Improving Attention-based Sequence-to-sequence Models

Qingyun Dou

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

King's College October 2021

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.

The following papers are published based on the original contents in this dissertation, and are hence not cited as references.

- Q Dou, X Wu, M Wan, Y Lu, MJF Gales. Deliberation-Based Multi-Pass Speech Synthesis. Annual Conference of the International Speech Communication Association (INTERSPEECH) 2021
- Q Dou, Y Lu, P Manakul, X Wu, MJF Gales. Attention Forcing for Machine Translation. arXiv preprint arXiv:2104.01264.
- Q Dou, J Efiong, MJF Gales. Attention Forcing for Speech Synthesis. Annual Conference of the International Speech Communication Association (INTERSPEECH) 2020
- Q Dou, M Wan, G Degottex, Z Ma, MJF Gales. Hierarchical RNNs for Waveform-level Speech Synthesis. IEEE Spoken Language Technology Workshop (SLT) 2018

Qingyun Dou
October 2021

Abstract

Improving Attention-based Sequence-to-sequence Models
Qingyun Dou

Attention-based models have achieved state-of-the-art performance in various sequence-to-sequence tasks, including Neural Machine Translation (NMT), Automatic Speech Recognition (ASR) and speech synthesis, also known as Text-To-Speech (TTS). These models are often autoregressive, which leads to high modeling capacity, but also makes training difficult. The standard training approach, teacher forcing, suffers from exposure bias: during training the model is guided with the reference output, but the generated output must be used at inference stage. To address this issue, scheduled sampling and professor forcing guide a model with both the reference and the generated output history. To facilitate convergence, they depend on a heuristic schedule or an auxiliary classifier, which can be difficult to tune. Alternatively, sequence-level training approaches guide the model with the generated output history, and optimize a sequence-level criterion. However, many tasks, such as TTS, do not have a well-established sequence-level criterion. In addition, the generation process is often sequential, which is undesirable for parallelizable models such as Transformer.

This thesis introduces attention forcing and deliberation networks to improve attention-based sequence-to-sequence models. Attention forcing guides a model with the generated output history and reference attention. The training criterion is a combination of maximum log-likelihood and the KL-divergence between the reference attention and the generated attention. This approach does not rely on a heuristic schedule or a classifier, and does not require a sequence-level criterion. Variations of attention forcing are proposed for more challenging application scenarios. For tasks such as NMT, the output space is multi-modal in the sense that the given an input, the distribution of the corresponding output can be multi-modal. So a selection scheme is introduced to automatically turn attention forcing on and off depending on the mode of attention.

For parallelizable models, an approximation scheme is proposed to run attention forcing in parallel across time.

Deliberation networks consist of multiple attention-based models. The output is generated in multiple passes, each one conditioned on the initial input and the free running output of the previous pass. This thesis shows that deliberation networks can address exposure bias, which is essential for performance gains. In addition, various training approaches are discussed, and a separate training approach is proposed for its synergy with parallelizable models. Finally, for tasks where the output space is continuous, such as TTS, deliberation networks tend to ignore the free running outputs, thus losing its benefits. To address this issue, a guided attention loss is proposed to regularize the corresponding attention, encouraging the use of the free running outputs.

TTS and NMT are investigated as example sequence-to-sequence tasks, and task-specific techniques are proposed, such as neural vocoder adaption using attention forcing. The experiments demonstrate that attention forcing improves the overall performance and diversity. It is also demonstrated that deliberation networks improve the overall performance, and reduce the chances of attention failure.

Acknowledgements

I would like to thank my supervisor, Mark Gales for his guidance and patience throughout the PhD. I would also like to thank my advisor, Bill Byrne, and college tutor, Laura Davies, for the help at various stages. Many thanks to my family and friends for making these four years smoother. Last but not lease, I would like to thank China Scholarship Council and Cambridge Commonwealth, European and International Trust for financial support.

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

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

$\boldsymbol{X}$	matrix
$\boldsymbol{x}$	vector
X	constant
x	scalar
$\boldsymbol{c}_{1:T}$	context vector sequence (from an attention mechanism)
$\boldsymbol{h}_{1:L}$	encoder output sequence
$\boldsymbol{s}_{1:T}$	decoder state sequence
$\boldsymbol{x}_{1:L}$	input of a sequence-to-sequence model; sequence of L vectors
$\boldsymbol{y}_{1:T}$	output of a sequence-to-sequence model
$\mathcal{D}$	distance metric
$\mathcal{L}$	loss
$\mathcal{Y}$	output space
D	dataset
D	dimension (of a distribution); size of a vector
$\boldsymbol{b}$	bias vector
$\boldsymbol{W}$	weight matrix
$\boldsymbol{k}$	key vector for an attention mechanism

$\boldsymbol{q}$	query vector for an attention mechanism
$\boldsymbol{v}$	value vector for an attention mechanism
$\boldsymbol{c}_{1:J}$	conditioning vector sequence of a neural vocoder
$\boldsymbol{e}_{1:J}$	combined input vector sequence of a neural vocoder
$\boldsymbol{f}_{1:J}$	frame vector sequence of a neural vocoder
$\boldsymbol{h}_{1:J}$	history vector sequence of a neural vocoder
$\boldsymbol{s}_{1:J}$	supervising vector sequence of a neural vocoder
F	frame size of a neural vocoder
R	upsampling ratio of a neural vocoder
$z_{1:J}$	waveform sequence

Greek Symbols

$\boldsymbol{\theta}$	model parameters
ϵ	small and positive constant; hyperparameter of scheduled sampling
$\boldsymbol{\alpha}_{1:T}$	alignment vector sequence from an attention mechanism
$\boldsymbol{\phi}$	model parameters of a down-stream model such as a neural vocoder
$\boldsymbol{\psi}$	model parameters of a discriminator, duration model, or conditioning network
γ	hyperparameter of attention forcing
λ	hyperparameter of scheduled attention forcing
β	hyperparameter of an attention score function; hyperparameter of gradient-based optimization
η	learning rate
μ	mean
σ	standard deviation
τ	optimization step index

$\boldsymbol{\lambda}$	scale vector of discretized mixture of logistics distribution
$\boldsymbol{\mu}$	mean vector of discretized mixture of logistics distribution
$\boldsymbol{\pi}$	mixture weights of discretized mixture of logistics distribution

Superscripts

$\top$	transpose
m	sample index (for Monte Carlo approximation); mixture index
n	data index; occasionally layer index
h	head index for multi-head attention
k	layer index; tier index; iteration index

Subscripts

d	dimension index
i, j	element index of a vector or a matrix
m, n	element index of a convolutional filter or pooling
j	output time step of a down-stream model, such as a neural vocoder
l	input time step
t	output time step

Other Symbols

$*$	convolution
$[;]$	concatenation
$\mathbb{E}$	expectation
$\odot$	element-wise multiplication
$\sigma()$	non-linear function such as sigmoid
$f()$	function; deterministic mapping
$p()$	probability density function or probability mass function

Acronyms / Abbreviations

AF	Attention Forcing
ASR	Automatic Speech Recognition
BAP	Band Aperiodicity
BART	Bidirectional and Auto-Regressive Transformers
BERT	Bidirectional Encoder Representations from Transformers
BLSTM	Bidirectional Long Short Term Memory
BOS	Beginning Of Sequence
BPE	Byte-Pair-Encoding
CDF	Cumulative Distribution Function
CNN	Convolutional Neural Network
DCNN	Dilated Convolutional Neural Network
DNN	Deep Neural Network
DTW	Dynamic Time Warp
EOS	End Of Sequence
F0	Fundamental Frequency
FFNN	Fully-connected Feedforward Neural Network
FR	Free Running
G2P	Grapheme-To-Phoneme
GAN	Generative Adversarial Network
GMM	Gaussian Mixture Model
GNMT	Google's NMT (translation model based on recurrent layers)
GPT	Generative PreTraining
GRU	Gated Recurrent Unit

GV	Global Variance
HMM	Hidden Markov Model
HRNN	Hierarchical Recurrent Neural Network
LSTM	Long Short Term Memory
MBR	Minimum Bayes Risk
MCEP	Mel Cepstrum
MoL	Mixture of Logistics
MOS	Mean Opinion Score
NLL	Negative Log-Likelihood
NMT	Neural Machine Translation
PAF	Parallel Attention Forcing
PML	Pulse Model in Log-domain (a conventional vocoder)
PSS	Parallel Scheduled Sampling
ReLU	Rectified Linear Units
RL	Reinforcement Learning
RMSE	Root-Mean-Square Error
RNN	Recurrent Neural Network
RNN-T	RNN-Transducer
SAF	Scheduled Attention Forcing
SGD	Stochastic Gradient Descent
SMT	Statistical Machine Translation
SS	Scheduled Sampling
STFT	Short-Time Fourier Transform
TF	Teacher Forcing
TTS	Text-To-Speech

Chapter 1

Introduction

Converting one sequence to another is a common goal of a wide range of tasks, including Neural Machine Translation (NMT) [168, 6], Automatic Speech Recognition (ASR) [21, 53] and Text-To-Speech (TTS) [158, 195], also known as speech synthesis. These tasks can be referred to as sequence-to-sequence tasks, and the corresponding models sequence-to-sequence models. A challenge here is that the input and output do not necessarily have the same length. Attention-based models are good at connecting sequences of different length, and have achieved state-of-the-art performance, in various sequence-to-sequence tasks [175, 103, 172]. Here the term performance refers to the overall quality of the output sequences, e.g. word error rate in ASR.

Despite their modeling capacity, attention-based sequence-to-sequence models can be difficult to train [9, 148]. The models are usually implemented as relatively complicated neural networks with tens of thousands of parameters, which makes optimization challenging [168, 51]. From a probabilistic perspective, sequence-to-sequence models estimate the probability of the output sequence conditioned on the input sequence. This probability is often factorized across time, each token conditioned on its previous tokens. To achieve more accurate estimation, sequence-to-sequence models are usually autoregressive [24].

For autoregressive models, a standard approach is teacher forcing, which guides a model with reference output history during training. This makes the model unlikely to recover from its mistakes during inference, where the model operates in free running mode, and the reference output is replaced by the generated output. This problem is referred to as exposure bias [9]. Many approaches have been introduced to tackle exposure bias, and there are mainly two lines of research. Scheduled sampling [9, 43] and professor forcing [97] are prominent examples along the first line. These approaches guide a model with both the reference and the generated output history, and the goal is to learn the data distribution via maximizing the likelihood of the training data. To facilitate convergence, they often depend on a heuristic schedule or an auxiliary classifier, which can be difficult to design and tune [9, 60]. The second line is a series of sequence-level training approaches, leveraging reinforcement learning [148], minimum risk training [159] or generative adversarial training [197]. Theses approaches guide a model with the generated output history. During training, the model operates in free running mode, and the goal is not to generate the reference output, but to optimize a sequence-level loss. However, many tasks do not have well established sequence-level objective metrics. Examples include speech synthesis, voice conversion, machine translation and text summarization [172]. Both lines of research require generating output sequences, and this process is sequential for autoregressive models. In recent years, models based on the Transformer [175] have been widely used, and a key advantage is that when teacher forcing is used, training can be run in parallel across time. To efficiently generate output sequences from Transformer-based models, an approximation scheme [43] has been proposed to parallelize scheduled sampling.

This thesis aims to improve attention-based sequence-to-sequence models, and follows the first line of research described above. The main ideas introduced are attention forcing and deliberation networks. Attention forcing guides the model with the generated output history and reference attention. This approach does not rely on a heuristic schedule or an auxiliary classifier, which makes it simpler to tune than scheduled sampling and professor forcing. Variations of attention forcing are proposed for more challenging application scenarios. For tasks such as machine translation and text summarization, the output space is discrete and multi-modal. To make sure that the training criterion is sensible, a selection scheme can be adopted to automatically turn attention forcing on and off depending on the mode of attention. For parallelizable models such as convolutional models and Transformer-based models, parallel training across time is a major advantage. Here the approximation scheme [43] previously mentioned can be adopted to parallelize attention forcing.

Deliberation networks consist of multiple attention-based models. Here the output is generated in multiple passes, each one conditioned on the initial input and the free running output of the previous pass. Deliberation networks were originally proposed as deeper networks that leverage additional attention mechanisms to access both past and future context when decoding [187]. In this thesis, it is argued and empirically demonstrated that deliberation networks address exposure bias, and that this is essential for performance gains. In addition, various training approaches are discussed, and a separate training approach is proposed for its synergy with parallelizable models. Finally, for tasks where the output space is continuous, such as speech synthesis and voice conversion, it is difficult to train the additional attention over the free running output. Here it is proposed to regularize the attention with a guided attention loss [169], encouraging the attention to be monotonic.

The structure of the thesis is as follows. Chapter 2 reviews the fundamentals of deep learning, covering commonly used building blocks and general training techniques. Chapter 3 describes sequence-to-sequence models that adopt the encoder-attention-decoder architecture. In particular, various training approaches are analyzed, motivating later discussions. Chapter 4 introduces attention forcing, covering the general framework and application considerations. Chapter 5 adopts the same structure to introduce deliberation networks. Experiments are conducted to demonstrate the effectiveness of the novel ideas, taking speech synthesis and machine translation as example tasks. Chapter 6 investigates speech synthesis, describing the general pipeline and analyzing the experimental results. Chapter 7 adopts the same structure to investigate machine translation. Chapter 8 summarizes the thesis and discusses future work.

Chapter 2

Fundamentals of Deep Learning

Deep learning refers to a family of machine learning methods based on deep neural networks, which can approximate complicated mappings. Models based on deep learning have been successfully applied to various sequence-to-sequence tasks, including speech recognition [66], speech synthesis [203] and machine translation [171].

Neural networks approximate mappings by a series of relatively simple linear and nonlinear mappings. Typically there are multiple layers in a network, and each layer realizes a simple mapping. The architecture of a neural network, including the nature of each layer and how they are connected, determines what tasks the neural network is more suitable for. While there is an infinite number of possible architectures, most neural networks can be viewed as a combination of many basic building blocks. For sequence-to-sequence tasks, the most commonly used building blocks are Deep Neural Networks (DNNs) [10, 66], Convolutional Neural Networks (CNNs) [101] and Recurrent Neural Networks (RNNs) [154]. Attention mechanisms [6, 117] and Transformer blocks [175] are more advanced modules, but are becoming standard building blocks in recent years.

2.1 Basic Building Blocks

2.1.1 Deep Neural Networks

A DNN, also known as a Fully connected Feedforward Neural Network (FFNN), is a neural network composed of multiple fully connected feedforward layers [66]. Each

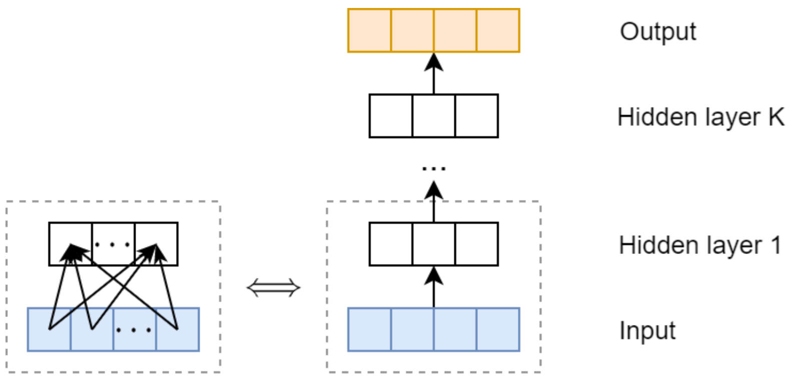

Fig. 2.1 Illustration of a DNN with K hidden layers; the arrows connecting two vectors indicate that each unit in one vector is connected to all units in another.

layer can be viewed as a function, which performs a linear mapping followed by a nonlinear mapping. The output of each hidden layer is the input of the next layer. For each layer, as well as the entire neural network, the output and input are vectors, and they are not necessarily of the same size. The term “fully connected” means that each unit in the output vector depends on all units in the input vector. The term “feedforward” means that information flows from the input, through the intermediate computations, to the output. There are no feedback connections in which the output of a layer is fed back into itself. Such feedback connections are extensively used in RNNs, which will be described in section 2.1.3.

Figure 2.1 illustrates the structure of a DNN with K hidden layers. This can be formulated as:

$$\boldsymbol{h}^{(k)} = f^{(k)}(\boldsymbol{W}^{(k)}\boldsymbol{h}^{(k-1)} + \boldsymbol{b}^{(k)}); 1 < k < K+1 \tag{2.1}$$

$$\boldsymbol{h}^{(1)} = f^{(1)}(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}) \tag{2.2}$$

$$\boldsymbol{y} = f^{(K+1)}(\boldsymbol{W}^{(K+1)}\boldsymbol{h}^{(K)} + \boldsymbol{b}^{(K+1)}) \tag{2.3}$$

the superscript $^{(k)}$, where $1 \leq k \leq K+1$, denotes the layer index; $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{h}$ denote input, output and hidden vectors; $\boldsymbol{W}$ and $\boldsymbol{b}$ denote weight matrix and bias vector of a linear mapping; f denotes a non-linear function, also referred to as an activation function. Typically an activation function operates independently on each element, i.e. unit, of its input vector. Section 2.1.4 will describe a range of standard activation functions.

In general, feedforward networks with one or more hidden layers are able to approximate any function [51, 68, 69].[1] However, there are several practical issues: 1) the hidden layers may need to be infeasibly large; 2) the training algorithm may fail to find the correct parameters; 3) there may not be enough training data. In many circumstances, using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error [51].

2.1.2 Convolutional Neural Networks

For DNNs, the input is a vector of a fixed size, which results in two limitations. First, the network is not flexible for inputs of a varying size. Second, flattening the input into a vector ignores some structures in the data. However, many natural signals have compositional structures, in which higher-level features are obtained by composing lower-level ones [100]. For example, in images, local combinations of edges form motifs, motifs assemble into parts, and parts form objects. Similarly, in text, characters form words, words assemble into phrases, and phrases form sentences. These limitations motivate the use of CNNs [101], which can take as input a multi-dimensional matrix of a varying size.

A CNN is a neural network that use convolution in place of general matrix multiplication in at least one of its layers [51]. Most standard CNNs have multiple convolutional layers and pooling layers, followed by some fully connected layers [100]. The intuition for this architecture is twofold. First, in array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. Second, the local statistics of images and other signals are invariant to location. In other words, if a motif can appear in one part of the image, it could appear anywhere, hence the idea of units at different locations sharing the same weights and detecting the same pattern in different parts of the array.

[1]The universal approximation theorem [68] states that a feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function, which saturates for very negative or very positive arguments, can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units. The derivatives of the feedforward network can also approximate the derivatives of the function arbitrarily well [69]. The concept of Borel measurability is beyond the scope of this thesis. Here the key is that any continuous function on a closed and bounded subset of $\mathbb{R}^D$, where $\mathbb{R}^D$ denotes the finite(D)-dimensional space of real numbers, is Borel measurable. The universal approximation theorem has also been extended to finite-dimensional discrete spaces, and to a wider class of activation functions [51], including the now commonly used rectified linear unit [122], which will be described in section 2.1.4.

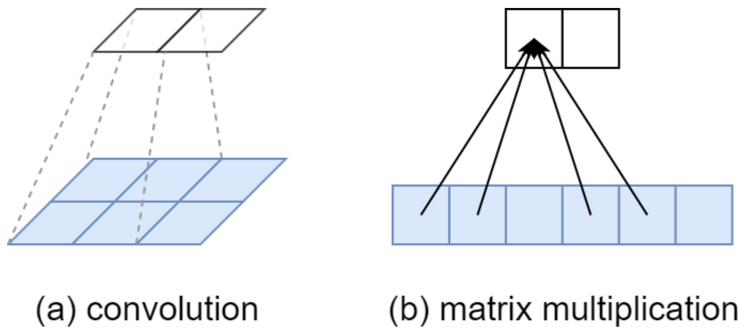

Fig. 2.2 Illustration of a 2D convolution and a corresponding matrix multiplication; the parallelograms are matrices, and the rectangles are vectors; the computation for the first output unit is shown, and the rest are analogous.

Convolution

In the context of neural networks, convolution is a special linear operation. It connects the output and input in such a way that it scales more efficiently w.r.t. the size of the input. A convolutional layer realizes such an operation. It is feedforward, but is not fully connected. In addition, many parameters, i.e. weights, in the layer are shared. Figure 2.2 illustrates a 2D convolution and a corresponding matrix multiplication. The left side is a convolution mapping a 2×3 matrix $\boldsymbol{X}$ to a 1×2 matrix $\boldsymbol{Y}$, using a sliding filter represented as a 2×2 matrix $\boldsymbol{W}$. The convolution can be formulated as

$$
\begin{aligned}
\boldsymbol{X} &= \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \end{bmatrix} \\
\boldsymbol{Y} &= \begin{bmatrix} y_{1,1} & y_{1,2} \end{bmatrix} \\
\boldsymbol{W} &= \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{bmatrix} \\
y_{i,j} &= \sum_{m=1}^{2} \sum_{n=1}^{2} w_{m,n} x_{i+m-1,j+n-1}; i = 1; j = 1, 2
\end{aligned}
$$

where i and j denote the indices of the input and output, and m and n denote the indices of the filter. The right side of figure 2.2 is a matrix multiplication that yields the same result. The input and output are flattened as vectors $\boldsymbol{x}$ and $\boldsymbol{y}$; the filter is used to construct a sparse weight matrix with tied weights $\widetilde{\boldsymbol{W}}$. This can be formulated as

$$\begin{aligned}\boldsymbol{x} &= \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{2,1} & x_{2,2} & x_{2,3} \end{bmatrix}^\top \\ \boldsymbol{y} &= \begin{bmatrix} y_{1,1} & y_{1,2} \end{bmatrix}^\top = \widetilde{\boldsymbol{W}}\boldsymbol{x} \\ \widetilde{\boldsymbol{W}} &= \begin{bmatrix} w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} & 0 \\ 0 & w_{1,1} & w_{1,2} & 0 & w_{2,1} & w_{2,2} \end{bmatrix}\end{aligned}$$

In general, convolutions are agnostic to the size of the input. They can be applied at any dimensionality, but 1D [174, 126] and 2D [92, 162, 64] are the most standard forms. The first successful applications of CNNs were in the field of computer vision, where the 2D convolutions are usually used [51]. For a 2D convolutional layer, the most general input is a 3D matrix, where the third dimension is for the channels; a group of 3D filters are used, each producing a 2D matrix, also called a feature map; the output is the stack of these feature maps, also a 3D matrix.

Generally speaking, the role of the convolutional layer is to detect local conjunctions of features from the previous layer. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter. The result of this local weighted sum is then passed through an activation function. All units in a feature map share the same filter, while different feature maps in a layer use different filters. Three hyperparameters control the size of the output feature maps: the size, stride and zero-padding. Stride determines the distance the filter slides each time; zero-padding determines whether and how to pad the input with zeros around the border. Dilation [196] has been introduced more recently. It inserts masked spaces into the receptive field of a filter, making it more efficient for modeling longer-range dependencies [174].

Pooling

The role of the pooling layer is to merge semantically similar features into one. Because the relative positions of the features forming a motif can vary somewhat, reliably detecting the motif can be done by coarse-graining the position of each feature. A typical pooling unit computes the maximum of a local patch of units in one feature map or in a few feature maps. Neighboring pooling units take input from patches that are shifted by more than one row or column, thereby reducing the dimension of the representation and creating an invariance to small shifts and distortions. The pooling

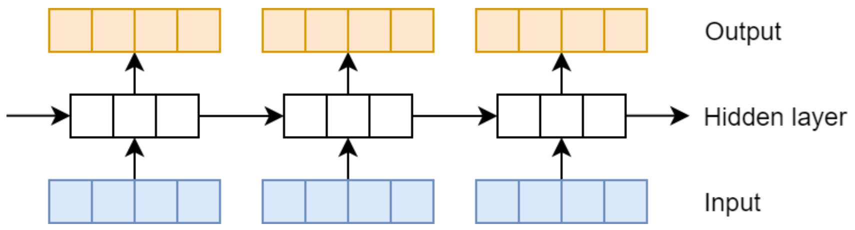

Fig. 2.3 Illustration of a unidirectional RNN with one hidden layer.

layer allows representations to change very little when elements in the previous layer change in position and appearance [100].

Standard pooling layers operate on each feature map independently of other maps, yielding a stack as deep as the input. For pooling layers, it is not common to pad the input using zero-padding. Hence the size of the output feature maps are usually controlled by the pooling window size and stride. A range of pooling functions have been proposed, such as maxout, soft-maxout and L_p-norm. These functions are pre-determined, so the pooling layers do not need to learn any parameters. Without loss of generality, for a pooling window of size $M \times N$, maxout, soft-maxout and p-norm pooling can be written respectively as equations 2.4 to 2.6.

$$z_{i,j} = \max_{m,n}(y_{i,j}, ..., y_{i+M-1,j+N-1}) \tag{2.4}$$

$$z_{i,j} = \log(\sum_{m=1}^{M}\sum_{n=1}^{N}\exp(y_{i+m-1,j+n-1})) \tag{2.5}$$

$$z_{i,j} = (\sum_{m=1}^{M}\sum_{n=1}^{N}|y_{i+m-1,j+n-1}|^{p})^{1/p} \tag{2.6}$$

The computation is shown for position i, j of the output feature map $\boldsymbol{Z}$. The input feature map is $\boldsymbol{Y}$, but the pooling window only covers a part of it: $\boldsymbol{Y}_{i:i+M-1,j:j+N-1}$. To simplify the notation, it is assumed that the stride of the pooling layer is one in both dimensions.

2.1.3 Recurrent Neural Networks

A RNN can be viewed as a time series of DNNs that share wights. RNNs process an input sequence one element at a time, maintaining in their hidden units a history vector, also referred to as a state vector. This vector implicitly contains information about the history of all the past elements of the sequence. Hence for tasks that involve sequential inputs, such as speech and text, it is often better to use RNNs [100]. Figure

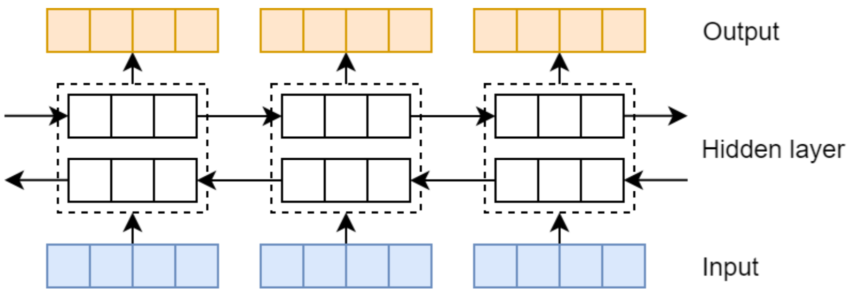

Fig. 2.4 Illustration of a bidirectional RNN with one hidden layer.

