks*, pages 5–22.

[201] Thanh-Le Ha, N.-Q. P., Nguyen, T.-N. N. T.-S., Salesky, E., and Waibel, S. S. J. N. A. (2020). Relative positional encoding for speech recognition and direct translation.

[202] Toda, T., Chen, L.-H., Saito, D., Villavicencio, F., Wester, M., Wu, Z., and Yamagishi, J. (2016). The voice conversion challenge 2016. In *Interspeech*, pages 1632–1636.

[203] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[204] Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95.

[205] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.

[206] Waibel, A., Jain, A. N., McNair, A. E., Saito, H., Hauptmann, A. G., and Tebelskis, J. (1991). JANUS: a speech-to-speech translation system using connectionist and symbolic processing strategies. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, pages 793–796. IEEE Computer Society.

[207] Wang, C., Wu, Y., Liu, S., Yang, Z., and Zhou, M. (2020a). Bridging the gap between pre-training and fine-tuning for end-to-end speech translation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9161–9168.

[208] Wang, S., Che, W., Liu, Q., Qin, P., Liu, T., and Wang, W. Y. (2020b). Multi-task self-supervised learning for disfluency detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9193–9200.

[209] Wang, Y., Wong, J. H. M., Gales, M. J. F., Knill, K. M., and Ragni, A. (2018). Sequence teacher-student training of acoustic models for automatic free speaking language assessment. *Proc. IEEE Spoken Language Technology Workshop (SLT)*, pages 994–1000.

[210] Wang, Y.-Y., Deng, L., and Acero, A. (2005). Spoken language understanding. *IEEE Signal Processing Magazine*, 22(5):16–31.

[211] Wang, Y.-Y. and Waibel, A. (1998). Modeling with structures in statistical machine translation. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*, pages 1357–1363.

[212] Watanabe, S., Hori, T., Karita, S., Hayashi, T., Nishitoba, J., Unno, Y., Enrique Yalta Soplin, N., Heymann, J., Wiesner, M., Chen, N., et al. (2018). ESPnet: End-to-End speech processing toolkit. *INTERSPEECH 2018, Hyderabad, India*.

[213] Watanabe, S., Hori, T., Kim, S., Hershey, J. R., and Hayashi, T. (2017). Hybrid ctc/attention architecture for end-to-end speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1240–1253.

[214] Weiss, R. J., Chorowski, J., Jaitly, N., Wu, Y., and Chen, Z. (2017). Sequence-to-sequence models can directly translate foreign speech. *Proc. Interspeech 2017*, pages 2625–2629.

[215] Wen, T.-H., Vandyke, D., Mrkšić, N., Gasic, M., Barahona, L. M. R., Su, P.-H., Ultes, S., and Young, S. (2017). A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 438–449.

[216] Weng, L. (2018). Attention? Attention! *URL: https://lilianweng.github.io/posts/2018-06-24-attention*.

[217] Weng, L. (2019). Generalised Language Models. *URL: https://lilianweng.github.io/posts/2019-01-31-lm*.

[218] Weng, L. (2020). The Transformer Family. *URL: https://lilianweng.github.io/posts/2020-04-07-the-transformer-family/*.

[219] Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[220] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

[221] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (2019). Huggingface's transformers: State-of-the-art natural language processing.

[222] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation.

[223] Xia, Y., Tian, F., Wu, L., Lin, J., Qin, T., Yu, N., and Liu, T.-Y. (2017). Deliberation networks: Sequence generation beyond one-pass decoding. *Advances in neural information processing systems*, 30.

[224] Xie, Z., Avati, A., Arivazhagan, N., Jurafsky, D., and Ng, A. Y. (2016). Neural language correction with character-based attention. *CoRR*.

[225] Xu, H., Chen, T., Gao, D., Wang, Y., Li, K., Goel, N., Carmiel, Y., Povey, D., and Khudanpur, S. (2018). A pruned rnnlm lattice-rescoring algorithm for automatic speech recognition. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5929–5933. IEEE.

[226] Xu, Y., Qiu, X., Zhou, L., and Huang, X. (2020). Improving BERT fine-tuning via self-ensemble and self-distillation. *Journal of Computer Science and Technology*.

[227] Yan, H., Deng, B., Li, X., and Qiu, X. (2019). TENER: adapting transformer encoder for named entity recognition. *arXiv preprint arXiv:1911.04474*.

[228] Yang, J., Yang, D., and Ma, Z. (2020). Planning and generating natural and diverse disfluent texts as augmentation for disfluency detection. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1450–1460.

[229] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.

[230] Yannakoudakis, H., Briscoe, T., and Medlock, B. (2011). A new dataset and method for automatically grading ESOL texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 180–189, Portland, Oregon, USA. Association for Computational Linguistics.

[231] Yee, K., Dauphin, Y., and Auli, M. (2019). Simple and effective noisy channel modeling for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5696–5701.

[232] Yu, L., Blunsom, P., Dyer, C., Grefenstette, E., and Kociský, T. (2017). The neural noisy channel. *In International Conference on Learning Representations, ICLR 2017*.

[233] Yuan, Z. and Briscoe, T. (2016). Grammatical error correction using neural machine translation. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–386.

[234] Zayats, V., Ostendorf, M., and Hajishirzi, H. (2016). Disfluency detection using a bidirectional LSTM.

[235] Zen, H., Tokuda, K., and Black, A. W. (2009). Statistical parametric speech synthesis. *speech communication*, 51(11):1039–1064.

[236] Zens, R., Och, F. J., and Ney, H. (2002). Phrase-based statistical machine translation. In *Annual Conference on Artificial Intelligence*, pages 18–32. Springer.

[237] Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., and Ma, K. (2019a). Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3713–3722.

[238] Zhang, P., Ge, N., Chen, B., and Fan, K. (2019b). Lattice transformer for speech translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6475–6484.

[239] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2019c). BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations*.

[240] Zhang, Y., Liu, Q., and Song, L. (2018). Sentence-state LSTM for text representation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 317–327.

[241] Zwarts, S., Johnson, M., and Dale, R. (2010). Detecting speech repairs incrementally using a noisy channel approach. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1371–1378.

[242] Zweig, G., Yu, C., Droppo, J., and Stolcke, A. (2017). Advances in all-neural speech recognition. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4805–4809. IEEE.