2.3 illustrates the structure of a unidirectional RNN. At each time step, the current history vector is updated as a function of the current input vector and the previous history vector, and the current output is a function of the current history vector. A RNN can be formulated as

$$\boldsymbol{h}_t = f_h(\boldsymbol{W}_h^f \boldsymbol{x}_t + \boldsymbol{W}_h^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{2.7}$$

$$\boldsymbol{y}_t = f_y(\boldsymbol{W}_y \boldsymbol{h}_t + \boldsymbol{b}_y) \tag{2.8}$$

$\boldsymbol{x}_{1:T}$, $\boldsymbol{y}_{1:T}$ and $\boldsymbol{h}_{1:T}$ denote the input, output and history vectors; T is the length of the sequences; $\boldsymbol{W}$ and $\boldsymbol{b}$ denote weight matrix and bias vector of a linear mapping; f denotes an activation function.

Unidirectional RNNs are good at capturing information from the past time steps, but not the future. To capture information from the entire input sequence, a pair of unidirectional RNNs running in reverse directions can be combined to form a bidirectional RNN. Figure 2.4 illustrates the structure of a bidirectional RNN. At each time step t, the history vector $\boldsymbol{h}_t$ combines the forward and backward history vectors $\overrightarrow{\boldsymbol{h}}_t$ and $\overleftarrow{\boldsymbol{h}}_t$, usually by concatenation. This can be formulated as

$$\overrightarrow{\boldsymbol{h}}_t = \overrightarrow{f}_h(\overrightarrow{\boldsymbol{W}}_h^f \boldsymbol{x}_t + \overrightarrow{\boldsymbol{W}}_h^r \overrightarrow{\boldsymbol{h}}_{t-1} + \overrightarrow{\boldsymbol{b}}_h) \tag{2.9}$$

$$\overleftarrow{\boldsymbol{h}}_t = \overleftarrow{f}_h(\overleftarrow{\boldsymbol{W}}_h^f \boldsymbol{x}_t + \overleftarrow{\boldsymbol{W}}_h^r \overleftarrow{\boldsymbol{h}}_{t+1} + \overleftarrow{\boldsymbol{b}}_h) \tag{2.10}$$

$$\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}}_t; \overleftarrow{\boldsymbol{h}}_t] \tag{2.11}$$

where $[;]$ denotes the concatenation of two column vectors.

When unfolded in time, RNNs can be seen as very deep feedforward neural networks. Although their main purpose is to learn long-term dependencies, training them has proved to be problematic because the gradients either grow or shrink at each time step, so over many time steps they often explode or vanish. Theoretical and empirical evidence shows that it is difficult for vanilla RNNs to learn to store information for very long [131]. To solve this problem, many variations have been proposed. Two popular proposals are the Gated Recurrent Unit (GRU) [30] and Long Short-Term Memory (LSTM). They still keep the structure in time; meanwhile, the simple activation function in equation 2.7 is extended to enhance memory. Detailed discussions on the GRU and LSTM can be found in appendix A.

2.1.4 Activation Functions

Commonly used activation functions include softmax function, sigmoid function, hyperbolic tan (tanh) function, Rectified Linear Units (ReLU) and some of its variations. Softmax function can be wirtten as:

$$f(x_i) = \frac{\exp(x_i)}{\sum_{i'=1}^{I} \exp(x_{i'})};\ 0 \leq f(x_i) \leq 1; \sum_{i=1}^{I} f(x_i) = 1 \tag{2.12}$$

Each element of the output vector is between 0 and 1, and all elements sum to 1. Hence the softmax function is often used in the output layer of a neural network designed for classification tasks.

Sigmoid function and tanh function are respectively shown in equations 2.13 and 2.14.

$$f(x_i) = \frac{1}{1 + \exp(-x_i)};\ 0 \leq f(x_i) \leq 1 \tag{2.13}$$

$$f(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)};\ -1 \leq f(x_i) \leq 1 \tag{2.14}$$

Both of them can keep a layer from outputting extremely large or small numbers, and are often used in hidden layers. A problem of sigmoid and tanh is that they are relatively easy to saturate, which results in slow convergence when using gradient-based optimization methods.

ReLU can be used to solve this problem. Equation 2.15 shows the expression of ReLU.

$$f(x_i) = \max(0, x_i);\ 0 \leq f(x_i) \tag{2.15}$$

It is very efficient as there is no exponential function or division. Figure 2.5 compares ReLU with sigmoid and tanh; it can be seen that ReLU does not saturate for positive

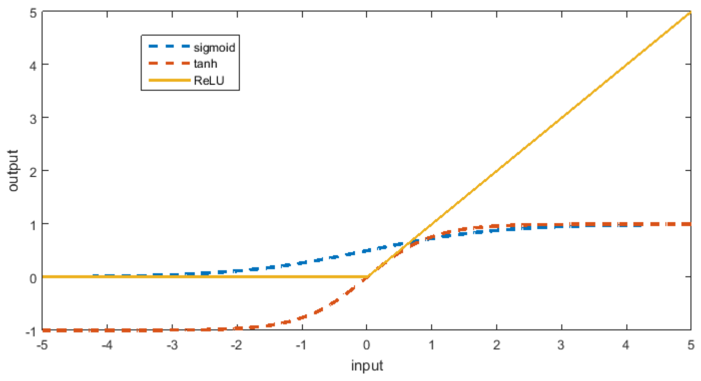

Fig. 2.5 Activation functions.

inputs. Experiments in previous research have shown that ReLU leads to rapid convergence in training [122].

2.2 Attention Mechanisms

2.2.1 Framework

Conceptually, an attention mechanism connects two sequences.[2] It generates a time-dependent context vector $\boldsymbol{c}_t$, which summarizes an input sequence $\boldsymbol{x}_{1:L}$ according to an output sequence $\boldsymbol{y}_{1:T}$. In other words, it maps $\boldsymbol{x}_{1:L}$ to a context vector sequence $\boldsymbol{c}_{1:T}$, which is aligned with $\boldsymbol{y}_{1:T}$. For each output token $\boldsymbol{y}_t$, the attention mechanism produces a context vector $\boldsymbol{c}_t$, by performing a weighted sum of the input sequence. The input and output sequences are typically embedded by hidden sequences, where each token has some information about the sequence. Let $\boldsymbol{h}_{1:L}$ embed the input sequence, and $\boldsymbol{s}_{1:T}$ the output sequence. At each time step t, the weights are packed into an attention vector $\boldsymbol{\alpha}_t = [\alpha_{t,1}, ..., \alpha_{t,L}]^\top$, and are computed based on one output embedding $\boldsymbol{s}_t$, and the sequence of input embeddings $\boldsymbol{h}_{1:L}$. This process can be formulated as follows.

$$\boldsymbol{c}_t = \sum_{l=1}^{L} \alpha_{t,l} \boldsymbol{h}_l \tag{2.16}$$

$$\alpha_{t,l} = \frac{\exp(f(\boldsymbol{s}_t, \boldsymbol{h}_l; \boldsymbol{\theta}_\alpha))}{\sum_{l'=1}^{L} \exp(f(\boldsymbol{s}_t, \boldsymbol{h}_{l'}; \boldsymbol{\theta}_\alpha))} \tag{2.17}$$

[2]More generally, an attention mechanism can be applied to non-sequential data, which is beyond the scope of this thesis.

where $\boldsymbol{\theta}_\alpha$ denotes the attention mechanism, and $f(\boldsymbol{s}_t, \boldsymbol{h}_l; \boldsymbol{\theta}_\alpha)$ denotes a score function computing unnormalized weights. The weights, i.e. the entries of an attention vector, indicate the focus on the input tokens. The sequence of attention vectors $\boldsymbol{\alpha}_{1:T}$ form an attention map, which is a $T \times L$ matrix.

The score function $f(\boldsymbol{s}_t, \boldsymbol{h}_l; \boldsymbol{\theta}_\alpha)$ is central to the attention mechanism, and can take various forms. The initial work proposing attention mechanism [6] adopted the additive score function:

$$f(\boldsymbol{s}_t, \boldsymbol{h}_l; \boldsymbol{\theta}_\alpha) = \boldsymbol{v}_\alpha^\top \tanh(\boldsymbol{W}_\alpha[\boldsymbol{s}_t; \boldsymbol{h}_l]) \tag{2.18}$$

where $\boldsymbol{v}_\alpha$ and $\boldsymbol{W}_\alpha$ are weight matrices to be learned, and $[;]$ denotes concatenation. The attention mechanism is parametrized by a feedforward network with a single hidden layer, using tanh as the activation function. The network is jointly trained as part of a sequence-to-sequence model.

In reference [6], the embedding sequences $\boldsymbol{h}_{1:L}$ and $\boldsymbol{s}_{1:T}$ are respectively produced by an encoder $\boldsymbol{\theta}_h$ and a decoder $\boldsymbol{\theta}_s$. Figure 2.6 illustrates this attention-based sequence-to-sequence model. The encoder is a bidirectional RNN, which has a forward state and a backward state for each encoder time step. The encoder output $\boldsymbol{h}_{1:L}$ is the concatenation of the two states. The decoder's time steps are aligned with the output $\boldsymbol{y}_{1:T}$. At each step, its state is updated as $\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_{t-1}; \boldsymbol{\theta}_s)$.[3] In sequence-to-sequence generation, it is common that the encoder is bidirectional, while the decoder is unidirectional. The reason is that at inference stage, the model does not have access to future tokens. There are alternative ways to build the encoder and the decoder, which will be described in section 3.1.

Following the success of the additive attention, various forms of attention have been investigated [17, 117, 175]. Table 2.1 lists several popular attention mechanisms and their corresponding score functions. β denotes a hyper-parameter, and D_h the dimension of the encoder output.

[3] In reference [6], the state update equation is $\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t; \boldsymbol{\theta}_s)$; the context vector $\boldsymbol{c}_t$ is computed with the state at the previous time step $\boldsymbol{s}_{t-1}$. This indexing is equivalent to the one adopted in this thesis, but is less readable as the subscript $_{t-1}$, instead of $_t$, will show up in the score functions. It is easier to see the equivalence, when the computation of $\boldsymbol{c}_t$ and $\boldsymbol{s}_t$ is expanded, starting from the beginning of the sequence, where $t = 0$.

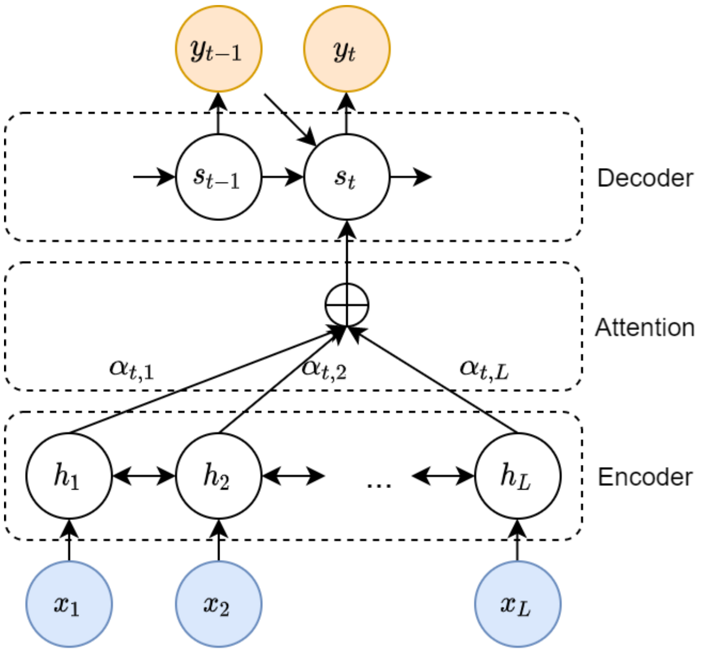

Fig. 2.6 Illustration of an encoder-decoder model with attention [6], at time step t; each circle depicts a vector.

Table 2.1 Popular attention mechanisms and their corresponding score functions.

Name	$f(\boldsymbol{s}_t, \boldsymbol{h}_l; \boldsymbol{\theta}_\alpha)$	Reference
Cosine	$\beta(\boldsymbol{s}_t^\top \boldsymbol{h}_l)/(\Vert\boldsymbol{s}_t\Vert\Vert h_l\Vert)$	[54]
Additive	$\boldsymbol{v}_\alpha^\top \tanh(\boldsymbol{W}_\alpha[\boldsymbol{s}_t; \boldsymbol{h}_l])$	[6]
General	$\boldsymbol{s}_t^\top \boldsymbol{W}_\alpha \boldsymbol{h}_l$	[117]
Dot-product	$\boldsymbol{s}_t^\top \boldsymbol{h}_l$	[117]
Scaled dot-product	$\boldsymbol{s}_t^\top \boldsymbol{h}_l/\sqrt{D_h}$	[175]

Incorporating Prior Knowledge

In theory, the most powerful attention mechanisms are the ones that make few assumptions about the alignment between the input and output, such as the general attention [117]. Incorporating prior knowledge induces some bias, but can make the model considerably more efficient, given that the corresponding assumptions are sensible. Here the term efficient means that the model requires relatively fewer parameters to achieve a certain level performance.

For example, by limiting the attention to parts of the input sequence [189, 117], both training and inference are more efficient. This can be crucial when the input is large. Another example is the location-sensitive attention [28], which extends the additive attention mechanism [6] by considering attention weights from previous decoder time steps. The score function is:

$$f(\boldsymbol{s}_t, \boldsymbol{h}_l; \boldsymbol{\theta}_\alpha) = \boldsymbol{v}_\alpha^\top \tanh(\boldsymbol{W}_\alpha[\boldsymbol{s}_t; \boldsymbol{h}_l; \boldsymbol{\beta}_l]); \quad \boldsymbol{\beta}_{1:L} = \boldsymbol{W}_\beta * \boldsymbol{\alpha}_{t-1} \tag{2.19}$$

where $\boldsymbol{W}_\beta$ denotes a stack of 1D filters and $*$ denotes convolution; $\boldsymbol{\beta}_{1:L}$ denotes the result of the convolution. This encourages the model to move forward consistently through the input, mitigating potential failure modes where some subsequences are repeated or ignored by the decoder. For tasks where the alignment should be monotonic, such as ASR and TTS, it is very beneficial to use location-sensitive attention [28, 158]. In contrast, for tasks where the alignment is not necessarily monotonic, such as NMT, it is more sensible to use a more general type of attention.

2.2.2 Self-attention

So far the attention mechanisms have been used to connect two sequences. In this case, a cross-sequence attention mechanism produces a time-dependent summary of the input sequence, for each time step of the output sequence. Self-attention [25, 130, 134] is an attention mechanism relating different tokens of a single sequence in order to compute a representation of the same sequence. It can be interpreted as special usage of attention, where the two sequences to be connected are the same. Here the context sequence $\boldsymbol{c}_{1:L}$ has the same length as the sequence of input embeddings $\boldsymbol{h}_{1:L}$, and can be viewed as a better representation of the input. Equations 2.16 and 2.17 become

$$\boldsymbol{c}_l = \sum_{l'=1}^{L} \alpha_{l,l'} \boldsymbol{h}_{l'} \tag{2.20}$$

$$\alpha_{l,l'} = \frac{\exp(f(\boldsymbol{h}_l, \boldsymbol{h}_{l'}; \boldsymbol{\theta}_\alpha))}{\sum_{l''=1}^{L} \exp(f(\boldsymbol{h}_l, \boldsymbol{h}_{l''}; \boldsymbol{\theta}_\alpha))} \tag{2.21}$$

Similar to the case of connecting two different sequences, the score function can take various forms. The scaled dot-product attention [175] is a common choice, which allows parallel computation across time. As a representation learning model, self-attention does not make assumptions about its input sequence. In contrast, both CNNs and RNNs implicitly assume that the embedding of one token depends more on its neighboring tokens. Such assumptions can make the model more efficient, but only when they are sensible [51].

In some recent litterateur [175], where both self-attention and cross-sequence attention are used, an alternative description of attention is often adopted. Here an attention mechanism is described as mapping a query and a sequence of key-value pairs to a context vector. The context vector is computed as a weighted sum of the values, where the weight assigned to each value is computed by a score function of the query and the corresponding key. The process involves three sequences: a query sequence $\boldsymbol{q}_{1:T}$, a key sequence $\boldsymbol{k}_{1:L}$ and a value sequence $\boldsymbol{v}_{1:L}$. This can be formulated as

$$\boldsymbol{c}_{1:T} = f(\boldsymbol{q}_{1:T}, \boldsymbol{k}_{1:L}, \boldsymbol{v}_{1:L}; \boldsymbol{\theta}_\alpha) \tag{2.22}$$

at the sequence-level, and

$$\boldsymbol{c}_t = \sum_{l=1}^{L} \alpha_{t,l} \boldsymbol{v}_l \tag{2.23}$$

$$\alpha_{t,l} = \frac{\exp(f(\boldsymbol{q}_t, \boldsymbol{k}_l; \boldsymbol{\theta}_\alpha))}{\sum_{l'=1}^{L} \exp(f(\boldsymbol{q}_t, \boldsymbol{k}_{l'}; \boldsymbol{\theta}_\alpha))} \tag{2.24}$$

at time step t of the query sequence. This formulation is equivalent to cross-sequence attention, shown in equations 2.16 and 2.17, when

$$\boldsymbol{q}_{1:T} = \boldsymbol{s}_{1:T} \tag{2.25}$$

$$\boldsymbol{k}_{1:L} = \boldsymbol{v}_{1:L} = \boldsymbol{h}_{1:L} \tag{2.26}$$

It is equivalent to self-attention, when all three sequences are the same:

$$\boldsymbol{q}_{1:T} = \boldsymbol{k}_{1:L} = \boldsymbol{v}_{1:L} = \boldsymbol{h}_{1:L}; \quad T = L \tag{2.27}$$

In this thesis, the key and value sequences are the same, except when Transformer blocks are used. Therefore the query-key-value description is not the default description. When the query-key-value description is adopted, it will be assumed that the query sequence has a different length to the key and value sequences.

2.2.3 Multi-head Attention

Multi-head attention [175] is the combination of a group of attention mechanisms. The heads, i.e. members of the group, have the same structure, but different parameters. The computation of multiple attention mechanisms can be done in parallel. Figure 2.7 illustrates the structure of multi-head attention. For each head, the query, key and value sequences are first linearly projected to a different space. Let H denote the number of heads, h the head index, and $\boldsymbol{c}^{(h)}_{1:T}$ the sequence of context vectors produced by head h, $\boldsymbol{W}^{(h)}_q$, $\boldsymbol{W}^{(h)}_k$ and $\boldsymbol{W}^{(h)}_v$ the linear projection matrices. This can be formulated as

$$\boldsymbol{c}^{(h)}_{1:T} = f(\boldsymbol{W}^{(h)}_q \boldsymbol{q}_{1:T}, \boldsymbol{W}^{(h)}_k \boldsymbol{k}_{1:L}, \boldsymbol{W}^{(h)}_v \boldsymbol{v}_{1:L}; \boldsymbol{\theta}_\alpha) \tag{2.28}$$

At each time step of the query sequence, each head produces a context vector; these vectors are then concatenated and linearly projected to have a reduced dimension. Let $\boldsymbol{W}_c$ denote the projection matrix. At time step t, multi-head attention can be formulated as

$$\boldsymbol{c}_t = \boldsymbol{W}_c[\boldsymbol{c}^{(1)}_t; \boldsymbol{c}^{(2)}_t; ...\boldsymbol{c}^{(H)}_t] \tag{2.29}$$

Intuitively, multi-head attention expands the model's ability to focus on different positions. From a different perspective, it gives the attention layer multiple representation spaces [175].

For multi-head attention, when the score function does not have any parameters, e.g. scaled dot-product attention [175] and cosine attention [54], it is essential to map the input sequences to different spaces for each head [175]. Without these linear mappings, the attention heads will be identical, losing the benefits of using multiple heads.

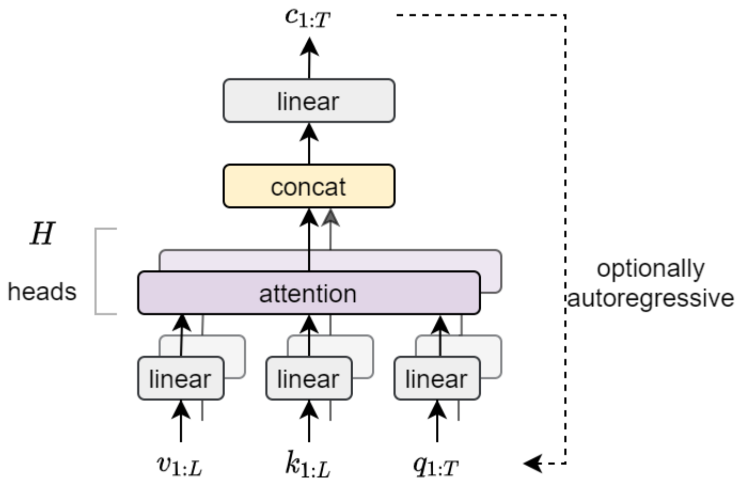

Fig. 2.7 Illustration of multi-head attention [175].

2.3 Transformer Blocks

A Transformer block [175], also referred to as a Transformer layer, is a combination of many basic modules. There are two types of Transformer blocks: Transformer encoder blocks, which encode a sequence, and Transformer decoder blocks, which connect two sequences. Figure 2.8 illustrates both types of Transformer blocks, as well as how they are combined to form an encoder-decoder model, which will be described in chapter 3.

A Transformer encoder block has two main modules: a multi-head self-attention followed by a feedforward network. Each sub-layer is followed by a residual connection and then layer normalization. The multi-head self-attention adopts scaled dot-product attention, described in Table 2.1. The feedforward network consists of two linear transformations with a ReLU activation in between. It is applied to each position separately and identically.

A Transformer decoder block has an extra main module, a multi-head cross-sequence attention, which is inserted between the multi-head self-attention and the feedforward network described above. For the cross-sequence attention, the query sequence is the output of the self-attention, and the key and value sequences are the usually the output of a Transformer encoder block. In the decoder block, the self-attention is often masked in such a way that prevents positions from attending to subsequent positions.

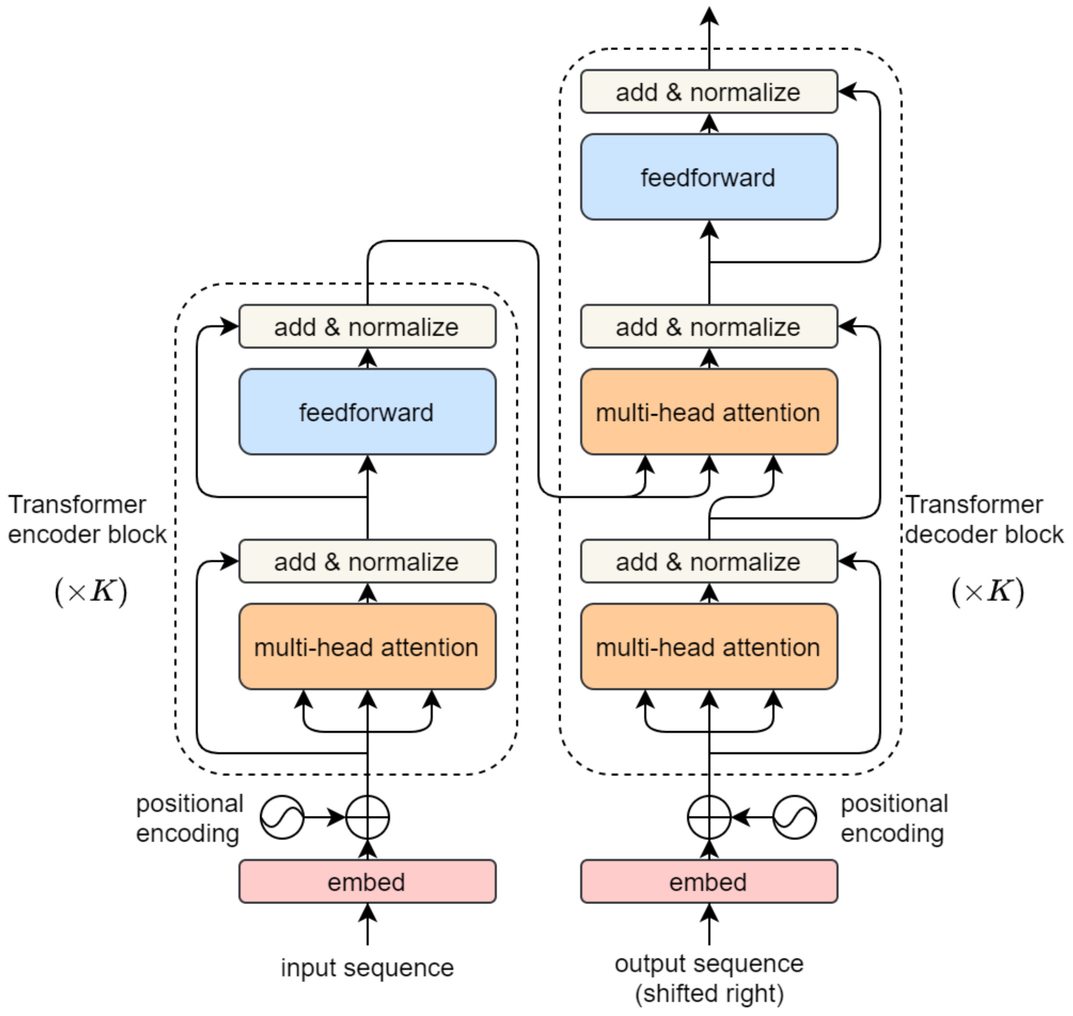

Fig. 2.8 Illustration of Transformer blocks [175]; the dashed rectangle in the left is a Transformer encoder block; the dashed rectangle in the right is a Transformer decoder block.

Residual Connection and Layer Normalization

For the main modules in Transformer blocks, residual connections [64] and layer normalization [4] are applied. The residual connections add the input of a module to its output, which creates an additional path to facilitate training [64]. Layer normalization adjusts the mean and standard deviation of the output of a layer, which can reduce the training time [4]. The original output vector is first normalized, over its elements, so that the mean is zero and the standard deviation is one; then the normalized vector is scaled and shifted element-wise. The scaling and shifting vectors are learnable parameters. Layer normalization can be formulated as

$$\bar{\boldsymbol{y}} = \boldsymbol{g} \odot \frac{\boldsymbol{y} - \mu}{\sigma} + \boldsymbol{b}; \quad \mu = \frac{1}{D}\sum_{d=1}^{D} y_d; \quad \sigma = (\frac{1}{D}\sum_{d=1}^{D}(y_d - \mu)^2)^{\frac{1}{2}} \tag{2.30}$$

where $\boldsymbol{y}$ is the original output vector of size D; d is the element index; μ and σ are its mean and standard deviation; $\boldsymbol{g}$ and $\boldsymbol{b}$ are the scaling and shifting vectors; $\odot$ denotes element-wise multiplication.

Positional Encoding

Unlike CNNs and RNNs, Transformer blocks are not sensitive to the order of the sequence. Hence positional encodings are added to the input and output embeddings. The positional encodings have the information about the relative or absolute position of the tokens in the sequence, and have the same size as the original embeddings. There are many choices of positional encodings, learned and fixed [49]. The standard Transformer blocks use sine and cosine functions of different frequencies [175]:

$$e_{l,d} = \begin{cases} \sin(l/10000^{d/D}) & \text{if } d \text{ is even} \\ \cos(l/10000^{d/D}) & \text{if } d \text{ is odd} \end{cases} \tag{2.31}$$

where $e_{l,d}$ is its d-th element of $\boldsymbol{e}_l$, the encoding at position l; D denotes its size. The notation D is slightly abused here to improve readability. The sinusoidal encodings allow the model to extrapolate to sequence lengths longer than the ones encountered during training [175].

2.4 Training

In general, a neural network is trained through optimizing a loss is as a function of parameters:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{\theta}) \tag{2.32}$$

$\boldsymbol{\theta}$ denotes parameters, and $\boldsymbol{\theta}^*$ the optimal model; $\mathcal{L}(\boldsymbol{\theta})$ denotes loss. This section takes supervised learning, the most common form of machine learning [100], as an example to elaborate on the basics of neural network training.

2.4.1 Training Criteria

The loss functions, i.e. the training criteria, are usually selected based on the nature of tasks. This section describes some simple loss functions for non-sequential tasks. Without loss of generality, it is assumed that the input and output are both vectors. More complicated training criteria will be described in section 3.3, in the context of sequence-to-sequence models.

For classification tasks, cross-entropy is a commonly used loss. Let $\{\boldsymbol{y}^{(n)}, \boldsymbol{x}^{(n)}\}_1^N$ denote N training examples; $\boldsymbol{x}^{(n)}$ denotes an input; $\boldsymbol{y}^{(n)}$ denotes the true one-hot class label, a D-dimensional discrete vector where all the elements are zero except for the one associated with the true class; $\hat{\boldsymbol{y}}^{(n)}$ denotes the corresponding output of a neural network.[4] Cross-entropy loss can be written as:

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{d=1}^{D} y_d^{(n)} \log(\hat{y}_d^{(n)}); \quad \hat{\boldsymbol{y}}^{(n)} = f(\boldsymbol{x}^{(n)}; \boldsymbol{\theta}) \tag{2.33}$$

For regression tasks, L_p loss is a common training criterion. Taking L_2 (least square error) as an example, this can be formulated as follows, where $\boldsymbol{y}^{(n)}$ and $\hat{\boldsymbol{y}}^{(n)}$ are continuous vectors.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} ||\hat{\boldsymbol{y}}^{(n)} - \boldsymbol{y}^{(n)}||_2^2; \quad \hat{\boldsymbol{y}}^{(n)} = f(\boldsymbol{x}^{(n)}; \boldsymbol{\theta}) \tag{2.34}$$

[4] In this section, it is assumed that both the input and the output are represented by a single vector. The discussion can be extended to more complex structures such as a sequence of vectors. Section 3.3 will give a more detailed description of training approaches, in the context of sequence-to-sequence modeling.

2.4.2 Optimization

For the optimization problem in equation 2.32, there is usually no closed-form solution, and gradient descent [153] is used to solve the problem numerically. This can be formulated as

$$\boldsymbol{\theta}[\tau+1] = \boldsymbol{\theta}[\tau] - \Delta\boldsymbol{\theta}[\tau] = \boldsymbol{\theta}[\tau] - \eta\frac{\partial\mathcal{L}}{\partial\boldsymbol{\theta}}|_{\boldsymbol{\theta}[\tau]} \tag{2.35}$$

where τ denotes an iteration, i.e. a time step in the optimization process, and η denotes the step size, i.e. the learning rate. The loss function can be seen as a kind of hilly landscape in the high-dimensional space of weight values. The negative gradient vector indicates the direction of steepest descent in this landscape, taking it closer to a minimum, where the output error is low on average [100]. To obtain a good solution, several questions should be considered: how to get the gradient for all model parameters, how to avoid local minima degrading performance, and how to set the step size. Here the term performance refers to the overall quality of the output sequences, e.g. word error rate in ASR.

For each training example, the gradient is computed using backpropagation, which is a practical application of the chain rule for derivatives. The key insight is that the gradient of the loss with respect to the input of a module can be computed by working backwards from the gradient with respect to the output of that module, or the input of the subsequent module. The backpropagation procedure can be applied repeatedly to propagate gradients through all modules, starting from the output all the way to back to where the input is fed in. Once these gradients have been computed, it is straightforward to compute the gradients with respect to the weights of each module [100]. Backpropagation can be formulated as the following equations, where $\boldsymbol{\theta} = \{\boldsymbol{W}^{(1)}, ..., \boldsymbol{W}^{(K+1)}\}$; $\boldsymbol{z}^{(k)} = \boldsymbol{W}^{(k)}\boldsymbol{x}^{(k)}$ is the input vector to the activation function; $\boldsymbol{y}^{(k)} = f(\boldsymbol{z}^{(k)}) = \boldsymbol{x}^{(k+1)}$ is the output vector.

$$\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{W}^{(k)}} = \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{z}^{(k)}}\frac{\partial\boldsymbol{z}^{(k)}}{\partial\boldsymbol{W}^{(k)}} = \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{z}^{(k)}}\boldsymbol{x}^{(k)\top} \tag{2.36}$$

$$\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{z}^{(k)}} = \frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{z}^{(k+1)}}\frac{\partial\boldsymbol{z}^{(k+1)}}{\partial\boldsymbol{y}^{(k)}}\frac{\partial\boldsymbol{y}^{(k)}}{\partial\boldsymbol{z}^{(k)}} = \frac{\partial\boldsymbol{y}^{(k)}}{\partial\boldsymbol{z}^{(k)}}\boldsymbol{W}^{(k+1)\top}\frac{\partial\mathcal{L}(\boldsymbol{\theta})}{\partial\boldsymbol{z}^{(k+1)}} \tag{2.37}$$

While the loss function is originally defined over all the training examples, the gradient can be efficiently computed over a random batch of training examples. The method is called Stochastic Gradient Descent (SGD) [15]. This consists of feeding in the input vector for a few examples, computing the outputs and the errors, computing the average gradient for those examples, and adjusting the weights accordingly. The process is repeated for many small sets of examples from the training set until the average of the loss function stops decreasing. It is called stochastic because each small set of examples gives a noisy estimate of the average gradient over all examples. This simple procedure usually finds a good set of weights quickly when compared with far more elaborate optimization techniques [100, 16]. The batch size usually requires tunning to be reasonable. If it is too small, the gradient estimate will be too noisy; if it is too large, estimating each gradient may take too long due to memory constraints.

The parameter update $\Delta\boldsymbol{\theta}[\tau]$ can be improved to avoid local minima degrading the performance, as well as to increase convergence speed. One type of improvement is to consider momentum when making a gradient-based update. An update considering momentum can be written as:

$$\Delta\boldsymbol{\theta}[\tau] = \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}|_{\boldsymbol{\theta}[\tau]} + \alpha\Delta\boldsymbol{\theta}[\tau-1] = \eta\nabla(\mathcal{L}(\boldsymbol{\theta}[\tau])) + \alpha\Delta\boldsymbol{\theta}[\tau-1] \tag{2.38}$$

$\nabla(\mathcal{L}(\boldsymbol{\theta}[\tau]))$ is a compact notation, and α is an additional tunable parameter. When momentum is considered, parameter changes over iterations are smoother.

Another type of improvement is to use an adaptive learning rate $\eta[\tau]$ instead of a fixed learning rate η. In general, a learning rate too small leads to slow convergence, and a learning rate too large leads to divergence. A range of approaches have been proposed. Nesterov [123] is shown in equation 2.39.

$$\Delta\boldsymbol{\theta}[\tau] = \eta\nabla(\mathcal{L}(\boldsymbol{\theta}[\tau] - \alpha\Delta\boldsymbol{\theta}[\tau-1])) + \alpha\Delta\boldsymbol{\theta}[\tau-1] \tag{2.39}$$

This approach introduces a look ahead: the gradient is evaluated after updating the parameters with momentum. Adagrad [42] is shown in equation 2.40, where ϵ is a small number ensuring positivity.

$$\Delta\boldsymbol{\theta}[\tau] = \eta\boldsymbol{\beta}_\tau \odot \nabla(\mathcal{L}(\boldsymbol{\theta}[\tau]));\ \beta_{\tau,i} = 1/\sqrt{\epsilon + \sum_{t=1}^{\tau} \nabla_i^2(\mathcal{L}(\boldsymbol{\theta}[t]))} \tag{2.40}$$

This approach adapts the learning rate for each parameter separately, but always decays the learning rate, which results in early stopping. Adam [86] is shown in equation 2.41, where β_1 and β_2 are additional tunable parameters.

$$\Delta\boldsymbol{\theta}_i[\tau] = \frac{\eta}{\sqrt{\sigma^2_{\tau i} + \epsilon}}\mu_{\tau i}; \begin{cases} \mu_{\tau i} = \beta_1\mu_{(\tau-1)i} + (1-\beta_1)\nabla_i(\mathcal{L}(\boldsymbol{\theta}[\tau])) \\ \sigma^2_{\tau i} = \beta_2\sigma^2_{(\tau-1)i} + (1-\beta_2)\nabla^2_i(\mathcal{L}(\boldsymbol{\theta}[\tau])) \end{cases} \tag{2.41}$$

This approach combines Adagrad and momentum; $\mu_{\tau i}$ is the decaying sum over gradients, which is more stable than a single gradient; $\sigma^2_{\tau i}$ decays the learning rate in the same way as Adagrad.

Pre-training techniques can also be used to avoid local minima and to increase convergence speed. Unsupervised learning procedures [35, 193] can create layers of feature detectors without requiring labeled data. The objective in learning each layer of feature detectors is to be able to reconstruct or model the activities of feature detectors or raw inputs in the layer below. By pre-training several layers of progressively more complex feature detectors using this reconstruction objective, the weights of a deep network could be initialized to more sensible values. A final layer of output units could then be added to the top of the network and the whole deep system could be finetuned using standard backpropagation [100, 11].

2.5 Chapter Summary

This chapter described the fundamentals of deep learning. Section 2.1 reviewed the basic building blocks, namely fully connected feedforward neural networks, convolutional neural networks and recurrent neural networks. Activation functions are then reviewed, covering sigmoid, tanh, ReLU, and softmax. Section 2.2 described attention mechanisms, including the general framework, self-attention and multi-head attention. Next, section 2.3 described Transformer blocks, which make extensive use of attention mechanisms. General training techniques were described in section 2.4, covering basic training criteria and gradient-based optimization. More advanced training techniques will be described in section 3.3, in the context of sequence-to-sequence modeling.

Chapter 3

Attention-based Sequence-to-sequence Models

Sequence-to-sequence (seq2seq) generation can be defined as the task of mapping an input sequence $\boldsymbol{x}_{1:L}$ to an output sequence $\boldsymbol{y}_{1:T}$ [9]. The two sequences do not need to be aligned or have the same length. Example tasks include Automatic Speech Recognition (ASR), Text-To-Speech (TTS) and Neural Machine Translation (NMT). From a probabilistic perspective, a model $\boldsymbol{\theta}$ estimates the distribution of $\boldsymbol{y}_{1:T}$ given $\boldsymbol{x}_{1:L}$. This can be formulated as

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1},\boldsymbol{x}_{1:L};\boldsymbol{\theta}) \tag{3.1}$$

for autoregressive models, and

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}) \approx \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{x}_{1:L};\boldsymbol{\theta}) \tag{3.2}$$

for non-autoregressive models.

This work mainly investigates autoregressive models, which are more general. When the back-history $\boldsymbol{y}_{1:t-1}$ does not contribute to the output, autoregressive models can be expected to produce the same output as their non-autoregressive counterparts, i.e. the two types of models can have the same performance in terms of the quality of the output. For non-autoregressive models, it is assumed that the output tokens are conditionally independent, which allows inference to be done in parallel across time. The assumption can be too strong and degrade the performance in many, although

Table 3.1 Default and alternative terms.

Default term	Alternative term(s)	Notation
input	source	$\boldsymbol{x}_{1:L}$
reference output	target	$\boldsymbol{y}_{1:T}$
generated output	prediction, hypothesis	$\hat{\boldsymbol{y}}_{1:T}$

not all, cases [57, 105]. To achieve similar performance to autoregressive models, non-autoregressive models often require more complicated training approaches [126, 135], which can be worth the effort considering the efficiency during inference.

Generally speaking, there are three fundamental questions: modeling, inference and training. How to model the conditional distribution $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})$? Given an input sequence, how to generate an output sequence from the model? Given some data, how to train the model? While all these aspects will be discussed, training is the focus of this chapter.

In the literature, there are alternative terms to refer to the input $\boldsymbol{x}_{1:L}$, the reference output $\boldsymbol{y}_{1:T}$ and the generated output $\hat{\boldsymbol{y}}_{1:T}$. Table 3.1 lists the most common alternative terms. In this work, the alternative terms are occasionally used to avoid frequent repetition. To improve readability, when there is no need to distinguish the reference output and the generated output, the term "output" is used, and the notation is $\boldsymbol{y}_{1:T}$. For example, when elaborating the structure of the model $\boldsymbol{\theta}$, $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})$ and $p(\hat{\boldsymbol{y}}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})$ are computed with the same operations. Hence it is stated that the model estimates the distribution of the output conditioned on the input.

3.1 Encoder-attention-decoder Architecture

3.1.1 Motivation

Sequence-to-sequence models usually consist of an encoder and a decoder. The encoder processes the input sequence, producing context vector(s) of a fixed size. The decoder takes the context vector(s) and generates the output sequence. For the first generation of sequence-to-sequence models [168], the encoder and the decoder are connected by a single context vector. Figure 3.1 illustrates such a sequence-to-sequence model. Let

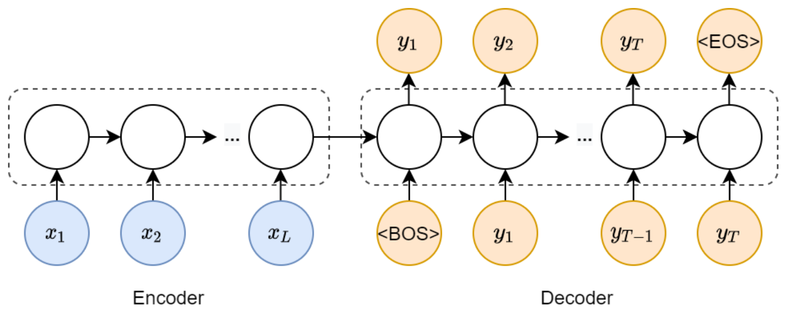

Fig. 3.1 Illustration of an encoder-decoder model without attention [168]. <BOS> and <EOS> are special tokens for the beginning and the end of the sequence.

$\boldsymbol{h}_{1:L}$ denote the encoder states, and $\boldsymbol{s}_{1:T}$ the decoder states. At decoding time step t[1], this model can be formulated as

$$\boldsymbol{h}_{1:L} = f(\boldsymbol{x}_{1:L}; \boldsymbol{\theta}_h) \tag{3.3}$$

$$\boldsymbol{s}_0 = \boldsymbol{h}_L; \quad \boldsymbol{s}_t = f(\boldsymbol{s}_0, \boldsymbol{y}_{1:t-1}; \boldsymbol{\theta}_s) = f(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}; \boldsymbol{\theta}_s) \tag{3.4}$$

$$\hat{\boldsymbol{y}}_t \sim p(\cdot|\boldsymbol{s}_t; \boldsymbol{\theta}_y) \tag{3.5}$$

This model $\boldsymbol{\theta}$ is split into three parts: $\{\boldsymbol{\theta}_y, \boldsymbol{\theta}_s, \boldsymbol{\theta}_h\}$; $\boldsymbol{\theta}_h$ is the encoder; $\boldsymbol{\theta}_y$ and $\boldsymbol{\theta}_s$ form the decoder. The encoder and decoder are both based on RNNs, described in section 2.1.3. The encoder is a bidirectional RNN, and equation 3.3 is equivalent to equations 2.9 to 2.11. The decoder is a unidirectional RNN, and its state update equation is equivalent to equation 2.7. The last state of the encoder is used as the context vector summarizing the entire input sequence. The context vector is passed to the decoder as its initial state.

A major disadvantage of using a single context vector is the incapability of memorizing long sequences [131]. Attention mechanisms [6, 54, 117] were introduced to address this issue. Rather than using the encoder's last hidden state as the fixed context vector, attention mechanisms create connections between a varying context vector and the entire input sequence. The weights of these connections are different for each output token, and can be interpreted as the alignment between the input and output. With attention mechanisms, the dependencies between input and output tokens can be modeled with much more precision. Note that there are alternative approaches to aligning two sequences. For example, duration models [149, 195, 39] can be used to replace attention mechanisms in speech synthesis, which will be discussed in more details in section 6.1.2.

[1]In this work, t denotes decoding time steps, and l denotes encoding time steps.

Attention mechanisms have been described in section 2.2.1. Figure 2.6 shows an attention-based encoder-decoder model. The key difference between this model and the one shown in figure 3.1 is that it connects the encoder and decoder with an attention mechanism. In this case, equation 3.4 becomes

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t; \boldsymbol{\theta}_s) \tag{3.6}$$

where $\boldsymbol{c}_t$ is the time-dependent context vector produced by the attention mechanism, as shown in equations 2.16 and 2.17. $\boldsymbol{s}_0$ is initialized independently of the input sequence.

3.1.2 Framework

The previous section briefly described an example attention-based sequence-to-sequence model. This section formulates this type of models in a more general fashion. Recall that this thesis focuses on autoregressive models that adopt the encoder-attention-decoder architecture. For these models, it is central to estimate $p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta})$:

$$\begin{aligned} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) &\approx p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \\ &\approx p(\boldsymbol{y}_t|\boldsymbol{s}_t, \boldsymbol{c}_t; \boldsymbol{\theta}_y) \end{aligned} \tag{3.7}$$

$\boldsymbol{\theta} = \{\boldsymbol{\theta}_y, \boldsymbol{\theta}_s, \boldsymbol{\theta}_\alpha, \boldsymbol{\theta}_h\}$. $\boldsymbol{\alpha}_t$ is an alignment vector, i.e. a set of attention weights. $\boldsymbol{s}_t$ is a state vector representing the output history $\boldsymbol{y}_{1:t-1}$, and $\boldsymbol{c}_t$ is a context vector summarizing $\boldsymbol{x}_{1:L}$ for time step t.

Figure 3.2 shows a general encoder-attention-decoder model. This can be viewed as a generalized version of figure 2.6, where there is no assumption about the building blocks. The following equations give more details about how $\boldsymbol{\alpha}_t$, $\boldsymbol{s}_t$ and $\boldsymbol{c}_t$ can be computed:

$$\boldsymbol{h}_{1:L} = f(\boldsymbol{x}_{1:L}; \boldsymbol{\theta}_h) \tag{3.8}$$

$$\boldsymbol{s}_t = f(\boldsymbol{y}_{1:t-1}; \boldsymbol{\theta}_s) \tag{3.9}$$

$$\boldsymbol{\alpha}_t = f(\boldsymbol{s}_t, \boldsymbol{h}_{1:L}; \boldsymbol{\theta}_\alpha) \quad \boldsymbol{c}_t = \sum_{l=1}^{L} \alpha_{t,l} \boldsymbol{h}_l \tag{3.10}$$

$$\hat{\boldsymbol{y}}_t \sim p(\cdot|\boldsymbol{s}_t, \boldsymbol{c}_t; \boldsymbol{\theta}_y) \tag{3.11}$$

The encoder maps $\boldsymbol{x}_{1:L}$ to $\boldsymbol{h}_{1:L}$, considering information from the entire input sequence; $\boldsymbol{s}_t$ summarizes $\boldsymbol{y}_{1:t-1}$, considering only the past. The corresponding equations are

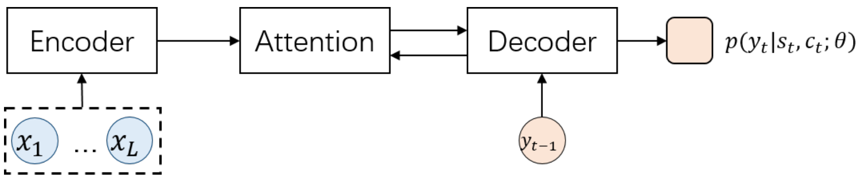

Fig. 3.2 Illustration of a general attention-based encoder-decoder model, operating in teacher forcing mode; a circle depicts a token, and a rounded square depicts a distribution.

3.8 and 3.9, which are applicable to various types of building blocks introduced in chapter 2. With $\boldsymbol{h}_{1:L}$ and $\boldsymbol{s}_t$, the attention mechanism computes $\boldsymbol{\alpha}_t$, and then $\boldsymbol{c}_t$. The corresponding equation is 3.10, which is a compact version of equations 2.16 and 2.17, and covers various forms of attention. Finally, the decoder estimates a distribution based on $\boldsymbol{s}_t$ and $\boldsymbol{c}_t$, and optionally generates an output token $\hat{\boldsymbol{y}}_t$.[2]

3.1.3 Design Considerations

The encoder, attention and decoder can be built with the building blocks described in chapter 2, namely feedforward layers, recurrent layers, convolutional layers and Transformer layers. The choice of the building blocks largely determines the model's capacity and efficiency. In sequence-to-sequence models, feedforward layers are usually combined with other types of building blocks, and operate independently at different positions of a sequence [2, 182, 175]. This subsection will first discuss the general pros and cons of different building blocks, and then describe how they can be used to construct the encoder, attention and decoder.

In theory, recurrent layers (RNNs) have an infinite receptive field, and belong to the most expressive members of the neural network family [161]. They excel at modeling sequential data [46, 55], and generalize relatively well [34]. In practice, their receptive field is limited by the vanishing / exploding gradient problem [12]. The most successful RNN-based models use LSTMs [67] or GRUs [27] to address this problem [170]. In addition, residual [63] or highway [166] connections are often used to facilitate training. The recurrent connections in RNNs make them unsuitable for parallel training across time. On the other hand, these connections reduce the amount of computation needed

[2]When computing the decoder state, the context vector can be optionally considered: $\boldsymbol{s}_t = f(\boldsymbol{y}_{1:t-1}, \boldsymbol{c}_{t-1}; \boldsymbol{\theta}_s)$. For the discussions in this chapter, it is not crucial whether the context vector is included.

to model a sequence. This makes RNNs memory efficient [24], and fast at inference stage [70], where the reference output is not available.

Convolutional layers (CNNs) have a finite receptive field, and are adept at capturing local correlations [100]. For each layer, the receptive field is fixed and is relatively small. This problem can be addressed by using dilated convolution [174]. Stacking many convolutional layers also alleviates this problem, at the expense of making training more difficult. For CNNs, it is also common to use residual [63] or highway [166] connections. For purely CNN-based models, a major advantage is parallel training [49]. When the reference output sequence is shifted and fed into the model, there are no recurrent connections and training can be run in parallel across time.

Similar to CNNs, Transformer layers have no recurrent connections, and can be trained in parallel. The issue of limited receptive field is addressed by the self-attention networks, which allows each position in the current layer to have access to information from all other positions in the previous layer. Due to the absence of recurrence, positional encodings are added to the input and output embeddings. Similar to the time step in a recurrent layer, the positional encodings inform the Transformer layers of the order of input and output tokens. Compared with RNNs and CNNs, a down-side of Transformer layers is their ability to scale w.r.t. the length of the input. The memory and computational complexity required to compute the attention matrix is quadratic in the input sequence length [172]. Several approaches have been proposed to address this issue, such as fixed patterns [145] and low-rank methods [181].

Encoder

The encoder can be built with all types of building blocks previously described. Compared with the decoder case to be discussed, there are few restrictions. The recurrent layers can be either unidirectional or bidirectional, and the convolutional layers are not required to be causal. As described in section 2.3, there are two types of Transformer layers, designed respectively for encoders and decoders [175]. Here the Transformer encoder layers should be used. These layers each have two main sub-layers: multi-head self-attention, followed by a feedforward network.

Attention

How to construct the encoder-decoder attention depends largely on the score function, described in section 2.2.1. RNNs are essential for the attention mechanisms whose score function is recurrent, e.g. location-sensitive attention [28]. In contrast, it is less common to use CNNs in attention mechanisms, as most score functions do not involve convolution, but there are exceptions [28], such as the score function in equation 2.19. For Transformer-based models, the encoder-decoder attention is merged into the decoder. There are multiple attention mechanisms. To allow parallel training, these attention mechanisms are normally feedforward. A standard choice is the scaled dot-product attention, shown in table 2.1.

Decoder

The decoder can be built with the same types of building blocks as the encoder. For autoregressive models, the decoding process normally follows a predetermined order, e.g. left-to-right. Therefore, the recurrent layers should be unidirectional, and the convolutional layers should be causal. When using Transformer decoder layers, the self-attention should be masked, to prevent attending to subsequent positions. Recall that compared with the Transformer encoder layer, the Transformer decoder layer has an extra cross attention in the middle, which is free from this restriction. For both the encoder and the decoder, it is common to stack multiple Transformer layers[175]. Typically, the layers have the same structure but different parameters, although there are exceptions, where the layers are tied to help the model generalize [34].

3.1.4 Application Considerations

When applying sequence-to-sequence models to a specific task, it is important to adapt the models to the nature of the task. On the embedding side, it is common to embed the input and output tokens as vectors of continuous elements [182, 21, 186]. If an input or output sequence is continuous, such as spectrogram, the raw tokens already meet the requirement. If the sequence is discrete, such as text, the raw tokens are one-hot vectors. A learnable embedding matrix is often used to map these tokens to continuous vectors [182, 186].

On the output prediction side, the distribution to use also depends on the continuity of the output sequence. For discrete outputs, the distribution of a token is usually modeled by a categorical distribution. The output layer is hence a linear layer, which controls the dimensionality, followed by a softmax function. For continuous outputs, the output layer is usually a fully connected feedforward layer, which estimates the mean of a continuous distribution, such as Gaussian or Laplace distribution. The log of the probability of a token, shown in equation 3.7, is proportional to some distance between the reference and the predicted mean, e.g. L_2 distance for Gaussian distribution and L_1 distance for Laplace distribution.

3.2 Inference

During inference, given an input $\boldsymbol{x}_{1:L}$, the output $\hat{\boldsymbol{y}}_{1:T}$ can be obtained from the distribution estimated by the model $\boldsymbol{\theta}$:

$$\hat{\boldsymbol{y}}_{1:T} = \operatorname*{argmax}_{\boldsymbol{y}_{1:T}} p(\boldsymbol{y}_{1:T} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{3.12}$$

The exact search is often too expensive and is often approximated by greedy search for continuous output, or beam search for discrete output [9]. Greedy search is done by generating the full sequence one token at a time, conditioned on previously generated tokens:

$$\hat{\boldsymbol{y}}_t = \operatorname*{argmax}_{\boldsymbol{y}_t} p(\boldsymbol{y}_t | \hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}); \quad t \geq 1 \tag{3.13}$$

$\hat{\boldsymbol{y}}_0$ is a fixed special token <BOS>, which is the beginning of the sequence but is often removed in the final output. The inference process stops when an <EOS> token is generated.

Beam search explores the output space more thoroughly, and yields B "best" sequences. This is done by maintaining a beam of B best candidate sequences. At each time step new candidates are generated by extending each candidate by one token and adding them to the beam. At the end of the step, the beam is re-pruned to only keep B candidates. The search is truncated when no new sequences are added, and B sequences are returned.

Greedy search is suitable for both continuous output, such as speech, and discrete output, such as text. Beam search is harder to use for continuous output, since the number of candidates that can be kept is extremely small compared with the search space [9]. For both approaches, the deterministic selection of the most likely token can be replaced by sampling, or combined with random noise.

3.3 Training

Ideally, the model is trained through minimizing the KL-divergence between the true distribution $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L})$ and the estimated distribution $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})$:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\boldsymbol{x}_{1:L} \sim p(\boldsymbol{x}_{1:L})} \mathrm{KL}\Big(p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L})||p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})\Big) \tag{3.14}$$

where $\mathcal{L}(\boldsymbol{\theta})$ denotes the loss. In practice, this is approximated by minimizing the Negative Log-Likelihood (NLL) over some training data $\{\boldsymbol{y}_{1:T}^{(n)}, \boldsymbol{x}_{1:L}^{(n)}\}_1^N$, sampled from the true distribution:

$$\mathcal{L}(\boldsymbol{\theta}) \propto -\sum_{n=1}^{N} \log p(\boldsymbol{y}_{1:T}^{(n)}|\boldsymbol{x}_{1:L}^{(n)};\boldsymbol{\theta}) \tag{3.15}$$

N denotes the size of the training dataset; n denotes the data index. To simplify the notation, the data index is omitted for the length of the sequences, although they also vary with the index. In the following sections, the sum over the training set $\sum_{n=1}^{N}$ will also be omitted.

3.3.1 Teacher Forcing

Recall that this thesis focuses on autoregressive models, and the sequence distribution $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})$ is factorized across time, as shown in equation 3.1. A key question then, is how to compute the token distribution $p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1},\boldsymbol{x}_{1:L};\boldsymbol{\theta})$. For teacher forcing, at each time step t, the token distribution is computed with the correct output history $\boldsymbol{y}_{1:t-1}$. In this case, the loss can be written as:

$$\mathcal{L}_y^{\mathtt{T}}(\boldsymbol{\theta}) = -\log p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1},\boldsymbol{x}_{1:L};\boldsymbol{\theta}) \tag{3.16}$$

From a theoretical point of view, this approach yields the correct model (zero KL-divergence) if the following assumptions hold: 1) the model is powerful enough ; 2) the model is optimized correctly; 3) there is enough training data to approximate the expectation shown in equation 3.14. In practice, these assumptions are often not true, hence the model is prone to mistakes. To illustrate the problem, suppose there is a reference output $\boldsymbol{y}^*_{1:T}$ for the test input $\boldsymbol{x}^*_{1:L}$. Due to data sparsity in high-dimensional space, $\boldsymbol{x}^*_{1:L}$ is likely to be unseen during training. If at time step t, the probability $p(\boldsymbol{y}^*_t|\boldsymbol{y}^*_{1:t-1}, \boldsymbol{x}^*_{1:L}; \boldsymbol{\theta})$ is wrongly estimated to be small, the probability of the reference output sequence $p(\boldsymbol{y}^*_{1:T}|\boldsymbol{x}^*_{1:L}; \boldsymbol{\theta})$ will also be small, i.e. it will be unlikely for the model to generate $\boldsymbol{y}^*_{1:T}$.

From a different perspective, teacher forcing suffers from exposure bias [148], which refers to the following problem. During training, the model is guided by the reference output history. At inference stage, however, the generated output history must be used. As previously analyzed, the model is often unable to recover the data distribution. Hence there is a train-inference mismatch, and the errors accumulate along the inference process [148].

3.3.2 Scheduled Sampling and Professor Forcing

To address exposure bias, the model could be trained in free running mode, where the generated output history $\hat{\boldsymbol{y}}_{1:t-1}$ is used, in instead of the reference output history $\boldsymbol{y}_{1:t-1}$. The corresponding loss function is

$$\mathcal{L}^{\mathtt{F}}_y(\boldsymbol{\theta}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{3.17}$$

This can be interpreted as using a noisy back-history to approximately estimate the token distribution:

$$p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \approx p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{3.18}$$

The problem with this approach is that training often struggles to converge [9]. The approximation is often not good enough, due to the model distribution being too different from the data distribution, especially at the early iterations. Intuitively, the back-history is too noisy for the model to learn to predict the next token. Therefore, several approaches, namely scheduled sampling and professor forcing, are proposed to train the model in a mode between teacher forcing and free running.

Scheduled sampling [9] randomly decides, for each time step, whether the reference or generated output token is added to the output history $\tilde{\boldsymbol{y}}_{1:t-1}$. For this approach, the loss function is

$$\mathcal{L}_y^{\mathrm{S}}(\boldsymbol{\theta}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t | \tilde{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{3.19}$$

$$\tilde{\boldsymbol{y}}_t = \begin{cases} \boldsymbol{y}_t & \text{with probability} \quad \epsilon \\ \hat{\boldsymbol{y}}_t & \text{with probability} \quad 1-\epsilon \end{cases} \tag{3.20}$$

ϵ gradually decays from 1 to 0 with a heuristic schedule. Considering that during training, $\tilde{\boldsymbol{y}}_{1:t-1}$ is mostly an inconsistent mixture of the reference output and the generated output, a natural variation is sequence-level scheduled sampling [9], where the decision is made for each sequence instead of token:

$$\tilde{\boldsymbol{y}}_{1:t-1} = \begin{cases} \boldsymbol{y}_{1:t-1} & \text{with probability} \quad \epsilon \\ \hat{\boldsymbol{y}}_{1:t-1} & \text{with probability} \quad 1-\epsilon \end{cases} \tag{3.21}$$

In terms of performance boost, token-level scheduled sampling is more consistent than sequence-level scheduled sampling [9]. However, even for token-level scheduled sampling, both positive [9] and negative [182] results have been reported. One concern is that the decay schedule does not fit the learning pace of the model.

Professor forcing [97] is a bundle of teacher forcing and generative adversarial training [52]. During training, the model $\boldsymbol{\theta}$ is viewed as a generator, which generates two output sequences for each input sequence, respectively in teacher forcing mode and free running mode[3]. For the training example $\{\boldsymbol{y}_{1:T}, \boldsymbol{x}_{1:L}\}$, let $\check{\boldsymbol{y}}_{1:T}$ denote the output generated in teacher forcing mode, and $\hat{\boldsymbol{y}}_{1:T}$ the output generated in free running mode, this can be expressed as:

$$\forall_t \; \check{\boldsymbol{y}}_t \sim p(\boldsymbol{y}_t | \boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{3.22}$$

$$\forall_t \; \hat{\boldsymbol{y}}_t \sim p(\boldsymbol{y}_t | \hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{3.23}$$

In addition to the final output, some intermediate output sequences are saved. Let $\check{\boldsymbol{z}}_{1:T}$ and $\hat{\boldsymbol{z}}_{1:T}$ denote the intermediate output sequences generated respectively in

[3]The term "teacher forcing", as well as "attention forcing", can refer to either an operation mode, or the approach to train a model in that operation mode. An operation mode can be used not only to train a model, but also to generate from it. For example, in teacher forcing mode, given the reference output $\boldsymbol{y}_{1:T}$, a model can generate a guided output $\check{\boldsymbol{y}}_{1:T}$, without evaluating the loss. $\check{\boldsymbol{y}}_{1:T}$ is likely to be different but similar to $\boldsymbol{y}_{1:T}$, and can be useful for training the discriminator.

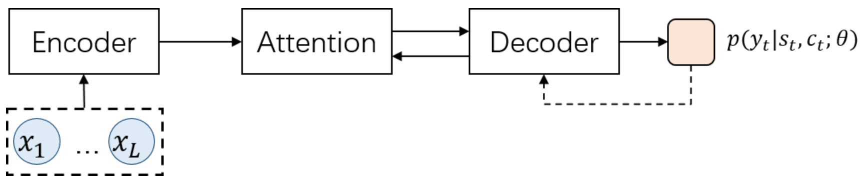

Fig. 3.3 Illustration of free running; a token is generated from the model distribution and fed back into the model; the dashed arrow indicates indirect connection and time delay.

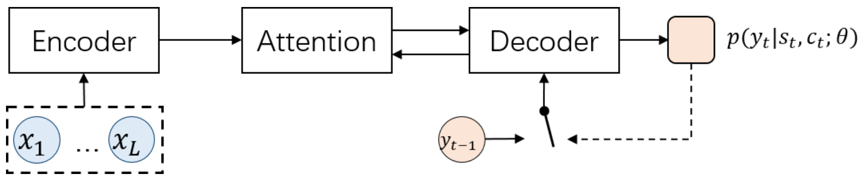

Fig. 3.4 Illustration of scheduled sampling / professor forcing; either the generated token or the reference token is fed into the model.

teacher forcing and free running mode. These generated sequences form a dataset $\{\check{\boldsymbol{y}}_{1:T}, \check{\boldsymbol{z}}_{1:T}, \hat{\boldsymbol{y}}_{1:T}, \hat{\boldsymbol{z}}_{1:T}\}_1^N$ that is used to train a discriminator $\boldsymbol{\psi}$. $\boldsymbol{\psi}$ is trained to predict the probability that a group of sequences is generated in teacher forcing mode, and the loss function is:

$$\mathcal{L}_{\psi}(\boldsymbol{\psi}|\boldsymbol{\theta}) = -\Big(\log\Big(f(\check{\boldsymbol{y}}_{1:T}, \check{\boldsymbol{z}}_{1:T}; \boldsymbol{\psi})\Big) + \log\Big(1 - f(\hat{\boldsymbol{y}}_{1:T}, \hat{\boldsymbol{z}}_{1:T}; \boldsymbol{\psi})\Big)\Big) \tag{3.24}$$

While this loss function is optimized w.r.t. $\boldsymbol{\psi}$, it depends on $\boldsymbol{\theta}$, hence the notation $\boldsymbol{\psi}|\boldsymbol{\theta}$. For the generator $\boldsymbol{\theta}$, there are three training objectives. The first one is the standard likelihood shown in equation 3.16. The second one is to fool the discriminator in free running mode:

$$\mathcal{L}_{\theta}^{\mathtt{F}}(\boldsymbol{\theta}|\boldsymbol{\psi}) = -\log\Big(f(\hat{\boldsymbol{y}}_{1:T}, \hat{\boldsymbol{z}}_{1:T}; \boldsymbol{\psi})\Big) \tag{3.25}$$

The third one, which is optional, is to fool the discriminator in teacher forcing mode:

$$\mathcal{L}_{\theta}^{\mathtt{T}}(\boldsymbol{\theta}|\boldsymbol{\psi}) = -\log\Big(1 - f(\check{\boldsymbol{y}}_{1:T}, \check{\boldsymbol{z}}_{1:T}; \boldsymbol{\psi})\Big) \tag{3.26}$$

This approach makes the distribution $p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta})$ estimated in free running mode similar to the corresponding distribution $p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta})$ estimated in teacher forcing mode. In addition, it regularizes some hidden layers, encouraging them to behave as if in teacher forcing mode. The disadvantage is that it requires designing and training the discriminator.

Intuitively, scheduled sampling and professor forcing are in the middle of teacher forcing and free running. To illustrate the intuition, recall that figure 3.2 shows teacher forcing. In a similar fashion, figure 3.3 shows free running; figure 3.4 shows scheduled sampling / professor forcing. Although the discussions in this section apply to all autoregressive models, the example model is attention-based. This is to facilitate the comparison with the approaches to be introduced in the following chapters.

3.3.3 Sequence-level Training

For the above training approaches, the model is trained at the token-level, i.e. the loss is computed for each token and summed across time. An alternative way of addressing exposure bias is to train the model at the sequence-level. Here the loss is computed for sequences instead of tokens, and the model sees not only the reference output during training. This type of approaches can be described in the framework of Minimum Bayes Risk (MBR) training. Assume that there is a distance metric $\mathcal{D}(\boldsymbol{y}_{1:T}, \underline{\boldsymbol{y}}_{1:\underline{T}})$ between the reference output $\boldsymbol{y}_{1:T}$ and a random output $\underline{\boldsymbol{y}}_{1:\underline{T}}$, whose probability is estimated with the model $\boldsymbol{\theta}$. $\mathcal{D}$ is minimal when the two sequences are equal. MBR training minimizes its expected value:

$$\mathcal{L}_y^{\mathsf{B}}(\boldsymbol{\theta}) = \sum_{\underline{\boldsymbol{y}}_{1:\underline{T}} \in \mathcal{Y}} p(\underline{\boldsymbol{y}}_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \mathcal{D}(\boldsymbol{y}_{1:T}, \underline{\boldsymbol{y}}_{1:\underline{T}}) \tag{3.27}$$

where $\mathcal{Y}$ is the entire output space. Intuitively, MBR training considers not just the reference output, but all possible outputs, although the range is often reduced in practice. In contrast, teacher forcing only considers the reference output; its loss, shown in equation 3.16, is equivalent to equation 3.27 if the distance metric $\mathcal{D}$ is the following:

$$\mathcal{D}(\boldsymbol{y}_{1:T}, \underline{\boldsymbol{y}}_{1:\underline{T}}) = \begin{cases} -1 & \text{if } \boldsymbol{y}_{1:T} = \underline{\boldsymbol{y}}_{1:\underline{T}} \\ 0 & \text{otherwise} \end{cases} \tag{3.28}$$

For sequence-to-sequence tasks, the number of correct outputs can be one, finitely many or infinitely many. For example, in ASR, there is usually one correct transcription; in NMT, there are often multiple valid translations; in TTS, there are infinitely many valid audio samples for the same text. Therefore, it can often be beneficial to consider non-reference outputs during training.

MBR training allows directly optimizing any distance metric $\mathcal{D}$ [148, 5]. $\mathcal{D}$ can be non-differentiable, and can be at sequence-level, i.e. $\boldsymbol{y}_{1:T}$ and $\underline{\boldsymbol{y}}_{1:\underline{T}}$ do not need to be aligned.[4] Typically, $\mathcal{D}$ is proportional to the evaluation metric [159]. Examples include word error rate for ASR, and BLEU [129] and ROUGE [107] for NMT.

Choosing an appropriate distance metric can be challenging, for tasks where the performance is not objective. For example, NMT models trained with one metric may not perform equally well in other metrics [148]. For TTS, the choice is even more complicated, as there is no well-established objective metric [179]. Generative adversarial training [197, 185] can be used to tackle this issue. Here the distance metric is learned by a discriminator, which is potentially better than hand-crafted metrics.

Another major challenge of MBR training is how to optimize the loss $\mathcal{L}_y^{\mathtt{B}}(\boldsymbol{\theta})$ shown in equation 3.27. The expectation is often impossible to compute, due to the output space being too large. Approximations are often applied to the loss [148, 159] or the gradients w.r.t. the model parameters [187]. In reference [159], the distribution is approximated by considering a subset of the output space:

$$\mathcal{L}_y^{\mathtt{B}}(\boldsymbol{\theta}) \approx \sum_{\underline{\boldsymbol{y}}_{1:\underline{T}} \in \widetilde{\mathcal{Y}}} \widetilde{p}(\underline{\boldsymbol{y}}_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \mathcal{D}(\boldsymbol{y}_{1:T}, \underline{\boldsymbol{y}}_{1:\underline{T}}) \tag{3.29}$$

$\widetilde{\mathcal{Y}}$ denotes the sub-space, constructed by using beam search or a noisy version of greedy search multiple times [142]; $\widetilde{p}$ denotes the approximate distribution, defined as

$$\widetilde{p}(\underline{\boldsymbol{y}}_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) = \frac{p(\underline{\boldsymbol{y}}_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta})^{\gamma}}{\sum_{\underline{\boldsymbol{y}}'_{1:\underline{T}} \in \widetilde{\mathcal{Y}}} p(\underline{\boldsymbol{y}}'_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta})^{\gamma}} \tag{3.30}$$

where γ is a hyperparameter that controls the sharpness of the approximate distribution. This approach is relatively simple to implement, but introduces an extra hyperparameter. Empirically, it requires generating more samples than the following Monte Carlo approach [159, 148, 187].

[4]In this work, the convention is to omit the accents of the superscripts and subscripts. For example, $\boldsymbol{y}_{1:T}$ denotes the reference output, whose length is T. $\hat{\boldsymbol{y}}_{1:\hat{T}}$ denotes the generated output, whose length is $\hat{T}$, but the accent is omitted by default, so the notation becomes $\hat{\boldsymbol{y}}_{1:T}$. In $\underline{\boldsymbol{y}}_{1:\underline{T}}$, which denotes the random output to be averaged, the accent of the subscript is not omitted, in order to emphasize that the distance metric $\mathcal{D}$ is between two unaligned sequences of different length.

Using Monte Carlo methods, the loss shown in equation 3.27 can also be approximated as:

$$\mathcal{L}_y^{\mathtt{B}}(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{m=1}^{M} \mathcal{D}(\boldsymbol{y}_{1:T}, \hat{\boldsymbol{y}}_{1:T}^{(m)}) \tag{3.31}$$

where $\hat{\boldsymbol{y}}_{1:T}^{(m)}$ is sampled from the distribution estimated by the model, and M is the number of samples. Each sample has its own length, but the accent is omitted to simplify the notation. The sampling process is usually approximated by either greedy search or beam search [148]. If the approximate loss is differentiable w.r.t the samples, it can be optimized using reparameterization tricks, such as the Gumbel softmax [78]. For more general cases, where the loss is not necessarily differentiable, Reinforcement Learning (RL) [148, 197] can be used. Here the model is viewed as an agent; its parameters define a policy; its action refers to predicting the next token; the reward is the negative of the distance metric, observed at the end of the sequence. A major drawback with reinforcement learning is the large action space, where it is extremely hard for a random policy to improve in any reasonable amount of time [148]. A common remedy is to initialize the policy with a well-performing model, e.g. a model trained with teacher forcing.

Alternatively, Monte Carlo methods can be applied to the gradients. The exact gradients of $\mathcal{L}_y^{\mathtt{B}}(\boldsymbol{\theta})$ can be derived as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_y^{\mathtt{B}}(\boldsymbol{\theta}) = \sum_{\underline{\boldsymbol{y}}_{1:\underline{T}} \in \mathcal{Y}} p(\underline{\boldsymbol{y}}_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\underline{\boldsymbol{y}}_{1:\underline{T}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \mathcal{D}(\boldsymbol{y}_{1:T}, \underline{\boldsymbol{y}}_{1:\underline{T}}) \tag{3.32}$$

The derivation uses the identity $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = f(\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta})$. The Monte Carlo approximate gradients are

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_y^{\mathtt{B}}(\boldsymbol{\theta}) \approx \frac{1}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}} \log p(\hat{\boldsymbol{y}}_{1:T}^{(m)} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \mathcal{D}(\boldsymbol{y}_{1:T}, \hat{\boldsymbol{y}}_{1:T}^{(m)}) \tag{3.33}$$

This approach does not make assumptions about the output space, and does not require optimization tricks such as reparameterization [187] or the use of reinforcement learning.

3.3.4 Parallel Training

Despite suffering from exposure bias, teacher forcing has a major advantage: it allows parallel training across time [51, 175], when there are no recurrent connections in the model. Models based on CNNs or Transformers meet this condition, and often leverage the efficiency of teacher forcing to train on large amounts of data [174, 103].

For scheduled sampling and professor forcing, parallel training is not possible without approximations [43]. This is because the model needs to generate a hypothesis sequence during training, and the generation process is sequential. For sequence-level training approaches, this issue is even more severe, as the model is expected to generate several hypotheses, in order to approximate the loss or the gradients.

3.4 Chapter Summary

This chapter described autoregressive sequence-to-sequence models that adopt the encoder-attention-decoder architecture. Section 3.1 described the motivation behind such models and their general framework. Application considerations are then described, covering the embedding and output layers to use for different tasks. Section 3.2 described commonly used inference approaches, namely greedy search and beam search. Section 3.3 explored various training approaches, paving the way for the novel ideas to be introduced in the following chapters.

For autoregressive models, teacher forcing is a standard training approach, especially for parallelizable models such as Transformer. This approach suffers from exposure bias: during training the model is guided with the reference output, but the generated output must be used at inference stage. Exposure bias can be addressed by two lines of research. Along the first line, scheduled sampling and professor forcing guide a model with both the reference and the generated output history. To stabilize training, they depend on a heuristic schedule or an auxiliary classifier, which can be difficult to tune. The second line is a series of sequence-level training approaches, leveraging reinforcement learning, minimum risk training or generative adversarial training. Theses approaches guide a model with the generated output history, and optimizes a sequence-level criterion. The generation process is often sequential, which is undesirable for parallelizable models such as Transformer. Another downside is that many tasks, such as speech synthesis and voice conversion, do not have a well-established sequence-level criterion.

Chapter 4

Attention Forcing

The previous chapter described attention-based sequence-to-sequence models. In particular, various training approaches were explored. The standard approach, teacher forcing, suffers from exposure bias: during training the model is guided with the reference output, but the generated output must be used at inference stage. Scheduled sampling and professor forcing address this issue by using both the reference and the generated output history. However, they require a heuristic schedule or an auxiliary classifier, which can be difficult to tune. Sequence-level training is an alternative line of research, where the model is guided with the generated output, and a sequence-level criterion is optimized. The downside is that sequential decoding is often performed during training, which is inefficient for parallelizable models. Additionally, many tasks do not have a well-established sequence-level criterion.

This chapter introduces attention forcing, which guides the model with the generated output history and reference attention. This approach reduces the training-inference mismatch without the need for a heuristic schedule or a classifier. In addition, it does not require the sequence-to-sequence task to have a well-established sequence-level criterion. The following variations of attention forcing are introduced. (1) Scheduled attention forcing automatically selects a suitable training mode for each input-output pair. This is essential for tasks such as machine translation, where the output space is discrete and multi-modal in the sense that the given an input, the distribution of the corresponding output can be multi-modal. (2) Parallel attention forcing approximates the sequential generation of the output history with a parallel generation process. This facilitates applying attention forcing to models that can be trained in parallel across time, such as convolutional and Transformer-based models.

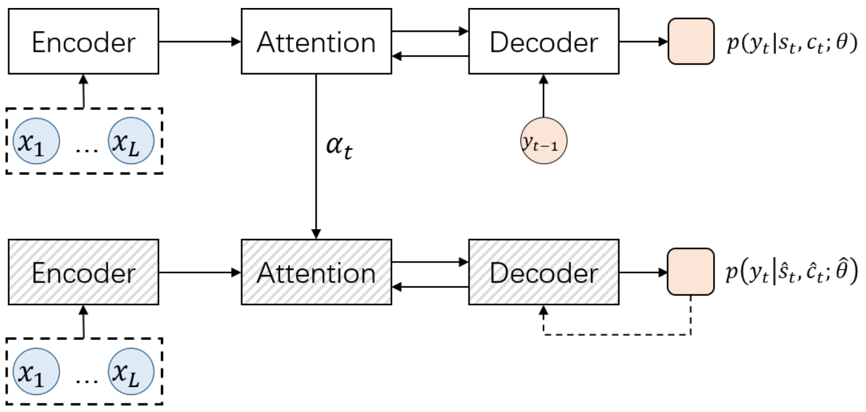

Fig. 4.1 General illustration of attention forcing; following the convention of section 3.1.2, the encoder, attention and decoder are packed as a single block; the shaded blocks form the attention forcing model; the dashed arrow indicates indirect connection and time delay.

4.1 Framework

The basic idea of attention forcing is to use the reference attention and generated output to guide the model during training. In attention forcing mode, the model does not need to learn to simultaneously infer the output and align it with the input. As the reference alignment is known, the decoder can focus on inferring the output, and the attention mechanism can focus on generating the correct alignment. From a different perspective, guiding the model with the generated output, instead of the reference output, helps addressing the exposure bias.

Let $\boldsymbol{\theta}$ denote a standard attention-based sequence-to-sequence model[1] , trained with teacher forcing. Let $\hat{\boldsymbol{\theta}}$ denote a model with the same structure, but trained with attention forcing, and later used for inference. Figure 4.1 illustrates attention forcing at a high level, following the convention of section 3.1.2. In attention forcing mode, the probability $p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}})$ is estimated with the generated output $\hat{\boldsymbol{y}}_{1:t-1}$ and the reference alignment $\boldsymbol{\alpha}_t$:

$$\begin{aligned} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) &\approx p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) \\ &\approx p(\boldsymbol{y}_t|\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{c}}_t; \hat{\boldsymbol{\theta}}_y) \end{aligned} \tag{4.1}$$

[1]In this thesis, the convention of notation is that a bold Greek letter without subscript denotes all the parameters of a model. The corresponding letters with subscripts denote a subset of the parameters. For example, $\boldsymbol{\theta} = \{\boldsymbol{\theta}_y, \boldsymbol{\theta}_s, \boldsymbol{\theta}_\alpha, \boldsymbol{\theta}_h\}$. $\boldsymbol{\theta}_y$ and $\boldsymbol{\theta}_s$ form the decoder; $\boldsymbol{\theta}_\alpha$ is the attention; $\boldsymbol{\theta}_h$ is the encoder.

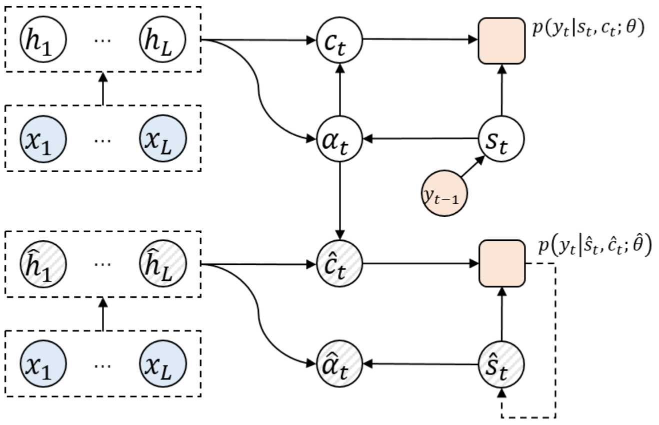

Fig. 4.2 Detailed illustration of attention forcing; the attention mechanism is unpacked.

$\hat{\boldsymbol{s}}_t$ and $\hat{\boldsymbol{c}}_t$ denote the state vector and context vector generated by $\hat{\boldsymbol{\theta}}$. Details of attention forcing are illustrated by figure 4.2, as well as the following equations:

$$\boldsymbol{h}_{1:L} = f(\boldsymbol{x}_{1:L}; \boldsymbol{\theta}_h) \qquad \hat{\boldsymbol{h}}_{1:L} = f(\boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}_h) \tag{4.2}$$

$$\boldsymbol{s}_t = f(\boldsymbol{y}_{1:t-1}; \boldsymbol{\theta}_s) \qquad \hat{\boldsymbol{s}}_t = f(\hat{\boldsymbol{y}}_{1:t-1}; \hat{\boldsymbol{\theta}}_s) \tag{4.3}$$

$$\boldsymbol{\alpha}_t = f(\boldsymbol{s}_t, \boldsymbol{h}_{1:L}; \boldsymbol{\theta}_\alpha) \qquad \hat{\boldsymbol{\alpha}}_t = f(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{h}}_{1:L}; \hat{\boldsymbol{\theta}}_\alpha) \tag{4.4}$$

$$\hat{\boldsymbol{c}}_t = \textstyle\sum_{l=1}^{L} \alpha_{t,l} \hat{\boldsymbol{h}}_l \tag{4.5}$$

$$\hat{\boldsymbol{y}}_t \sim p(\cdot|\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{c}}_t; \hat{\boldsymbol{\theta}}_y) \tag{4.6}$$

The right side of the equations 4.2 to 4.4, as well as equations 4.5 and 4.6, show how the attention forcing model $\hat{\boldsymbol{\theta}}$ operates. The decoder state $\hat{\boldsymbol{s}}_t$ is computed with $\hat{\boldsymbol{y}}_{1:t-1}$. While an alignment $\hat{\boldsymbol{\alpha}}_t$ is generated by $\hat{\boldsymbol{\theta}}$, it is not used by the decoder, because the context $\hat{\boldsymbol{c}}_t$ is computed with the reference alignment $\boldsymbol{\alpha}_t$. In most cases, $\boldsymbol{\alpha}_t$ is not available. One option of obtaining it is shown by the left side of equations 4.2 to 4.4: to generate $\boldsymbol{\alpha}_t$ from a teacher forcing model $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ is trained in teacher forcing mode, and generates $\boldsymbol{\alpha}_t$, also in teacher forcing mode. Although the reference output is used to compute the reference attention, it is not directly fed into the model, hence the model is unlikely to rely too much on the back-history. For example, when the output sequence is strongly correlated in time, attention forcing prevents the model from copying the back-history. Sections 6.4 and 7.3 will respectively demonstrate the effectiveness of attention forcing in speech synthesis and machine translation.

At inference stage, the attention forcing model operates in free running mode, as described in section 3.2. In this case, equation 4.5 becomes $\hat{\boldsymbol{c}}_t = \sum_{l=1}^{L} \hat{\alpha}_{t,l}\hat{\boldsymbol{h}}_l$. The decoder is guided by $\hat{\boldsymbol{\alpha}}_t$, instead of $\boldsymbol{\alpha}_t$.

4.2 Training

During training, there are two objectives: to infer the reference output and to imitate the reference alignment. This can be formulated as:[2]

$$\mathcal{L}_{y,\alpha}(\hat{\boldsymbol{\theta}}) = \mathcal{L}_y(\hat{\boldsymbol{\theta}}) + \gamma\mathcal{L}_\alpha(\hat{\boldsymbol{\theta}}) \tag{4.7}$$

$$\mathcal{L}_y(\hat{\boldsymbol{\theta}}) = -\textstyle\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) \tag{4.8}$$

$$\mathcal{L}_\alpha(\hat{\boldsymbol{\theta}}) = \textstyle\sum_{t=1}^{T}\mathrm{KL}(\boldsymbol{\alpha}_t||\hat{\boldsymbol{\alpha}}_t) = \displaystyle\sum_{t=1}^{T}\sum_{l=1}^{L} \alpha_{t,l} \log \frac{\alpha_{t,l}}{\hat{\alpha}_{t,l}} \tag{4.9}$$

where $\mathcal{L}_y$ and $\mathcal{L}_\alpha$ respectively denote the loss over the output and the attention; γ is a scaling factor, set according to the dynamic range of the two losses. As an alignment corresponds to a categorical distribution, KL-divergence is a natural difference metric. Our default optimization option is as follows. $\boldsymbol{\theta}$ is trained in teacher forcing mode, and then fixed to generate the reference attention. $\hat{\boldsymbol{\theta}}$ is trained with the joint loss $\mathcal{L}_{y,\alpha}$. This option makes training more stable, in the sense that training converges more often, most probably because the reference attention is the same in each epoch. An alternative is to train $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ simultaneously to save time. Another is to tie (parts of) $\boldsymbol{\theta}$ and $\hat{\boldsymbol{\theta}}$ to save memory.

An empirical issue is that as training progresses, most elements in the alignment vectors will be close to 0. So the KL attention loss, which involves the log of the elements, might be unstable. To improve numerical stability, a trick is to smooth all alignment vectors with a uniform distribution $\boldsymbol{u}$. For example, $\mathrm{KL}(\boldsymbol{\alpha}_t||\hat{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}})$ can be smoothed as $\mathrm{KL}((1-\epsilon)\boldsymbol{\alpha}_t + \epsilon\boldsymbol{u}||(1-\epsilon)\hat{\boldsymbol{\alpha}}_t + \epsilon\boldsymbol{u}; \hat{\boldsymbol{\theta}})$. ϵ is a small constant balancing the stability and accuracy of training; its default value is e^{-10} in our experiments.

The form of attention forcing described so far has two main problems. First, it may degrade the performance in tasks such as NMT, where output sequences can take

[2] In the previous chapter, a superscript, such as $^{\mathtt{T}}$ and $^{\mathtt{F}}$, is added to the notation of the loss, in order to differentiate various training approaches. If this convention were followed, the losses for attention forcing would be denoted as $\mathcal{L}_y^{\mathtt{A}}$ and $\mathcal{L}_\alpha^{\mathtt{A}}$. This chapter focuses on attention forcing, so the superscripts are omitted to simplify the notation.

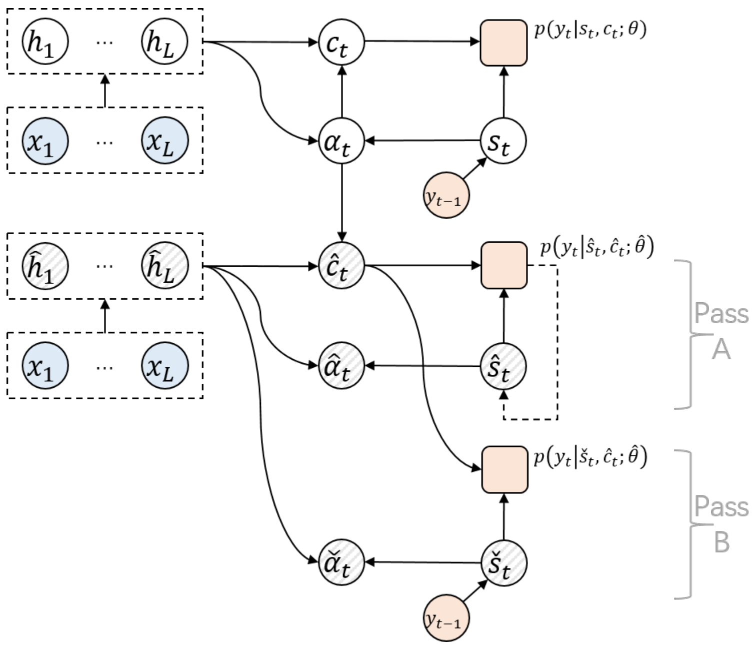

Fig. 4.3 Illustration of scheduled attention forcing. Passes A and B share the same model parameters.

multiple valid orderings and mistakes are more likely to be grave. Detailed analysis will be given in section 4.3. Second, it requires the model to generate sequentially during training, which is undesirable for Transformer-style models. The following sub-sections will introduce two variants of attention forcing to address these problems respectively.

4.2.1 Scheduled Attention Forcing

Scheduled attention forcing is proposed for applications where attention forcing, also referred to as “vanilla attention forcing”, may result in an inappropriate loss. The basic idea is to automatically decide, for each input-output pair in the training data, whether vanilla attention forcing will be used. This is realized by tracking the alignment between the reference and the generated output. If they are relatively well-aligned, vanilla attention forcing will be used, otherwise a more stable training mode will be used.

Figure 4.3 illustrates scheduled attention forcing. For each input sequence, the attention forcing model $\hat{\boldsymbol{\theta}}$ takes two forward passes. Pass A is guided by the generated output history $\hat{\boldsymbol{y}}_{1:t-1}$, which is the same as vanilla attention forcing. Pass B is guided by the reference output history $\boldsymbol{y}_{1:t-1}$. The reference attention is always used, so the context vector $\hat{\boldsymbol{c}}_t$ is the same in both passes. If memory permits, the two forward passes can be completed in parallel, resulting in no extra time. This can be formulated as follows. For vectors produced in pass A, the notation has the hat $\hat{}$ accent; the equivalent for pass B is the check $\check{}$ accent.

$$\boldsymbol{h}_{1:L} = f(\boldsymbol{x}_{1:L}; \boldsymbol{\theta}_h) \qquad \hat{\boldsymbol{h}}_{1:L} = f(\boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}_h) \tag{4.10}$$

$$\boldsymbol{s}_t = f(\boldsymbol{y}_{1:t-1}; \boldsymbol{\theta}_s) \qquad \begin{aligned} \hat{\boldsymbol{s}}_t &= f(\hat{\boldsymbol{y}}_{1:t-1}; \hat{\boldsymbol{\theta}}_s) \\ \check{\boldsymbol{s}}_t &= f(\boldsymbol{y}_{1:t-1}; \hat{\boldsymbol{\theta}}_s) \end{aligned} \tag{4.11}$$

$$\boldsymbol{\alpha}_t = f(\boldsymbol{s}_t, \boldsymbol{h}_{1:L}; \boldsymbol{\theta}_\alpha) \qquad \begin{aligned} \hat{\boldsymbol{\alpha}}_t &= f(\hat{\boldsymbol{s}}_t, \hat{\boldsymbol{h}}_{1:L}; \hat{\boldsymbol{\theta}}_\alpha) \\ \check{\boldsymbol{\alpha}}_t &= f(\check{\boldsymbol{s}}_t, \hat{\boldsymbol{h}}_{1:L}; \hat{\boldsymbol{\theta}}_\alpha) \end{aligned} \tag{4.12}$$

$$\hat{\boldsymbol{c}}_t = \textstyle\sum_{l=1}^{L} \alpha_{t,l} \hat{\boldsymbol{h}}_l \tag{4.13}$$

$$\begin{aligned} \hat{\boldsymbol{y}}_t &\sim p(\cdot | \hat{\boldsymbol{s}}_t, \hat{\boldsymbol{c}}_t; \hat{\boldsymbol{\theta}}_y) \\ \check{\boldsymbol{y}}_t &\sim p(\cdot | \check{\boldsymbol{s}}_t, \hat{\boldsymbol{c}}_t; \hat{\boldsymbol{\theta}}_y) \end{aligned} \tag{4.14}$$

Next, the choice of training mode is made at the sequence level, which ensures the consistency of the output history. If $\sum_{t=1}^{T} \mathrm{KL}(\boldsymbol{\alpha}_t || \hat{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}}) < \lambda \sum_{t=1}^{T} \mathrm{KL}(\boldsymbol{\alpha}_t || \check{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}})$, meaning that $\hat{\boldsymbol{y}}_{1:T}$ is well aligned with $\boldsymbol{y}_{1:T}$, forward pass A will be used in the back-propagation. The loss is the same as in vanilla attention forcing:

$$\mathcal{L}_{y,\alpha}(\hat{\boldsymbol{\theta}}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t | \boldsymbol{x}_{1:L}, \hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{\alpha}_t; \hat{\boldsymbol{\theta}}) + \gamma \sum_{t=1}^{T} \mathrm{KL}(\boldsymbol{\alpha}_t || \hat{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}}) \tag{4.15}$$

Otherwise forward pass B will be used:

$$\mathcal{L}_{y,\alpha}(\hat{\boldsymbol{\theta}}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t | \boldsymbol{x}_{1:L}, \boldsymbol{y}_{1:t-1}, \boldsymbol{\alpha}_t; \hat{\boldsymbol{\theta}}) + \gamma \sum_{t=1}^{T} \mathrm{KL}(\boldsymbol{\alpha}_t || \check{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}}) \tag{4.16}$$

The KL attention loss is used to determine if the alignment is good enough between the reference output $\boldsymbol{y}_{1:T}$ and the free running output $\hat{\boldsymbol{y}}_{1:T}$. As both $\boldsymbol{\alpha}_t$ and $\check{\boldsymbol{\alpha}}_t$ are computed using $\boldsymbol{y}_{1:t-1}$, they are expected to be similar, yielding a relatively small $\mathrm{KL}(\boldsymbol{\alpha}_t || \check{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}})$. In contrast, $\hat{\boldsymbol{\alpha}}_t$ is computed using $\hat{\boldsymbol{y}}_{1:t-1}$, and $\mathrm{KL}(\boldsymbol{\alpha}_t || \hat{\boldsymbol{\alpha}}_t; \hat{\boldsymbol{\theta}})$ is expected to be larger. λ is a hyper-parameter controlling how much out-of-alignment $\hat{\boldsymbol{y}}_{1:T}$ and

$\boldsymbol{y}_{1:T}$ can be. If $\lambda \to +\infty$, scheduled attention forcing will be the same as vanilla attention forcing.

For each pair of training data, scheduled attention forcing makes a choice whether to guide the model with the reference output history or the generated output history. This approach is named "scheduled attention forcing", because scheduled sampling also selectively uses the generated output history. For scheduled attention forcing, the selection is not random, but determined by the alignment of the two types of history. For scheduled sampling, the selection is random, as described in section 3.3.

4.2.2 Parallel Attention Forcing

Transformer-style models have achieved state-of-the-art performance in various tasks including NMT and TTS. For such models which have a large number of parameters, parallel training is essential. It is significantly more efficient than sequential training, and reduces the training time to a practical level. When teacher forcing is used, there are no recurrent connections in the model, and training can be done in parallel across the length T of the output $\boldsymbol{y}_{1:T}$. This is more obvious when teacher forcing is rewritten as follows:

$$\boldsymbol{\alpha}_t = f(\boldsymbol{y}_{1:t-1}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{4.17}$$

$$\hat{\boldsymbol{y}}_t \sim p(\cdot|\boldsymbol{y}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}) \tag{4.18}$$

Note that the reference output history $\boldsymbol{y}_{1:t-1}$ is available for any t, so $\hat{\boldsymbol{y}}_{1:T}$ can be computed in parallel. Attention forcing can be rewritten in a similar fashion:

$$\hat{\boldsymbol{\alpha}}_t = f(\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) \tag{4.19}$$

$$\hat{\boldsymbol{y}}_t \sim p(\cdot|\hat{\boldsymbol{y}}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) \tag{4.20}$$

The model is guided with generated output history $\hat{\boldsymbol{y}}_{1:t-1}$. $\hat{\boldsymbol{y}}_{1:T}$ is not available beforehand, and is generated sequentially, i.e. $\hat{\boldsymbol{y}}_t$ must be generated after $\hat{\boldsymbol{y}}_{t-1}$. So when applying attention forcing to Transformer-style models, training is no longer parallel. This sub-section introduces parallel attention forcing, an approximation of attention forcing that allows parallel training.[3]

[3]Transformer-style models have multiple encoder-decoder attention mechanisms. So when applied to these models, attention forcing involves a group of the reference attention $\boldsymbol{\alpha}_t^{(1:N,1:H)}$ and generated

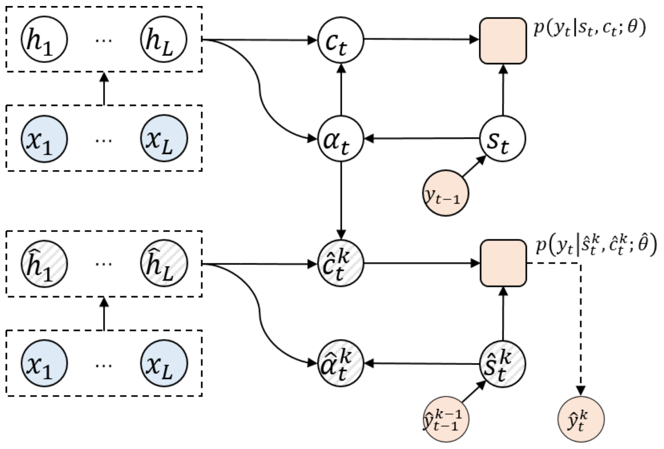

Fig. 4.4 Illustration of parallel attention forcing, at forward pass k.

Parallel attention forcing is inspired by parallel scheduled sampling [43]. The core idea is to approximate the sequential generation of $\hat{\boldsymbol{y}}_{1:T}$ with a parallelizable process. For parallel attention forcing, as well as parallel scheduled sampling, the output history $\hat{\boldsymbol{y}}_{1:T}^K$ is generated iteratively in K forward passes. For each forward pass, the complete output history is available beforehand, so training can be run in parallel across time $t = 1 : T$. This can be illustrated by figure 4.4.

For the first pass, the output history is the reference $\boldsymbol{y}_{1:T}$. For the following passes, the output history is the output of the previous pass $\hat{\boldsymbol{y}}_{1:T}^{k-1}$.

$$\hat{\boldsymbol{y}}_{1:T}^0 = \boldsymbol{y}_{1:T} \tag{4.21}$$

$$\hat{\boldsymbol{\alpha}}_t^k = f(\hat{\boldsymbol{y}}_{1:t-1}^{k-1}, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) \tag{4.22}$$

$$\hat{\boldsymbol{y}}_t^k \begin{cases} = & \hat{\boldsymbol{y}}_t^{k-1} & \text{if } t < k \\ \sim & p(\cdot|\hat{\boldsymbol{y}}_{1:t-1}^{k-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) & \text{if } t \geq k \end{cases} \tag{4.23}$$

Similar to scheduled attention forcing, multiple forward passes can be taken for each pair of training data. The loss function is selected based on the alignment between the reference and generated output sequences. If $\sum_{t=1}^{T} \text{KL}(\boldsymbol{\alpha}_t || \hat{\boldsymbol{\alpha}}_t^K; \hat{\boldsymbol{\theta}}) < \lambda \sum_{t=1}^{T} \text{KL}(\boldsymbol{\alpha}_t || \hat{\boldsymbol{\alpha}}_t^1; \hat{\boldsymbol{\theta}})$, it will be assumed that $\hat{\boldsymbol{y}}_{1:T}^K$ is well aligned with $\boldsymbol{y}_{1:T}$, and the

attention $\hat{\boldsymbol{\alpha}}_t^{(1:N,1:H)}$, where N is the number of decoder layers, and H the number of heads in each layer. These superscripts are omitted, to simplify the notation and to facilitate comparison with other forms of attention forcing.

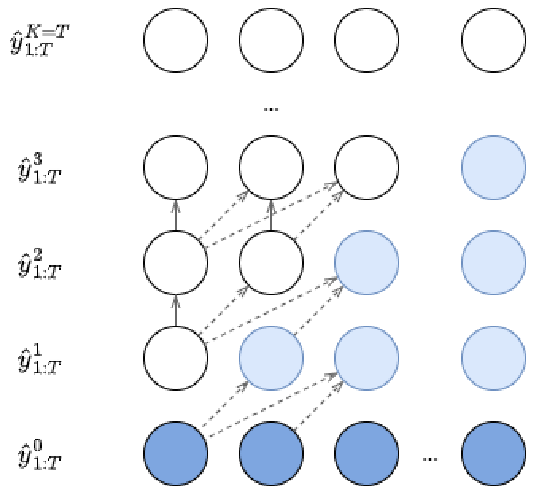

Fig. 4.5 Illustration of iterative parallel generation; the dark blue circles are the reference tokens, and the rest are generated tokens; the light blue circles are influenced by the reference, and the white circles are not; the solid arrows represent copying, the dashed arrows represent dependency.

K-th forward pass will be used in the back-propagation:

$$\mathcal{L}_{y,\alpha}(\hat{\boldsymbol{\theta}}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}^{K}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) + \gamma \sum_{t=1}^{T} \mathrm{KL}(\boldsymbol{\alpha}_t||\hat{\boldsymbol{\alpha}}^{K}_t; \hat{\boldsymbol{\theta}}) \tag{4.24}$$

Otherwise the first pass will be used:

$$\mathcal{L}_{y,\alpha}(\hat{\boldsymbol{\theta}}) = -\sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\hat{\boldsymbol{y}}^{1}_{1:t-1}, \boldsymbol{\alpha}_t, \boldsymbol{x}_{1:L}; \hat{\boldsymbol{\theta}}) + \gamma \sum_{t=1}^{T} \mathrm{KL}(\boldsymbol{\alpha}_t||\hat{\boldsymbol{\alpha}}^{1}_t; \hat{\boldsymbol{\theta}}) \tag{4.25}$$

Figure 4.5 illustrates how the iterative parallel generation approximates sequential generation. It can be proved that when $K = T$, $\hat{\boldsymbol{y}}^{K}_{1:T}$ is independent of the reference back-history, and is equivalent to an output sequentially generated [43]. Empirically, K could be much smaller than T, while still addressing the exposure bias [43]. So although parallel attention forcing requires more computation than vanilla attention forcing, it is more efficient thanks to parallel training.

4.3 Application Considerations

4.3.1 Sequence-to-sequence Task

When applying attention forcing to a specific sequence-to-sequence task, it is important to consider the nature of the attention mechanism connecting the input and output, and select the appropriate implementation of attention forcing. For some tasks, the attention is monotonic, and there is only one type of valid attention maps, where the important positions form an approximately diagonal line. In other words, the focus of the decoder either stays on the same input token or moves forward, with the possibility of skipping input tokens. Example tasks include ASR, TTS and voice conversion. For some other tasks, the attention is not necessarily monotonic. More importantly, and there may be multiple valid modes of attention: the ordering of tokens can be changed while the output sequence remains correct. Example tasks include NMT, grammatical error correction and text summarization. If the model takes an ordering that is different from the reference output, the token-level losses will mislead both the output and the attention. To illustrate the problem, consider the following NMT example. For the input *"je suis rentrée chez moi hier"*, the reference output is *"I went home yesterday"*; the alignment $\boldsymbol{\alpha}_t$ is shown in the left of figure 4.6. When using attention forcing, the model is guided by the generated back-history and outputs *"yesterday I went home"*; the alignment $\hat{\boldsymbol{\alpha}}_t$ is shown in the right of figure 4.6. It can be observed that the alignment $\boldsymbol{\alpha}_t$ is not a sensible target for $\hat{\boldsymbol{\alpha}}_t$.

Furthermore, even if the model happens to follow the reference ordering, there might be other issues such as grave mistakes. For tasks where the output is continuous,[4] such as TTS and voice conversion, a small deviation from the reference output is usually not a serious problem. However, this is more serious for tasks where the output is discrete, such as NMT and text summarization. During training, errors in the output history can be so serious that the token-level target is not appropriate, often due to misalignment between the generated output and the reference output. To illustrate the problem, suppose the reference output is *"thank you for listening"*, and the model predicts *"thanks"* at the first time step. In this case, the next output should not be *"you"*, and *"for"* would be a more sensible target.

[4]The meaning of "continuous" comes in two folds. First, natural speech is continuous in time, although it is often sampled as a discrete-time sequence. For a speech sequence, the correlation between tokens is much stronger than that in a text sequence. Second, a speech token follows a continuous distribution. In contrast, a text token follows a discrete distribution, and can only take values in the vocabulary.

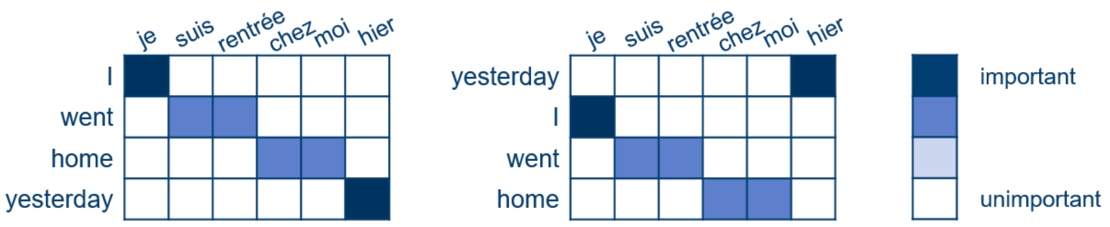

Fig. 4.6 Alignments between: left) the input and the reference output; right) the input and the generated output

Considering the above analysis, for tasks with monotonic attention and continuous outputs, such as TTS and voice conversion, it is recommended to use the default form of attention forcing, introduced in section 4.1. For tasks with non-monotonic attention and discrete outputs such as NMT and text summarization, it is recommended to use scheduled attention forcing, which automatically decides whether attention forcing should be switched on or off. In this thesis, TTS and NMT are used as example tasks to test the effectiveness of the proposed approaches, respectively in chapters 6 and 7.

4.3.2 Down-stream Task

Attention forcing has a feature that is desirable for many down-stream tasks: when the reference alignment is given, the generated output is very likely to be aligned with the reference. Recall that sequence-to-sequence tasks aim to map an input sequence $\boldsymbol{x}_{1:L}$ to an output sequence $\boldsymbol{y}_{1:T}$, where the two sequences are not necessarily aligned or equal in length. The down-stream tasks aim to further process $\boldsymbol{y}_{1:T}$, and produce a final output sequence $\boldsymbol{z}_{1:J}$. The key difference is that models in the down-stream tasks require aligned input and output sequences. For example, in spoken disfluency detection, a sequence-to-sequence model maps speech $\boldsymbol{x}_{1:L}$ to text $\boldsymbol{y}_{1:T}$, and a down-stream disfluency detection model maps $\boldsymbol{y}_{1:T}$ to a label sequence $\boldsymbol{z}_{1:J=T}$. The label sequence and the text sequence are equally long, and each word in the text corresponds to one label. TTS is another example: the sequence-to-sequence model maps text $\boldsymbol{x}_{1:L}$ to acoustic features $\boldsymbol{y}_{1:T}$, and a down-stream vocoder maps $\boldsymbol{y}_{1:T}$ to a waveform sequence $\boldsymbol{z}_{1:J=RT}$, where R is the upsampling rate of the vocoder. Each acoustic feature corresponds to a window of waveform samples, sliding in time at a constant rate.

Let $\boldsymbol{\theta}$ denote the sequence-to-sequence model; let $\boldsymbol{\phi}$ denote the down-stream model; let D_θ and D_ϕ denote the data used respectively to train the sequence-to-sequence model and the down-stream model:

$$\mathrm{D}_\theta = \{\boldsymbol{x}_{1:L}^{(n)}, \boldsymbol{y}_{1:T}^{(n)}\}_1^N \tag{4.26}$$

$$\mathrm{D}_\phi = \{\boldsymbol{y}_{1:T}^{(n)}, \boldsymbol{z}_{1:J}^{(n)}\}_1^N \tag{4.27}$$

where N is the number of training data sequences and $^{(n)}$ the data index. To simplify the notation, the data index is omitted for the length of the sequences, although they also vary with the index. If the sequence-to-sequence model can generate outputs aligned with the references, there will be a half-generated dataset $\hat{\mathrm{D}}_\phi$ for the down-stream model:

$$\hat{\mathrm{D}}_\phi = \{\hat{\boldsymbol{y}}_{1:T}^{(n)}, \boldsymbol{z}_{1:J}^{(n)}\}_1^N \tag{4.28}$$

where $\hat{\boldsymbol{y}}_{1:T}^{(n)}$ denotes a generated output, aligned with $\boldsymbol{y}_{1:T}^{(n)}$. Training with the half-generated dataset $\hat{\mathrm{D}}_\phi$ allows $\boldsymbol{\phi}$ to fix some mistakes made by $\boldsymbol{\theta}$, but this is only possible when $\hat{\boldsymbol{y}}_{1:T}^{(n)}$ is aligned with $\boldsymbol{z}_{1:J}^{(n)}$. To ensure the alignment, the standard approach is to train $\boldsymbol{\theta}$ in teacher forcing mode, and then generate from it in the same mode. This work proposes an alternative approach: to use attention forcing instead of teacher forcing. Training $\boldsymbol{\theta}$ with attention forcing is expected to improve its performance; the intuition has been described in section 4.1, and the effect will be demonstrated by the experiments in section 6.4. Furthermore, in attention forcing mode, each output is predicted based on generated back-history, and is more likely than in teacher forcing mode to contain errors that $\boldsymbol{\theta}$ makes at inference stage.

4.3.3 Over Regularization

Attention forcing has a regularizing effect, in the sense that it encourages the model to behave in a presumably sensible way. During training, the attention mechanism(s) of the attention forcing model is regularized to mimic the teacher forcing model. Hence there is the risk of over regularization, in which case the attention forcing model converges to the teacher forcing model, losing the benefits of attention forcing. For example, when applying attention forcing to Transformer-based models, the default option is to force all the attention maps connecting the encoder and the decoder. However, in a Transformer-based model, there are usually dozens of such attention maps. The exact number is equal to the number of decoder layers times the number of attention heads in each layer. In contrast, in a model based on RNN or CNN, there is only one attention map. As a result, this form of attention forcing tends to over regularize Transformer-based models.

This problem can be addressed by forcing selected attention heads only. For the selected attention heads, the reference attention is given to the following layer, and an alignment loss is computed between the reference and the predicted attention. For the other attention heads, the predicted attention is given to the following layer as usual. For Transformer-based models, different attention heads have different functions [178, 176]. For example, attention heads in the deepest layers of the model capture the most distant relationships [176]. In this thesis, the selection is mainly based on the layer. Figure 4.7 illustrates the idea of forcing selected attention heads. Parallel attention forcing is applied to a Transformer with two decoder layers, but only forcing the attention heads in the second layer. Here the Transformer model is simplified; the detailed model structure is shown in figure 2.8.

4.4 Related Work

Intuitively, attention forcing, as well as scheduled sampling and professor forcing, is between teacher forcing and free running. An advantage of attention forcing is that it does not require a heuristic schedule or a discriminator, which can be difficult to tune. Variations of scheduled sampling have been applied to NMT [204, 43]; while [204] finds it helpful, [43] reports slightly worse performance. Similarly, in TTS, both positive [110] and negative [60] results have been reported, showing that the schedule can be hard to tune. In terms of regularization, attention forcing is similar to professor forcing. For attention forcing, the output layer of the attention mechanism is regularized, and the KL-divergence is a well-established difference metric. [60] and [110] also perform hidden layer regularization, and both regularize the decoder states, for which there is not a natural difference metric. To address this issue, [60] introduces a specific discriminator, and [110] experiments with L_1 loss.

The effect of regularization on attention mechanisms has been studied in previous work [198, 109, 7], where alternative approaches of obtaining reference attention are introduced. [7] and [198] require collecting extra data for reference attention, and [109] uses a statistical machine translation model to estimate them. In contrast, we propose to generate the reference attention with a teacher forcing model, which can be trained simultaneously with the attention forcing model.

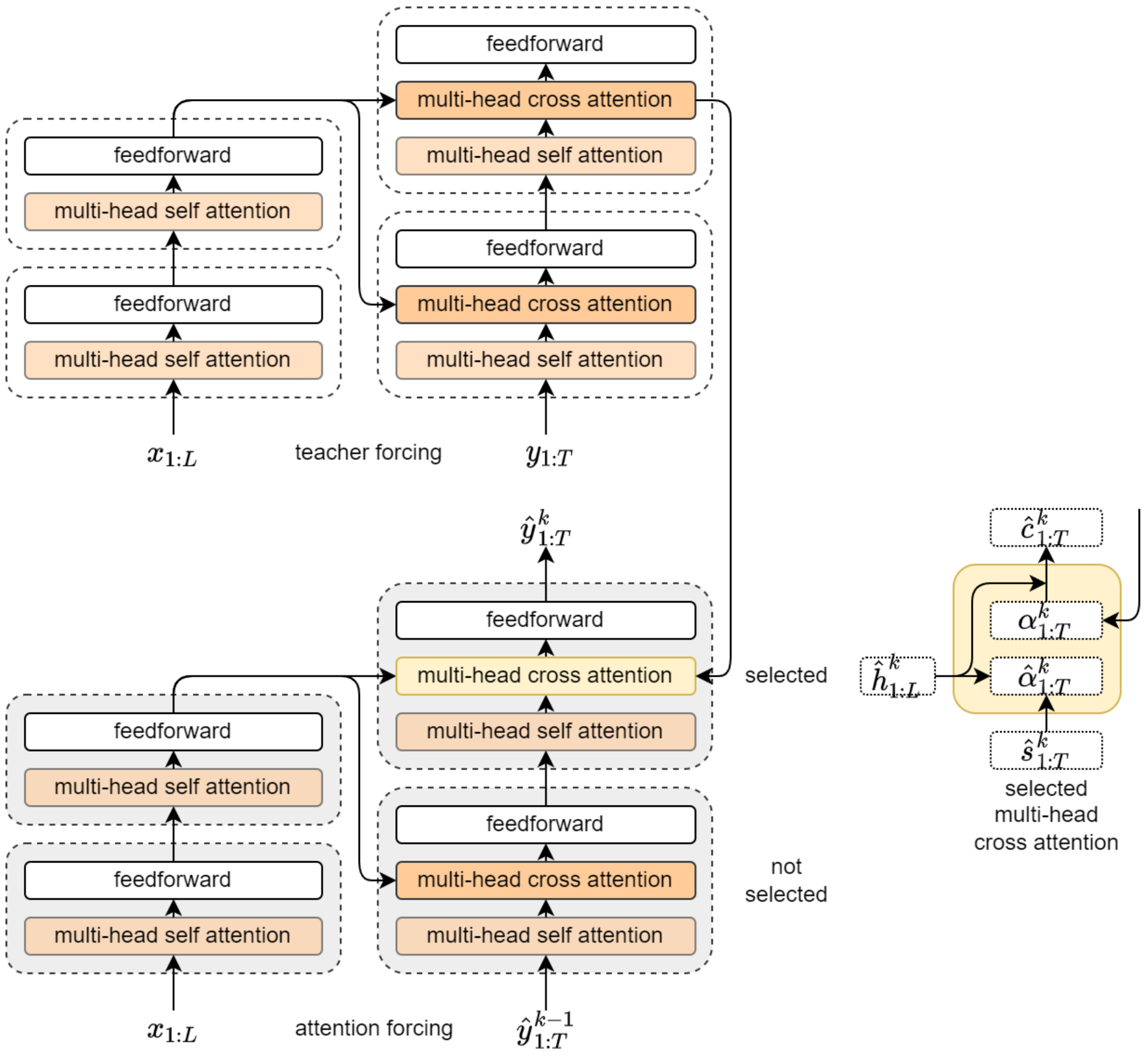

Fig. 4.7 Illustration of parallel attention forcing, applied to a Transformer with two encoder layers and two decoder layers; the attention heads in the second decoder layer are forced.

Compared with sequence-level training approaches, described in section 3.3.3, attention forcing is more efficient, in the sense that it does not require generating multiple output sequences during training. Reference [159] applied Minimum Bayes Risk (MBR) training to NMT, and approximates the expectation of the risk by sampling and renormalizing the probability of the samples. References [148, 5] approximate the same loss with Monte Carlo sampling, and optimizes the loss using Reinforcement Learning (RL). Empirically, this approach requires fewer samples to surpass the baseline performance [148]. A problem for RL is that it struggles when the search space is large [148], and a common remedy is to pretrain the model with teacher forcing. For tasks where the output is discrete, such as NMT and text summarization, the search space is finite given the length of the output. However as the length increases, the search space becomes prohibitively large. For tasks where the output is continuous, such as TTS and voice conversion, the search space is infinite, and is even more challenging.

For sequence-level training, another general concern is the choice of the distance metric, i.e. the risk. As discussed previously, many tasks, including NMT and TTS, do not have a gold-standard objective metric. Empirically, models trained with one metric may not perform equally well when assessed using another metric [148]. To tackle this issue, adversarial training [197, 185] can be used: a discriminator learns a loss function, which is potentially better than standard metrics. The difficulty here is that the discriminator itself can be difficult to train [205]. While attention forcing does not directly optimize a sequence-level loss, it can indirectly reduce the loss by training the model to recover from errors. This is because sequence-level metrics are usually computed by comparing units of the sequences. Examples include word error rate for ASR, and BLEU [129] and ROUGE [107] for NMT. BLEU is essentially a geometric mean over n-gram precision scores as well as a brevity penalty, and ROUGE is recall over bi-grams [148]. If models can go back, from previous errors, to producing the reference output, the sequence-level loss can be reduced.

It can be challenging to apply attention forcing to sequence-to-sequence tasks where the attention is complicated, e.g. multi-modal attention in NMT. In this case, scheduled attention forcing can be used, and there is an extra hyper-parameter λ controlling the tendency to guide the model with the generated output history. The difference from scheduled sampling is that λ is not annealed with a heuristic schedule, and is often easier to tune, which will be demonstrated by the experiments in section 6.4. In addition, whether to use the the generated output history is not random, but determined by the alignment between the reference and generated output.

Although not trivial, the concept of attention forcing can be applied to models without an attention mechanism, where it is essential to find something analogous to attention. For convolutional neural networks, for example, attention maps can be defined based on the activation or gradient [199]. Some recent work on TTS [149, 195, 39, 44] uses a duration model instead of attention. Here duration can be forced in the place of attention [149, 195]. Alternatively, one-hot alignment vectors can be defined according to the duration of input tokens.

4.5 Chapter Summary

This chapter introduced attention forcing. Section 4.1 introduced the general idea: guiding an attention-based sequence-to-sequence model with the generated output history and reference attention. Section 4.2 introduced some variations of attention forcing, namely scheduled attention forcing and parallel attention forcing, for the challenging application scenarios described in section 4.3. The discussion will be recapitulated in the next paragraph. Section 4.4 compared attention forcing with the training approaches described in section 3.3 of the previous chapter.

When the output space is continuous and the attention is monotonic, such as in speech synthesis and voice conversion, it is suggested to use the default form of attention forcing. When the output space is discrete and multi-modal, such as in machine translation and text summarization, scheduled attention forcing should be adopted. Here a selection scheme automatically turns attention forcing on and off depending on the mode of attention, making sure that the training criterion is sensible. When the sequence-to-sequence model is parallelizable, e.g. Transformer, parallel attention forcing can be used. Here the sequential generation is approximated by parallel generation. When using Transformer-based models, a lot of information is available in the attention maps. To avoid over regularization, it is important to limit the information passed from the reference attention.

Section 4.3 also introduced how attention forcing can be combined with down-stream tasks, such as training neural vocoders. In attention forcing mode, the generated outputs are aligned with their references. These generated outputs can be used to train the down-stream models to correct the mistakes made by the sequence-to-sequence model.

Chapter 5

Deliberation Networks

Chapter 3 introduced attention-based sequence-to-sequence models. As described in section 3.3, these models are usually trained with teacher forcing, where the reference output history is used to predict the next token. This makes training efficient, but limits the performance, because during inference the generated output history must be used. The problem is often referred to as exposure bias. Scheduled sampling and professor forcing address this issue by selectively using the generated output history. To make training stable, they leverage a heuristic schedule or an auxiliary classifier, which can be difficult to tune. Sequence-level training is an alternative option, where the model is guided with the generated output history, and a sequence-level criterion is optimized. The downside is that the generation is often sequential, which is inefficient for parallelizable models such as Transformers. Additionally, many tasks, such as speech synthesis and voice conversion, do not have a well-established sequence-level criterion.

Chapter 4 introduced attention forcing, which addresses exposure bias by guiding the model with the generated output history and reference attention. This approach does not require a heuristic schedule or a classifier, and does not require the sequence-to-sequence task to have a well-established sequence-level criterion. As discussed in section 4.4, it is more efficient than most sequence-level training approaches, because it does not require generating multiple output sequences for each input sequence. However, it still requires one sequential generation. Section 4.2.2 introduced parallel attention forcing to approximately generate the output history in parallel across time. This significantly improves efficiency, but potentially loses some benefits of non-parallel attention forcing.

This chapter introduces deliberation networks, an alternative approach that is more suitable for parallelizable models. Here the output sequence is generated in multiple passes, each one conditioned on the initial input and the free-running output of the previous pass. This approach has been successful in sequence-to-sequence tasks where the output is discrete and relatively short, such as NMT [187]. However, it is challenging to apply this approach to tasks where the output is continuous and relatively long, such as TTS. Here the multi-pass model tends to converge to the standard single-pass model, ignoring the previous output. To tackle this issue, a guided attention loss is proposed, encouraging more extensive use of the free-running output.

The novelties of this chapter are as follows. First, section 5.1 describes the framework from a probabilistic perspective, and section 5.2 investigates a range of training approaches. In contrast, previous work [187, 72, 71] takes a deterministic perspective, and describes the one or two training approaches adopted in the experiments. Second, section 5.2 draws the connection between the training of deliberation networks and Minimum Bayes Risk (MBR) training, described in section 3.3.3. Leveraging the connection, the end of section 5.2.1 introduces a novel training approach, which approximates the loss, unlike previous work [187] approximating the gradients. The separate training approach described in section 5.2.2 is not novel, but its synergy with parallel training is pointed out for the first time. Finally, section 5.3 proposes the guided attention loss, facilitating the application of deliberation networks to tasks where the output is continuous and relatively long.

5.1 Framework

Deliberation networks are inspired by a common human behavior: when producing a sequence, be it text or speech, we often revise the initial output to improve its quality. For example, to write a good article, we usually first create a draft and then polish it. To record a section of an audio book, the readers often record several times until the quality is good enough.

A deliberation network consists of multiple models. Its output is generated in multiple passes, each one conditioned on the initial input and the previous free-running output. With the iterative refinement, the final output is expected to be better than the previous ones. For deliberation networks, an essential element is to condition all but the first model on its previous free-running output. This allows the later models to

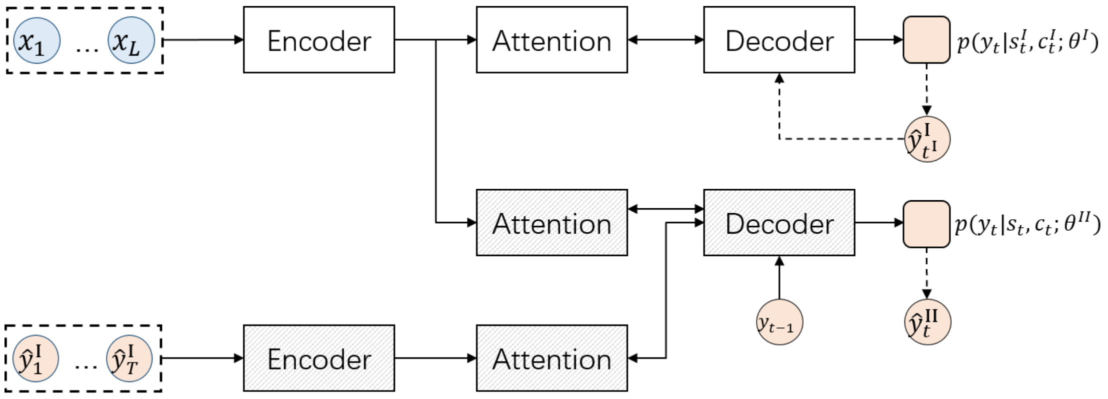
x_1 ... x_L
Encoder
Attention
Decoder
$p(y_t|s_t^I, c_t^I; \theta^I)$
$\hat{y}_{t^{\mathrm{I}}}^{\mathrm{I}}$
Attention
Decoder
$p(y_t|s_t, c_t; \theta^{II})$
y_{t-1}
$\hat{y}_t^{\mathrm{II}}$
$\hat{y}_1^{\mathrm{I}}$ ... $\hat{y}_T^{\mathrm{I}}$
Encoder
Attention

Fig. 5.1 Illustration of a two-pass deliberation network; the clear blocks depict the first pass, operating in free running mode; the shaded blocks depict the second pass, operating in teacher forcing mode.

learn to correct the free-running output, alleviating exposure bias. Without loss of generality, this section describes a two-pass deliberation network, shown in figure 5.1.

In terms of notation, $\boldsymbol{x}_{1:L}$ and $\boldsymbol{y}_{1:T}$ denote the input and reference output; $\boldsymbol{\theta}^{\mathrm{I}}$ and $\boldsymbol{\theta}^{\mathrm{II}}$ denote the first-pass and second-pass models; $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ denotes an intermediate output sequence. The deliberation network models $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L})$ as

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = \sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) \tag{5.1}$$

if $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ is discrete, and

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = \int_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}})d\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}} \tag{5.2}$$

if $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ is continuous. The summation/integration is over $\mathcal{Y}$, the entire space of $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$. $p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})$ is computed by $\boldsymbol{\theta}^{\mathrm{I}}$, a standard single-pass model. $p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}})$ is computed by $\boldsymbol{\theta}^{\mathrm{II}}$, a model with an additional attention mechanism over $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$. $\boldsymbol{\theta}^{\mathrm{I}}$ and $\boldsymbol{\theta}^{\mathrm{II}}$ have different time steps. At time t, assuming $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ is available, $\boldsymbol{\theta}^{\mathrm{II}}$ operates as follows.

$$\boldsymbol{s}_t = f(\boldsymbol{y}_{1:t-1}; \boldsymbol{\theta}_s^{\text{II}}) \tag{5.3}$$

$$\boldsymbol{h}_{x,1:L} = f(\boldsymbol{x}_{1:L}; \boldsymbol{\theta}_{h,x}^{\text{II}}) \tag{5.4}$$

$$\boldsymbol{h}_{y,1:T^{\text{I}}} = f(\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}; \boldsymbol{\theta}_{h,y}^{\text{II}}) \tag{5.5}$$

$$\boldsymbol{\alpha}_{x,t} = f(\boldsymbol{s}_t, \boldsymbol{h}_{x,1:L}; \boldsymbol{\theta}_{\alpha,x}^{\text{II}}) \quad \boldsymbol{c}_{x,t} = \textstyle\sum_{l=1}^{L} \alpha_{x,t,l} \boldsymbol{h}_{x,l} \tag{5.6}$$

$$\boldsymbol{\alpha}_{y,t} = f(\boldsymbol{s}_t, \boldsymbol{h}_{y,1:T^{\text{I}}}; \boldsymbol{\theta}_{\alpha,y}^{\text{II}}) \quad \boldsymbol{c}_{y,t} = \textstyle\sum_{l=1}^{T^{\text{I}}} \alpha_{y,t,l} \boldsymbol{h}_{y,l} \tag{5.7}$$

$$\hat{\boldsymbol{y}}_t^{\text{II}} \sim p(\cdot | \boldsymbol{s}_t, \boldsymbol{c}_{x,t}, \boldsymbol{c}_{y,t}; \boldsymbol{\theta}_y^{\text{II}}) \tag{5.8}$$

$\boldsymbol{\theta}^{\text{II}}$ is built upon $\boldsymbol{\theta}^{\text{I}}$, and has an additional encoder-attention pair. One pair $\{\boldsymbol{\theta}_{h,x}^{\text{II}}, \boldsymbol{\theta}_{\alpha,x}^{\text{II}}\}$ is for the initial input $\boldsymbol{x}_{1:L}$; the additional pair $\{\boldsymbol{\theta}_{h,y}^{\text{II}}, \boldsymbol{\theta}_{\alpha,y}^{\text{II}}\}$ is for the intermediate output $\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}$. The encoder for $\boldsymbol{x}_{1:L}$ shares the same parameters as that of $\boldsymbol{\theta}^{\text{I}}$, i.e. $\boldsymbol{\theta}_{h,x}^{\text{II}} = \boldsymbol{\theta}_h^{\text{I}}$. The probability of $\boldsymbol{y}_t$ depends on $\boldsymbol{s}_t$, $\boldsymbol{c}_{x,t}$ and $\boldsymbol{c}_{y,t}$. $\boldsymbol{s}_t$ is the state vector tracking the output history, and is used by both attention mechanisms. $\boldsymbol{c}_{x,t}$ and $\boldsymbol{c}_{y,t}$ summarize $\boldsymbol{x}_{1:L}$ and $\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}$ respectively. In this work, $\boldsymbol{c}_{x,t}$ and $\boldsymbol{c}_{y,t}$ are concatenated to form a new context vector. The intermediate output of $\boldsymbol{\theta}^{\text{I}}$, such as $\boldsymbol{s}_{1:T^{\text{I}}}^{\text{I}}$ and $\boldsymbol{c}_{1:T^{\text{I}}}^{\text{I}}$, can be combined with the $\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}$ as the input to $\boldsymbol{\theta}_{h,y}^{\text{II}}$. This extra connection speeds up training, but requires more parameters.

During inference, the generated output history replaces the reference, and equation 5.3 becomes $\boldsymbol{s}_t = f(\hat{\boldsymbol{y}}_{1:t-1}^{\text{II}}; \boldsymbol{\theta}_s^{\text{II}})$. The decoding of $\boldsymbol{\theta}^{\text{II}}$ begins when that of $\boldsymbol{\theta}^{\text{I}}$ is complete.

By default, the rest of this chapter assumes that $\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}$ is discrete, and most discussions are agnostic to the continuity of $\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}$. When the continuity does make a difference, the discrete and continuous cases will be discussed separately.

5.2 Training

5.2.1 Joint Training

In theory, $\boldsymbol{\theta}^{\text{I}}$ and $\boldsymbol{\theta}^{\text{II}}$ can be trained by directly maximizing the log of the likelihood in equation 5.1, and the loss function $\check{\mathcal{L}}_y$ is

$$\check{\mathcal{L}}_y(\boldsymbol{\theta}^{\text{I}}, \boldsymbol{\theta}^{\text{II}}) = -\log \sum_{\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}} \in \mathcal{Y}} p(\boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\text{I}}) p(\boldsymbol{y}_{1:T} | \boldsymbol{y}_{1:T^{\text{I}}}^{\text{I}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\text{II}}) \tag{5.9}$$

As discussed in section 3.3, if $\check{\mathcal{L}}_y$ were fully optimized, the KL-divergence between the model distribution $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}})$ and the true distribution $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L})$ would be zero. In general, the actual divergence is limited by the model, data and optimization process. In particular, for deliberation networks, the sum over $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ is intractable due to the prohibitively large space of $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$. A Monte Carlo estimator is often used to approximate either the loss or the gradients. The gradients of $\check{\mathcal{L}}_y$ w.r.t the model parameters $\boldsymbol{\theta}^{\mathrm{I}}$ and $\boldsymbol{\theta}^{\mathrm{II}}$ are

$$\nabla_{\boldsymbol{\theta}^{\mathrm{I}}}\check{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = -\frac{\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}})\nabla_{\boldsymbol{\theta}^{\mathrm{I}}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})}{\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})} \tag{5.10}$$

$$\nabla_{\boldsymbol{\theta}^{\mathrm{II}}}\check{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = -\frac{\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})\nabla_{\boldsymbol{\theta}^{\mathrm{II}}} p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}})}{\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})} \tag{5.11}$$

The sum over the output space appears twice in equations 5.10 and 5.11, which makes the gradient computation more difficult than necessary. A commonly used technique is to instead minimize an upper bound $\mathcal{L}_y$ [187]:

$$\mathcal{L}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = -\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}}) \log p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) \tag{5.12}$$

$$\check{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = -\log \sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}}) p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) \le \mathcal{L}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}})$$

The upper bound is derived with the concavity of the log function and the fact that $\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}}) = 1$. The gradients of $\mathcal{L}_y$ w.r.t. $\boldsymbol{\theta}^{\mathrm{I}}$ and $\boldsymbol{\theta}^{\mathrm{II}}$ are

$$\begin{aligned}\nabla_{\boldsymbol{\theta}^{\mathrm{I}}}\mathcal{L}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = &\\ -\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}}) &\log p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) \nabla_{\boldsymbol{\theta}^{\mathrm{I}}} \log p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}})\end{aligned} \tag{5.13}$$

$$\nabla_{\boldsymbol{\theta}^{\mathrm{II}}}\mathcal{L}_y(\boldsymbol{\theta}^{\mathrm{I}},\boldsymbol{\theta}^{\mathrm{II}}) = -\sum_{\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}\in\mathcal{Y}} p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{I}}) \nabla_{\boldsymbol{\theta}^{\mathrm{II}}} \log p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}},\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\mathrm{II}}) \tag{5.14}$$

Equation 5.13 is derived using the identity $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = f(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log f(\boldsymbol{\theta})$. Compared with the previous gradients shown in equations 5.10 and 5.11, the new gradients, $\nabla_{\boldsymbol{\theta}^{\mathrm{I}}}\mathcal{L}_y$ and $\nabla_{\boldsymbol{\theta}^{\mathrm{II}}}\mathcal{L}_y$, are simpler in the sense that the summation over the output space appears only once.

Comparing equations 5.12 and 3.27, it can be seen that for $\boldsymbol{\theta}^{\mathrm{I}}$, the loss fits into the framework of Minimum Bayes Risk (MBR) training, described in section 3.3.3. Here the risk is defined as $-\log p(\boldsymbol{y}_{1:T}|\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathrm{II}})$. The following subsections will describe two ways of using Monte Carlo approximation, in order to approximate the summation over the output space. They differ in whether to approximate the loss or the gradient, but they can adopt the same sampling process.

Approximating Gradients

Applying Monte Carlo approximation, the gradients $\nabla_{\boldsymbol{\theta}^{\mathrm{I}}}\mathcal{L}_y$ and $\nabla_{\boldsymbol{\theta}^{\mathrm{II}}}\mathcal{L}_y$ are estimated as

$$\nabla_{\boldsymbol{\theta}^{\mathrm{I}}}\mathcal{L}_y(\boldsymbol{\theta}^{\mathrm{I}}, \boldsymbol{\theta}^{\mathrm{II}}) \approx \tag{5.15}$$

$$-\frac{1}{M}\sum_{m=1}^{M} \log p(\boldsymbol{y}_{1:T}|\hat{\boldsymbol{y}}^{\mathrm{I}(m)}_{1:T^{\mathrm{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathrm{II}}) \nabla_{\boldsymbol{\theta}^{\mathrm{I}}} \log p(\hat{\boldsymbol{y}}^{\mathrm{I}(m)}_{1:T^{\mathrm{I}(m)}}|\boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathrm{I}})$$

$$\nabla_{\boldsymbol{\theta}^{\mathrm{II}}}\mathcal{L}_y(\boldsymbol{\theta}^{\mathrm{I}}, \boldsymbol{\theta}^{\mathrm{II}}) \approx -\frac{1}{M}\sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}^{\mathrm{II}}} \log p(\boldsymbol{y}_{1:T}|\hat{\boldsymbol{y}}^{\mathrm{I}(m)}_{1:T^{\mathrm{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathrm{II}}) \tag{5.16}$$

$\{\hat{\boldsymbol{y}}^{\mathrm{I}(1)}_{1:T^{\mathrm{I}(1)}}, ..., \hat{\boldsymbol{y}}^{\mathrm{I}(M)}_{1:T^{\mathrm{I}(M)}}\}$ are M i.i.d. samples drawn from the distribution $p(\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}|\boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathrm{I}})$. The Monte Carlo estimator is unbiased, and its variance is proportional to $\frac{1}{M}$. There is a trade-off between the variance and the computational cost: fewer samples results in higher variance, but lower cost. For SGD-based optimization, noisy estimates of the gradients are used, and this variance adds another level of noise.

The sampling process is often realized by beam search or (a noisy version of) greedy search [142], as described in section 3.2. Once the sampling process is complete, it is straight-forward to to compute $\nabla_{\boldsymbol{\theta}^{\mathrm{I}}}\mathcal{L}_y$ and $\nabla_{\boldsymbol{\theta}^{\mathrm{II}}}\mathcal{L}_y$. For $\boldsymbol{\theta}^{\mathrm{I}}$, the training is equivalent to MBR training. For $\boldsymbol{\theta}^{\mathrm{II}}$, the training can be viewed as teacher forcing, because to compute the conditional probability of the reference output, the reference back-history is used.

Approximating Loss

Alternatively, Monte Carlo approximation can be applied to the loss $\mathcal{L}_y$ shown in equation 5.12, before deriving the gradients. Let $\bar{\mathcal{L}}_y$[1] denote the approximation:

$$\mathcal{L}_y(\boldsymbol{\theta}^{\mathtt{I}}, \boldsymbol{\theta}^{\mathtt{II}}) \approx \bar{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathtt{I}}, \boldsymbol{\theta}^{\mathtt{II}}) = -\frac{1}{M} \sum_{m=1}^{M} \log p(\boldsymbol{y}_{1:T} | \hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{II}}) \tag{5.17}$$

It is simple to compute $\nabla_{\boldsymbol{\theta}^{\mathtt{II}}} \bar{\mathcal{L}}_y$. However, computing $\nabla_{\boldsymbol{\theta}^{\mathtt{I}}} \bar{\mathcal{L}}_y$ is not trivial, because it requires differentiating through $\hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}$, a random sample drawn from a distribution. The sampling process can be formulated as a deterministic function of $\boldsymbol{\theta}^{\mathtt{I}}$, using reparameterization tricks, such as using Gumbel softmax [78]. In general, suppose $\boldsymbol{y}^{\mathtt{I}}_{1:T^{\mathtt{I}}}$ can be viewed as a function of a random variable $\boldsymbol{z}$, and drawing samples from its distribution $p(\boldsymbol{z})$ is practical. Then instead of drawing $\{\hat{\boldsymbol{y}}^{\mathtt{I}(1)}_{1:T^{\mathtt{I}(1)}}, ..., \hat{\boldsymbol{y}}^{\mathtt{I}(M)}_{1:T^{\mathtt{I}(M)}}\}$ from $p(\boldsymbol{y}^{\mathtt{I}}_{1:T^{\mathtt{I}}} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{I}})$, we can draw $\{\hat{\boldsymbol{z}}^{(1)}, ..., \hat{\boldsymbol{z}}^{(M)}\}$ from $p(\boldsymbol{z})$:

$$\begin{gathered} \hat{\boldsymbol{z}}^{(m)} \sim p(\boldsymbol{z}); \quad \hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}} = f(\hat{\boldsymbol{z}}^{(m)}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{I}}) \\ \bar{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathtt{I}}, \boldsymbol{\theta}^{\mathtt{II}}) = -\frac{1}{M} \sum_{m=1}^{M} \log p(\boldsymbol{y}_{1:T} | f(\hat{\boldsymbol{z}}^{(m)}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{I}}), \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{II}}) \end{gathered} \tag{5.18}$$

The gradients for $\boldsymbol{\theta}^{\mathtt{I}}$ and $\boldsymbol{\theta}^{\mathtt{II}}$ are

$$\begin{aligned} &\nabla_{\boldsymbol{\theta}^{\mathtt{I}}} \bar{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathtt{I}}, \boldsymbol{\theta}^{\mathtt{II}}) \approx \qquad (5.19) \\ &- \frac{1}{M} \sum_{m=1}^{M} \nabla_{\hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}} [\log p(\boldsymbol{y}_{1:T} | \hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{II}})] \nabla_{\boldsymbol{\theta}^{\mathtt{I}}} [f(\hat{\boldsymbol{z}}^{(m)}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{I}})] \\ &\nabla_{\boldsymbol{\theta}^{\mathtt{II}}} \bar{\mathcal{L}}_y(\boldsymbol{\theta}^{\mathtt{I}}, \boldsymbol{\theta}^{\mathtt{II}}) \approx -\frac{1}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}^{\mathtt{II}}} \log p(\boldsymbol{y}_{1:T} | \hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{II}}) \qquad (5.20) \end{aligned}$$

Note that $\boldsymbol{y}^{\mathtt{I}}_{1:T^{\mathtt{I}}}$ is independent of $\boldsymbol{\theta}^{\mathtt{I}}$, and the summation in equation 5.9 is over the entire space of $\boldsymbol{y}^{\mathtt{I}}_{1:T^{\mathtt{I}}}$. Hence $\nabla_{\boldsymbol{\theta}^{\mathtt{I}}} p(\boldsymbol{y}_{1:T} | \boldsymbol{y}^{\mathtt{I}}_{1:T^{\mathtt{I}}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{II}}) = 0$, and equations 5.13 and 5.15 hold. In contrast, $\hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}$ depends on $\boldsymbol{\theta}^{\mathtt{I}}$, and the summation in equation 5.17 is over a set depending on $\boldsymbol{\theta}^{\mathtt{I}}$. As described previously, $\hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}$ can be viewed as a deterministic function of $\boldsymbol{\theta}^{\mathtt{I}}$. Hence $\nabla_{\boldsymbol{\theta}^{\mathtt{I}}} p(\boldsymbol{y}_{1:T} | \hat{\boldsymbol{y}}^{\mathtt{I}(m)}_{1:T^{\mathtt{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\mathtt{II}}) \neq 0$, and equation 5.19 holds.

[1]After approximating the loss, a bar is added to the symbols, in order to facilitate comparison with the previous option, where Monte Carlo approximation is applied to the gradients.

For the second-pass model $\boldsymbol{\theta}^{\text{II}}$, given the same group of samples, the gradients remain the same, i.e. $\nabla_{\boldsymbol{\theta}^{\text{II}}}\mathcal{L}_y = \nabla_{\boldsymbol{\theta}^{\text{II}}}\bar{\mathcal{L}}_y$, whether Monte Carlo approximation is applied to the loss or the gradients. For the first-pass model $\boldsymbol{\theta}^{\text{I}}$, unless $M \rightarrow +\infty$, the gradients $\nabla_{\boldsymbol{\theta}^{\text{I}}}\mathcal{L}_y$ and $\nabla_{\boldsymbol{\theta}^{\text{I}}}\bar{\mathcal{L}}_y$ are usually different. It is not trivial to mathematically characterize the difference. However, from a practical point of view, it is simpler to apply Monte Carlo approximation to the gradients, which does not require any reparameterization trick. This is probably why applying Monte Carlo approximation to the gradients is more common in previous research [187].

5.2.2 Separate Training

The joint training scheme has several drawbacks. As there is no loss over the intermediate output $\hat{\boldsymbol{y}}^{\text{I}(m)}_{1:T^{\text{I}(m)}}$, it is likely to deviate from valid target sequences. This makes it difficult to analyze the system. More importantly, if $\boldsymbol{\theta}^{\text{I}}$ is randomly initialized, the intermediate output will be close to random noise, making it difficult for $\boldsymbol{\theta}^{\text{II}}$ to learn to refine the intermediate output. In practice, it is common to pretrain $\boldsymbol{\theta}^{\text{I}}$ with teacher forcing, in order to address these problems [72].

In terms of efficiency, one important problem is that the sampling process is very often auto-regressive, in which case joint training cannot be run in parallel. Recently, Transformer-style models are widely used in various tasks including NMT and TTS. One of their main advantages is parallel training. If teacher forcing is used, there is no recurrent connection in the model, and training can be done in parallel across the length T of the output $\boldsymbol{y}_{1:T}$, because the reference output history $\boldsymbol{y}_{1:t-1}$ is available for any t. However, if sampling is required, these models must operate sequentially, because the generated output history $\hat{\boldsymbol{y}}_{1:t-1}$ must be used, which is not available beforehand.

In this work, we propose to train the two models separately. This is a simple alternative to the joint training scheme previously described. For the proposed approach, $\boldsymbol{\theta}^{\text{I}}$ is trained as a standard sequence-to-sequence model with teacher forcing:

$$\mathcal{L}_y(\boldsymbol{\theta}^{\text{I}}) = -\log p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}^{\text{I}}) \tag{5.21}$$

Then it is fixed to generate samples $\{\hat{\boldsymbol{y}}^{\text{I}(1)}_{1:T^{\text{I}(1)}}, ..., \hat{\boldsymbol{y}}^{\text{I}(M)}_{1:T^{\text{I}(M)}}\}$ for each input $\boldsymbol{x}_{1:L}$, using beam search or (a noisy version of) greedy search. Next, $\boldsymbol{\theta}^{\text{II}}$ is again trained with teacher forcing:

$$\mathcal{L}_y(\boldsymbol{\theta}^{\text{II}}) = -\log \frac{1}{M} \sum_{m=1}^{M} p(\boldsymbol{y}_{1:T} | \hat{\boldsymbol{y}}^{\text{I}(m)}_{1:T^{\text{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\text{II}}) \tag{5.22}$$

The gradients for $\boldsymbol{\theta}^{\text{I}}$ and $\boldsymbol{\theta}^{\text{I}}$ are

$$\nabla_{\boldsymbol{\theta}^{\text{I}}} \mathcal{L}_y(\boldsymbol{\theta}^{\text{I}}) = -\nabla_{\boldsymbol{\theta}^{\text{I}}} \log p(\boldsymbol{y}_{1:T} | \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\text{I}}) \tag{5.23}$$

$$\nabla_{\boldsymbol{\theta}^{\text{II}}} \mathcal{L}_y(\boldsymbol{\theta}^{\text{II}}) = -\frac{1}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}^{\text{II}}} \log p(\boldsymbol{y}_{1:T} | \hat{\boldsymbol{y}}^{\text{I}(m)}_{1:T^{\text{I}(m)}}, \boldsymbol{x}_{1:L}; \boldsymbol{\theta}^{\text{II}}) \tag{5.24}$$

For $\boldsymbol{\theta}^{\text{II}}$, the gradient is the same for separate training and joint training, given the same group of samples. This can be seen by comparing equations 5.16 and 5.24. For $\boldsymbol{\theta}^{\text{I}}$, however, there is a major difference. As described in the previous subsection, joint training uses a noisy Monte Carlo estimator for either the loss or the gradient, and is empirically found to be unstable [187]. In contrast, for separate training, $\boldsymbol{\theta}^{\text{I}}$ is trained with teacher forcing, and is free from the above problem. This can be seen by comparing equation 5.23 to equations 5.15 and 5.19.

In terms of efficiency, the separate training approach has the advantage that it allows parallel training. As $\boldsymbol{\theta}^{\text{I}}$ is fixed, the sample $\hat{\boldsymbol{y}}^{\text{I}}_{1:T^{\text{I}}}$ can be generated and stored beforehand. So that when predicting $\boldsymbol{y}_t$, all the required information is available, including $\boldsymbol{x}_{1:L}$, $\boldsymbol{y}_{1:t-1}$ and $\hat{\boldsymbol{y}}^{\text{I}}_{1:T^{\text{I}}}$. Therefore, the experiments in this thesis only investigate separate training. Following previous research [148, 187], this work adopts greedy search as the sampling scheme, and generates only one sample for each input.

5.3 Application Considerations

5.3.1 Sequence-to-sequence Task

For some sequence-to-sequence tasks, the input and output have the same continuity, and can be embedded in the same way. Examples include NMT and voice conversion. More precisely speaking, the initial input $\boldsymbol{x}_{1:L}$ and the first-pass output $\boldsymbol{y}^{\text{I}}_{1:T^{\text{I}}}$ are either both continuous or both discrete. Hence the additional encoder and attention mechanism for the first-pass output, $\boldsymbol{\theta}^{\text{II}}_{h,y}$ and $\boldsymbol{\theta}^{\text{II}}_{\alpha,y}$, can have exactly the same structure as those for the initial input, $\boldsymbol{\theta}^{\text{II}}_{h,x}$ and $\boldsymbol{\theta}^{\text{II}}_{\alpha,x}$. When the input and output are different in terms of continuity, the additional encoder needs to be modified. For example, for TTS, $\boldsymbol{x}_{1:L}$ is a discrete text sequence, and $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ is a continuous speech sequence.[2] In this work, the text embedding layer in $\boldsymbol{\theta}^{\mathrm{II}}_{h,x}$ is replaced by the linear layer in $\boldsymbol{\theta}^{\mathrm{II}}_{h,y}$.

It is also important to consider the length of the first-pass output $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$, which is often coupled with its continuity. Continuous outputs, such as audio, are usually longer than discrete outputs, such as text. In general, longer sequences are harder for the attention mechanism, and reducing the time resolution alleviates the problem. For example, a pyramid encoder is often used in attention-based ASR models [21]. In this work, when dealing with audio sequences, adjacent frames in $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ are stacked in groups of four, forming a shorter sequence, before being fed into the encoder.

With two sources of information, the second pass-model sometimes converges to the standard single-pass model, ignoring one of the sources. In this case, the attention over the corresponding input sequence does not produce any meaningful alignment. To tackle this issue, the attention can be regularized. This is relatively simple when the attention is expected to be monotonic. For example, when applying deliberation networks to TTS, this work proposes using a guided attention loss [169]:

$$\begin{aligned}\mathcal{L}_{\alpha}(\boldsymbol{\theta}^{\mathrm{II}}) &= \sum_{t=1}^{T}\sum_{l=1}^{T^{\mathrm{I}}}[\alpha_{y,t,l}w_{t,l}] \\ w_{t,l} &= 1 - \exp\left(-(t/T - l/T^{\mathrm{I}})/2g^{2}\right)\end{aligned} \tag{5.25}$$

where g is a hyperparameter controlling the sharpness, and $\alpha_{y,t,l}$ is an element of the attention map $\boldsymbol{\alpha}_{y,1:T}$.[3] In this work, the guided attention loss is only applied to the attention over the first-pass output $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$, as it is a long continuous sequence. This encourages $\boldsymbol{\alpha}_{y,1:T}$ to be diagonal, enabling $\boldsymbol{\theta}^{\mathrm{II}}$ to make more extensive use of $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$. For $\boldsymbol{\theta}^{\mathrm{II}}$, the complete loss is

$$\mathcal{L}_{y,\alpha}(\boldsymbol{\theta}^{\mathrm{II}}) = \mathcal{L}_{y}(\boldsymbol{\theta}^{\mathrm{II}}) + \gamma\mathcal{L}_{\alpha}(\boldsymbol{\theta}^{\mathrm{II}}) \tag{5.26}$$

where γ is a scaling factor. When $\mathcal{L}_{\alpha}$ is used, it is important to monitor $\boldsymbol{\alpha}_{y,1:T}$ and the inference performance on a validation set via objective metrics such as Global Variance (GV). When $\boldsymbol{\alpha}_{y,1:T}$ is sharply diagonal, $\mathcal{L}_{\alpha}$ is low, but the inference performance may degrade. In this work, early stopping is used when the GV drops considerably below the baseline.

[2]In most cases $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$ is a feature sequence, and a neural vocoder maps it to a waveform.

[3]In the subscripts of $\alpha_{y,t,l}$ and $\boldsymbol{\alpha}_{y,1:T}$, $_{y}$ indicates that the attention is over the first-pass output $\boldsymbol{y}^{\mathrm{I}}_{1:T^{\mathrm{I}}}$, and $_{t,l}$ is the position in the map.

5.4 Related Work

In previous research [187, 72], the success of deliberation networks is mainly explained by the improved modeling capacity: the second-pass model can attend to the complete output sequence of the first-pass, in addition to its output history. In this work, it is argued that deliberation networks can address exposure bias, which is essential to their success. It is very important to train the second-pass model to fix the free-running output of the first-pass model, instead of the teacher forcing output. In other words, the secret ingredient of success is to not only use powerful models, but also train them to address exposure bias. This will be demonstrated by the experiments in sections 6.4.5 and 7.3.5.

This work investigates several approaches to train deliberation networks, including ones that are not discussed in previous research. It is pointed out that the training of the first-pass model resembles MBR training, described in section 3.3, when the joint training scheme is adopted. Here the first-pass model is required to generate output sequences sequentially during training, which can be too expensive for Transformer-based models. Therefore, this work proposes the use of the separate training scheme. If efficiency is important, the sequential generation can be done in an offline fashion, so that both the first-pass and second-pass models can be trained in parallel across time.

On a historical note, deliberation networks were first introduced for sequence-to-sequences tasks where both the input and output are text, such as NMT [187]. Their application was later extended to ASR, where the input is audio and the output is text. For these tasks, the additional attention connects two text sequences, which are relatively short and uncorrelated in time. Hence it is easier for the additional attention to learn [24] to align the sequences. However, for tasks such as TTS, where the output is audio, the application is more challenging due to audio sequences being long and strongly correlated in time. As a remedy, this work proposes using the guided attention loss to regularize the attention over the first-pass audio sequence. Some recent works on TTS [150, 149, 195, 39] use a duration model instead of attention. The idea of deliberation is compatible with such models: the extra attention connecting two speech sequences can either be kept or replaced by a duration model.

When it comes to ASR, streaming is an increasingly important demand [72, 71, 118]. Typically, the first-pass model is a streaming model such as RNN-Transducer (RNN-T) [65], and the RNN-T loss [65] for the first-pass model is combined with the likelihood loss for the second-pass model, described in section 5.2. In some cases, MBR training is also applied to the second-pass model, directly optimizing the word error rate [72, 71]. To improve the streaming outputs, the second-pass model often adopts more powerful building blocks, such as Transformer blocks. The most common training scheme is separate training followed by joint training. During joint training, the first-pass model generates samples sequentially, but this is less problematic than the other application cases, thanks to the efficiency of the first-pass model.

5.5 Chapter Summary

This chapter introduced deliberation networks as an alternative approach to addressing exposure bias. Section 5.1 described the general framework: the output sequence is generated in multiple passes, each one conditioned on the initial input and the free-running output of the previous pass. Section 5.2 explored several training approaches, and proposed a separate training scheme, which is more suitable for parallelizable models such as Transformers. Here the multiple passes are trained in turn with teacher forcing. For all passes but the last, the model is fixed, after training, to generate free running outputs, which will be stored for the training of the next pass. Section 5.3 described the application considerations, and proposed a training technique. For tasks such as speech synthesis and voice conversion, the output sequences are relatively long. As a result, the multi-pass model tends to converge to the standard single-pass model, ignoring the previous output. To tackle this issue, a guided attention loss can be added, enabling the system to make more extensive use of the free-running output. Section 5.4 compared deliberation networks with related approaches.

The key novelties of this chapter are as follows. First, section 5.1 described the framework from a probabilistic perspective, instead of the deterministic perspective commonly seen in the literature [187]. Second, section 5.2 investigated a range of training approaches, and drew the connection between the training of deliberation networks and MBR training. While equations 5.9 to 5.16 can be found in the literature, equations 5.17 to 5.20 are newly derived. Furthermore, section 5.2.2 pointed out the synergy between the proposed separate training approach and parallelizable models. Finally, section 5.3 proposed the guided attention loss, facilitating the application of deliberation networks to tasks where the output is continuous and relatively long.

Chapter 6

Text-to-speech Synthesis

This chapter investigates speech synthesis as an example sequence-to-sequence task, in order to validate the effectiveness of attention forcing and deliberation networks. For speech synthesis, the output space is inherently continuous; the attention between the input and output is expected to be monotonic, in the sense that as the output step increases, the input step(s) in focus either increases or stays the same. Similar tasks include voice conversion and hand writing trace generation. To begin with, section 6.1 describes the speech synthesis pipeline, including text analysis, acoustic modeling and vocoder synthesis. Next, section 6.2 describes the acoustic models used in the experiments. In addition, the impact of frame rate is investigated, which helps adapting various training approaches. Section 6.3 takes the same structure to describe neural vocoders, and introduces two novel training techniques: pretraining with unlabeled speech and adaptation using attention forcing, which can generate outputs aligned with the references. Section 6.4 reports and analyzes the experimental results.

6.1 Pipeline

Text-To-Speech (TTS) synthesis is the task of generating speech from text. Over the years there have been many different approaches. The most prominent approaches are concatenative synthesis [73] and parametric synthesis [202, 108]. Concatenative synthesis is based on concatenating units of recorded speech. In contrast, parametric synthesis relies on modeling the probability of speech conditioned on text. Parametric synthesis has various advantages over concatenative synthesis, including robustness and flexibility (to change its voice characteristics) [202]. Conventionally, parametric

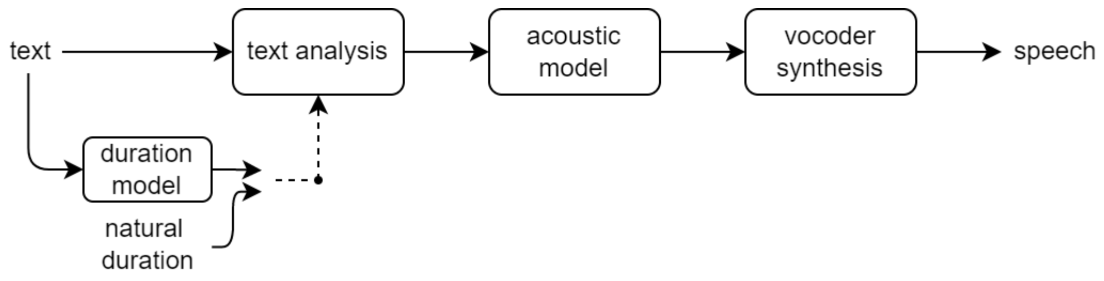

Fig. 6.1 Illustration of a text-to-speech pipeline.

synthesis depends on decision tree-clustered context-dependent Hidden Markov Models (HMMs) [194]. The performance of such systems is not as good as concatenative synthesis, and the main reasons include vocoding, accuracy of acoustic models, and over-smoothing [203]. This work focuses on the next generation of parametric synthesis, which is centered around deep learning. These new parametric systems [202, 174, 182] can address the above issues and have dramatically improved the state-of-the-art performance in TTS.

In general, parametric synthesis involves three stages: text analysis, acoustic modeling and vocoder synthesis. Text analysis maps raw text to a sequence of linguistic features, optionally with the help of a duration model; an acoustic model maps the linguistic feature sequence to an acoustic feature sequence; a vocoder maps the acoustic feature sequence to a waveform. These stages are illustrated by figure 6.1, and will be discussed with more detail in the following subsections.

6.1.1 Text Analysis

Text analysis is the process of mapping raw text to a linguistic feature sequence. In this thesis, raw text refers to normalized text, where the numbers are verbalized; the tokens, i.e. units, of a text sequence are characters. Conventionally, linguistic features are vectors containing contextual and linguistic information such as the identities of the phoneme to be articulated and the neighboring phonemes [202]. The mapping usually leverages linguistic knowledge such as a pronunciation lexicon, and can be rule-based. During training, the natural duration obtained from forced-alignment is fed into the pipeline, to ensure that the linguistic feature sequence is aligned with the acoustic feature sequence. Let f_d and f_l respectively denote the duration and linguistic feature extraction processes, which usually have no parameters to train. Let $\boldsymbol{x}_{1:L}$ denote the text sequence, $\boldsymbol{d}_{1:L}$ the duration of each token, $\widetilde{\boldsymbol{x}}_{1:T}$ the linguistic feature sequence, and $\boldsymbol{y}_{1:T}$ the acoustic feature sequence. Text analysis can be formulated as

$$\boldsymbol{d}_{1:L} = f_d(\boldsymbol{x}_{1:L}, \boldsymbol{y}_{1:T}) \tag{6.1}$$

$$\widetilde{\boldsymbol{x}}_{1:T} = f_l(\boldsymbol{x}_{1:L}, \boldsymbol{d}_{1:L}) \tag{6.2}$$

The natural duration is also used as the reference to train a duration model $\boldsymbol{\psi}$. At the inference stage, the duration model predicts the duration $\hat{\boldsymbol{d}}_{1:L}$ of each token:

$$\hat{\boldsymbol{d}}_{1:L} = f(\boldsymbol{x}_{1:L}; \boldsymbol{\psi}) \tag{6.3}$$

$$\widetilde{\boldsymbol{x}}_{1:\hat{T}} = f_l(\boldsymbol{x}_{1:L}, \hat{\boldsymbol{d}}_{1:L}) \tag{6.4}$$

For conventional text analysis, a linguistic feature sequence can be viewed as a knowledge-based representation of text, which is context-aware but has some loss of information. For recent TTS systems based on deep learning, text analysis can be merged with acoustic modeling. Here the acoustic models can directly map a character sequence to an acoustic feature sequence [182], or even a waveform [184]. The context-aware representation of text is implicitly learned, which simplifies the TTS pipeline [182]. With enough data and modeling capacity, the learned representation can be more effective than conventional knowledge-based representation [35]. In particular, using self-supervised training techniques, powerful language models can be trained with massive amounts of unlabeled text data [35, 98], and then integrated with acoustic models as feature extractors [61].

With the development of multi-lingual TTS, phone-based systems [191, 23] have gained increasing attention. Here the text analysis stage is reduced to mapping character sequences to phone sequences. This is often done using external Grapheme-To-Phoneme (G2P) [167] systems. The accuracy of G2P systems limits the final TTS performance, but also reduces the need for powerful acoustic models. In particular, for multi-lingual TTS, phone-based systems often make more efficient use of data. Languages that do not have the same alphabet often have many common phones, which can facilitate the learning of multiple languages. This work focuses on mono-lingual TTS. In the experiments, unless otherwise stated, character sequences are directly fed into acoustic models.

6.1.2 Acoustic Modeling

Strictly speaking, an acoustic model takes as input a linguistic feature sequence, and outputs an acoustic feature sequence. This definition can be extended: the input can be any form of text representation, and the output can be any form of waveform representation. From a probabilistic perspective, the model estimates the distribution of the output sequence conditioned on the input sequence. As previously stated, this work focuses on acoustic models based on deep learning. For the early generation of such models, the input and output sequences are aligned and have the same length. Let $\boldsymbol{\theta}$ denote an acoustic model; let $\widetilde{\boldsymbol{x}}_{1:T}$, $\boldsymbol{h}_{1:T}$ and $\boldsymbol{y}_{1:T}$ denote the input, hidden and output sequences, which have the same length T. The acoustic modeling process can be formulated as

$$p(\boldsymbol{y}_{1:T}|\widetilde{\boldsymbol{x}}_{1:T};\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1},\widetilde{\boldsymbol{x}}_{1:T};\boldsymbol{\theta}) \approx \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{h}_t;\boldsymbol{\theta}) \tag{6.5}$$

The acoustic model $\boldsymbol{\theta}$ can be built with various building blocks described in chapter 2. For example, for models based on bidirectional RNNs, which have been successful in TTS [47], $\boldsymbol{h}_{1:T}$ can be computed as shown in equations 2.9 to 2.11.

This work uses sequence-to-sequence acoustic models that adopt the encoder-attention-decoder architecture. The term "sequence-to-sequence" is used to exclusively refer to the models whose output and input sequences are not aligned or equally long. For these models, equation 6.5 is replaced by

$$p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta}) = \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{y}_{1:t-1},\boldsymbol{x}_{1:L};\boldsymbol{\theta}) \approx \prod_{t=1}^{T} p(\boldsymbol{y}_t|\boldsymbol{s}_t,\boldsymbol{c}_t;\boldsymbol{\theta}) \tag{6.6}$$

This is a compact equivalence of equations 3.1 and 3.7. $\boldsymbol{s}_t$ and $\boldsymbol{c}_t$ are vectors respectively summarizing the output history and the input. The output is not aligned with the input, which is of length L. Attention mechanisms are used to handle the alignment. There are alternative forms of sequence-to-sequence acoustic models. For example, some recent works on TTS [150, 149, 195, 39] use duration models to replace the attention mechanism. Here each input token is upsampled based on its duration. As a result, monotonic alignment is guaranteed, but the application is limited to tasks such as TTS, where the alignment is expected to be monotonic. In contrast, attention-based models are more flexible in the sense that they can be applied to other tasks such as ASR and NMT. The downside is that sometimes task-specific training techniques are essential, e.g. a guided attention loss [169] encouraging monotonic attention. Generally speaking, there is a design choice of whether to express prior knowledge in the model or in the training algorithm. When applying a flexible model to a specific task, prior knowledge can be leveraged during training.

Details on attention mechanisms are available in chapter 2, and details on the encoder-attention-decoder architecture are available in chapter 3. Chapters 4 and 5 introduced novel training approaches, and compared them with more standard approaches described in section 3.3. Section 6.2 will elaborate on some typical models, which are used in the experiments.

6.1.3 Vocoder Synthesis

Conventional vocoders, e.g. STRAIGHT [84] and PML [32], have two parts. The vocoders can map audio to acoustic features, which are used as the reference for training acoustic models. This process is referred to as vocoder analysis. Acoustic features are also referred to as vocoder features. Each acoustic feature vector includes information such as the mel-spectrum, band aperiodicity, and fundamental frequency. At the inference stage, the vocoders map acoustic features, generated by the acoustic models, to audio. This process is referred to as vocoder synthesis. For conventional vocoders, both analysis and synthesis are based on signal processing techniques, and do not require training. During vocoder analysis, there is some loss of information, which cannot be fully recovered by conventional vocoder synthesis. This is a major limit for the final TTS performance [174].

Neural vocoders leverage the capacity of neural networks to address the above issue. The neural vocoders are trained to map acoustic features to audio, and they replace conventional vocoders at the synthesis stage. From a probabilistic perspective, they model the distribution of a waveform sequence conditioned on an acoustic feature sequence. Typically, this distribution is factorized across time:

$$p(z_{1:J}|\boldsymbol{y}_{1:T};\boldsymbol{\phi}) = \prod_{j=1}^{J} p(z_j|z_{1:j-1},\boldsymbol{y}_{1:T};\boldsymbol{\phi}) \tag{6.7}$$

where $z_{1:J}$ denotes a waveform sequence, and $\boldsymbol{\phi}$ denotes a neural vocoder. Equation 6.7 looks similar to equation 6.6, which formulates acoustic modeling. However, there are fundamental differences. First, for neural vocoders, the input and output sequences are aligned, so each input token is upsampled and corresponds to a fixed number of output tokens. In this regard, neural vocoders can be viewed as special sequence-to-sequence models, where the output and input sequences are aligned. Second, the waveform is usually orders of magnitude longer than the corresponding text or acoustic feature sequence. It can be very challenging to model such a long sequence: for standard RNNs and CNNs, the receptive field is not enough [174].

Various neural vocoders have been proposed to address this challenge. They differ in the estimation of $p(z_j|z_{1:j-1}, \boldsymbol{y}_{1:T}; \boldsymbol{\phi})$, and keep a balance between audio quality and efficiency. Section 6.3 will review a range of neural vocoders, and then elaborate on some typical neural vocoders, which will be used in the experiments.

6.2 Acoustic Models

This section reviews a range of acoustic models, and then elaborates on the models that will be used in the experiments. Inference and training tricks will be discussed at the end of the section.

6.2.1 General Review

Recall that this work focuses on models that adopt the encoder-attention-decoder architecture. Section 3.1.3 generally discussed the pros and cons of various building blocks, which are described in chapter 2. Briefly speaking, RNNs are powerful [161] but can be difficult to train [12]. In particular, due to the recurrent connections, training cannot be run in parallel across time. CNNs have a finite receptive field and are adept at capturing local correlations. When teacher forcing is used, there are no recurrent connections and training can be run in parallel across time. Transformer blocks are also suitable for parallel training. Compared with CNNs, they have longer receptive fields, and are better at modeling global correlations.

Tacotron [182] was the first sequence-to-sequence model achieving state-of-the-art performance in TTS. This model is based on RNNs. Its output is linear-spectrogram, which is converted to a waveform using the Griffin-Lim algorithm. Tacotron 2 [158] improves Tacotron by removing excessive model parameters. It uses mel-spectrograms as the output, and leverages the modeling power of a neural vocoder to generate waveforms. Several works improve Tacotron from different aspects. For example, Ref-Tacotron [163] and GST-Tacotron [183] extend the architecture with prosody and more general style embedding. Wave-Tacotron [184] integrates the neural vocoder, and directly generates waveforms. Parallel Tacotron [45, 44] is the non-autoregressive version of Tacotron, which significantly improves the inference speed. DurIAN [195] replaces the attention mechanism with a duration model, which is a TTS-specific approach.

The DeepVoice series [2, 50, 138] are based on CNNs. Similar to the Tacotron series, each generation improves the modeling power, and makes the model more suitable for multi-speaker TTS. The training and the model itself are also simplified over time [170]. DeepVoice 2 [50] extends DeepVoice [2] with speaker embedding. DeepVoice 3 [138] is a more compact model, and can scale up to real-word multi-speaker datasets. Several works follow this CNN line of research. For example, ClariNet [137] generates waveforms from text in a fully end-to-end fashion. ParaNet [135] is a non-autoregressive model, which speeds up inference while maintaining good speech quality.

Transformer TTS [104] was the first application of Transformers to TTS, which generates mel-spectrograms from phonemes. This model absorbs some designs from Tacotron 2 such as pre-net, post-net and stop token prediction. It achieves similar voice quality as Tacotron 2 but has faster training speed thanks to parallel training. In TTS, the attention connecting the input and output should be monotonic [169]. For RNN-based models, this can be realized with location-sensitive attention, which requires recurrent connections. For Transformer TTS, however, this conflicts with parallel training. As a result, the cross attention mechanisms in Transformer TTS are less robust. Several follow-up works aim to address this issue. For example, MultiSpeech [23] uses techniques such as encoder normalization, decoder bottleneck, and diagonal attention constraint.

6.2.2 Tacotron

Tacotron [182] is sequence-to-sequence model that takes a character sequence as input and outputs the corresponding spectrogram. It adopts the encoder-attention-decoder architecture described in section 3.1. Figure 6.2 depicts the model, which includes an encoder, an attention mechanism, a decoder, and a post-processing network.

CBHG Module Tacotron extensively uses a module called CBHG, which is a powerful module for extracting representations from sequences [182]. It consists of several basic building blocks described in chapter 2: a bank of 1D convolutional filters, followed by highway networks [166] and a bidirectional GRU [30]. The input sequence

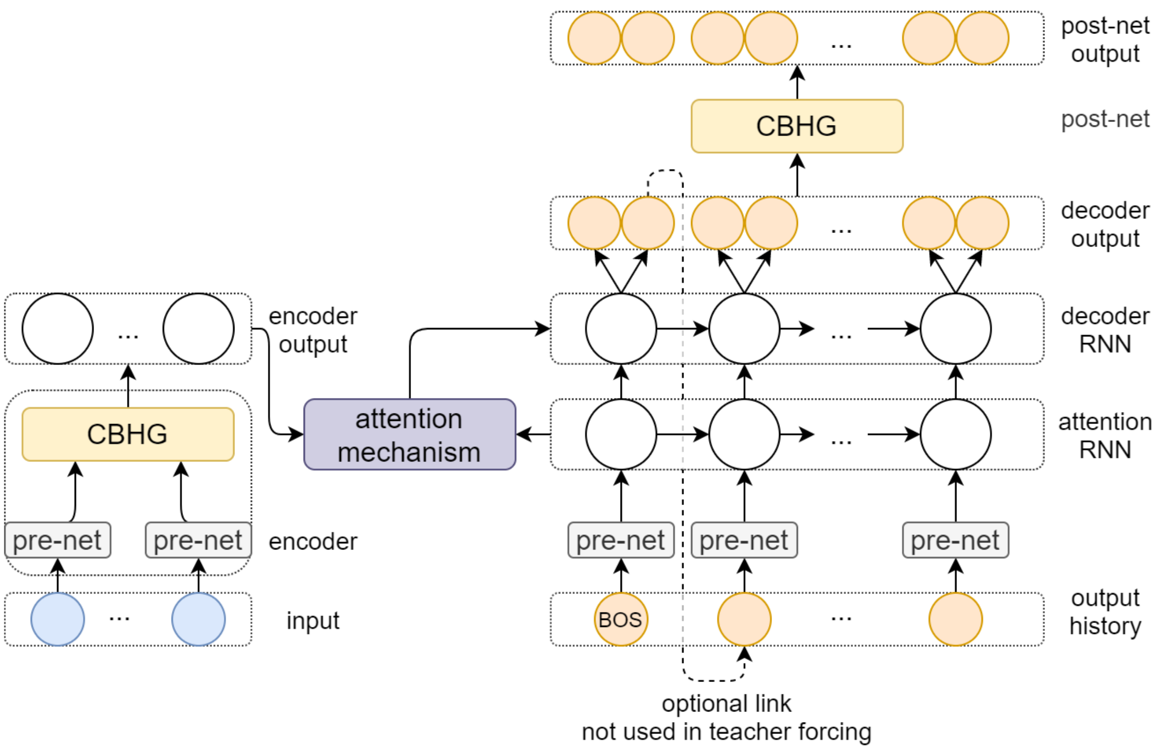

Fig. 6.2 Illustration of Tacotron [182].

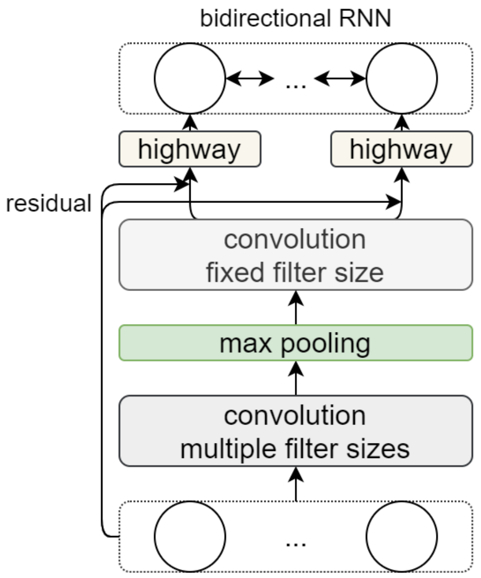

Fig. 6.3 Illustration of a CBHG module [182].

is first convolved with G sets of 1D convolutional filters, where the g-th set contains C_g filters of width g. These filters explicitly model dependencies at different scales, which is similar to modeling unigrams, bigrams, up to G-grams. The outputs are stacked together and further max pooled along time to increase local invariances. The max pooling is followed by a few fixed-width 1D convolutions, whose outputs are added with the original input sequence via residual connections [64]. For all convolutional layers, batch normalization [76] is used, and the stride is one, in order to preserve the original time resolution. The convolution outputs are fed into a multi-layer highway network to extract high-level features. Finally, a bidirectional GRU extracts sequential features, considering both forward and backward context.

Encoder The goal of the encoder is to extract representations of text sequences. The input to the encoder is a character sequence, where each character is represented as a one-hot vector and embedded into a continuous vector. A bottleneck layer with dropout, collectively called a “pre-net”, is applied to each embedding. This helps convergence and improves generalization [182]. A CBHG module transforms the pre-net outputs into the final encodings.

Attention The attention mechanism connecting the encoder and the decoder is additive attention [6]. As described in section 2.2.1, this is a content-based attention mechanism. An attention RNN, which is part of the decoder, produces the attention query at each decoder time step.

Decoder The decoder consists of a pre-net, an attention RNN and an decoding RNN. The first decoder step is conditioned on an all-zero frame (a token), which represents the beginning of decoding. After that, there is a train-inference mismatch, which is referred to as exposure bias and is described in section 3.3. During training, the decoder operates in teacher forcing mode, where every step is conditioned on the last reference frame. During inference, the decoder operates in free running mode, and the last generated frame is used instead. The conditioning frames are processed by a pre-net, as is done in the encoder. The pre-net outputs are fed into the attention RNN, which produces state vectors. At each decoding step, the attention mechanism takes a state vector, and summarizes the encoder outputs with a context vector. The context vector and the state vector are concatenated and fed into the decoding RNN. The RNNs are unidirectional GRUs with residual connections [186]. The target of the decoder is a 80-band mel-spectrogram. At each time step, a fully-connected output layer maps the output of the decoding RNN to multiple frames, instead of just one. This reduces the number of decoder time steps, and makes it easier for the attention mechanism to learn meaningful alignments [182]. As section 6.2.4 will explain, this also addresses exposure bias.

Post-processing Network Finally, a CBHG module acts as a post-processing network, and maps the generated mel-spectrograms to linear-spectrograms, which can be converted to waveforms using the Griffin-Lim algorithm [56]. The post-processing network sees the entire mel-spectrogram sequence, which mitigates the limitations of the unidirectional decoding process. The Griffin-Lim algorithm was originally used for its simplicity [182], and will be replaced by vocoders in this work.

6.2.3 Tacotron 2

Tacotron 2 [158] can be viewed as the compact version of Tacotron. It keeps the encoder-attention-decoder architecture, but removes some excessive model parameters. Tacotron 2 uses standard LSTM and convolutional layers in the encoder and decoder instead of CBHG modules and GRU layers. At each decoding step, a single spectrogram frame is predicted, instead of many. The convolutional layers in Tacotron 2 are regularized using dropout [165], and LSTM layers are regularized using zoneout [93]. In order to introduce output variation at inference time, dropout is also applied to the pre-net of the decoder.

Encoder The encoder converts a character sequence into a hidden sequence. The character embeddings are passed through a stack of three convolutional layers, followed by batch normalization [76] and ReLU activations. The output of the final convolutional layer is passed into a single bidirectional LSTM layer to generate the hidden sequence.

Attention The attention mechanism is location-sensitive attention [28], described in section 2.2.1. It extends additive attention [6], which is used in Tacotron, by considering attention weights from previous decoder time steps. This encourages the model to move forward consistently through the input, leveraging the monotonic nature of the attention in TTS [158]. Attention weights are computed after projecting inputs and location features to hidden representations. Location features are computed using 1D convolution filters. Note that the use of location-sensitive attention is an example of expressing prior knowledge in the model, a design choice discussed in section 6.1.2.

Decoder The decoder consists of two pre-net layers, two LSTM layers, and a linear projection layer. The prediction from the previous time step is first passed through the pre-net, which contains two fully connected layers with ReLU units. The pre-net output and attention context vector are concatenated and passed through a stack of two unidirectional LSTM layers. The concatenation of the LSTM output and the attention context vector is then projected through a linear transform to produce a prediction of the target spectrogram frame.

In parallel to spectrogram frame prediction, the concatenation of decoder LSTM output and the attention context is projected down to a scalar and passed through a sigmoid activation to predict the probability that the output sequence has completed. This stop token prediction is used during inference to allow the model to dynamically determine when to terminate generation instead of always generating for a fixed duration [158].

Post-processing Network The post-processing network is a CNN, which predicts a residual to add to the initial prediction to improve the overall reconstruction. Each post-net layer uses batch normalization, followed by tanh activation for all but the final layer.

6.2.4 Inference and Training

Section 3.2 generally described the inference process for sequence-to-sequence models. For acoustic models, the output is continuous, hence greedy search is used during inference. The models stop when the output meets a certain condition, e.g. when the L_1 distance between the output and the padding token are small enough. Tacotron adopts this approach, and does not mask the zero-paddings during training. Alternatively, the model can make an additional binary decision about whether to stop, as is done in Tacotron 2. Here a cross-entropy loss is combined with the loss over the acoustic features.

Section 3.3 generally described a range of training approaches for sequence-to-sequence models. To the author's knowledge, teacher forcing is the standard training approach for TTS. The loss is shown in equation 3.16. As described in section 3.1.4, this loss is equivalent to the average distance between the generated tokens and their references. Tacotron assumes a Laplace distribution, and uses a L_1 distance; Tacotron 2 assumes a Gaussian distribution, and uses a L_2 distance. In these two cases, the distance is measured using the outputs of both the decoder and the post-processing network, and combined with equal weights.

To address exposure bias, recent research has investigated scheduled sampling and variations of professor forcing. References [60] and [110] both regularize the decoder states. As there is no standard distance metric, reference [60] designs a discriminator and uses the hinge version of adversarial loss, and reference [110] uses a L_1 distance. Reference [110] finds scheduled sampling beneficial, while reference [60] finds the opposite, showing that the schedule can be hard to tune. For TTS, there is not a well-established objective metric, and sequence-level training is not often adopted for reasons described in section 3.3.

Frame Rate For acoustic models, the output is a continuous sequence, and the frames are highly correlated in time. This makes exposure bias more severe: models trained with teacher forcing tend to copy from the output history. In this case, it is essential to reduce the frame rate. Recall that acoustic features are extracted from waveforms, using a sliding window. The sampling rate of the waveform and the hopping length of the window determines the frame rate of the acoustic feature sequence. The sampling rate equals the number of waveform samples in one second of speech, usually above ten thousand [174][1]; the frame rate equals the number of feature vectors in the corresponding acoustic feature sequence, usually hundreds [182]. In conventional parametric TTS, the standard frame rate is 200 Hz [201, 200]. For more recent TTS models, which have more memorizing capacity, this frame rate is reduced to 80 Hz [182, 158].

Chapter 4 and chapter 5 respectively introduced attention forcing and deliberation networks. When these approaches are applied to TTS, reducing the frame rate is no longer essential. For attention forcing, the reference output is not fed into the model; for deliberation networks, the second-pass models are trained to fix the errors in the free-running output from the first-pass models. As described in section 5.3, the length of the output sequence makes it difficult for attention mechanisms to find the right focus. Hence when applying deliberation networks to TTS, it is important to regularize

[1]For good quality speech data is often sampled at 16 KHz or 22.05 KHz [174, 77].

the attention over the acoustic feature sequence, leveraging the monotonic nature of the task.

6.3 Neural Vocoders

This section reviews a range of neural vocoders, describes some distributions for waveform modeling, and then discusses the neural vocoders that will be used in the experiments, namely SampleRNN and WaveRNN. Inference and training techniques will be discussed at the end of the section.

6.3.1 General Review

WaveNet [174] was the first neural vocoder to achieve state-of-the-art performance. It leverages dilated CNNs to model long-range dependency. SampleRNN [120] is an alternative introduced at about the same time. It leverages hierarchical RNNs to reach similar performance as WaveNet. Initially, WaveNet and SampleRNN are not investigated as neural vocoders. WaveNet is originally conditional on conventional linguistic features, and then adapted to linear-spectrograms [182] and mel-spectrograms [158]. SampleRNN was developed for unconditional waveform generation, and later works [164, 41][2] demonstrated its neural vocoding capacity.

A common problem for the first generation of neural vocoders is inference speed. The models have a large number of parameters, many of which can be pruned. More importantly, the autoregressive generation of audio is very time consuming due to the length the output sequence. A lot of works have investigated lightweight and fast vocoders. WaveRNN [81] is a prominent example. This model is still autoregressive, but is built on a standard RNN. To increase the inference speed, several techniques are adopted, including a dual softmax layer, weight pruning, and subscale sampling. Some later works [115, 133, 79] further improved the robustness.

LPCNet [173] is another efficient neural vocoder. It incorporates prior knowledge about vocal tract into neural networks, and uses linear prediction coefficients to compute the next waveform point while leveraging a lightweight RNN to compute

[2]Reference [41] was written by the author of this thesis during the PhD course. It compared two ways of using SampleRNN (as a neural vocoder or a combination of acoustic model and neural vocoder), and proposed training techniques such as tier-wise training.

the residual. LPCNet is initially conditioned on Bark-Frequency Cepstral Coefficients (BFCC), but can be easily adapted for mel-spectrograms [170]. Some later works further improved LPCNet from different perspectives, such as reducing complexity for speedup [177, 140, 83], and improving stability for better quality [75].

Normalizing flow [36, 37] is a kind of generative model. It transforms a probability density with a sequence of invertible mappings [151] to another probability density. During training, a standard/normalized probability distribution (e.g., Gaussian) is obtained through the sequence of invertible mappings based on the change-of-variables rules. At the inference stage, it generates data from a standard probability distribution through the inverse of these transforms. Flow-based neural vocoders can be divided into two categories, according to the the type of mapping [128]: autoregressive transforms [88] (e.g., inverse autoregressive flow used in Parallel WaveNet [126]), and bipartite transforms (e.g., Glow [87] used in WaveGlow [143], and RealNVP [37] used in FloWaveNet [85]). Both autoregressive and bipartite transforms have their pros and cons. Autoregressive transforms are more expressive than bipartite transforms by modeling dependency between data distribution and standard probability distribution, but require more complicated training such as teacher-student training. Bipartite transforms enjoy a much simpler training pipeline, but usually require a larger number of parameters to reach comparable performance. WaveFlow [139] introduces a unified view of these models to explicitly trade inference parallelism for model capacity [170].

Generative Adversarial Networks (GANs) [52] have been widely used in generative tasks, such as image generation [52], text generation [197], and audio generation [38]. GANs consist of a pair of neural networks: a generator for data generation, and a discriminator to tell generated data from true data. Here the generator is equivalent to the neural vocoders described above. The discriminator effectively learns a loss function, and is not used during inference. Popular GAN-based neural vocoders include WaveGAN [38], GAN-TTS [13], MelGAN [96], Parallel WaveGAN [190], and HiFi-GAN [91]. For the generators, modeling long-range dependency is still a key challenge. Most models use dilated convolution to increase the receptive field, and transposed convolution to upsample the conditioning vectors (e.g. mel-spectrograms) to match the length of waveform [170]. As for the discriminators, the key question is how to capture the characteristics of waveform. Typically, multiple discriminators are adopted, each one focusing on the dependency of a particular time scale. Other regular GAN losses such as the hinge-loss [106], waveform-specific losses such as STFT loss [3] and feature matching loss [99] have been introduced to train GAN-based neural vocoders. These additional losses can improve the stability and efficiency of adversarial training [190], and improve the perceptual audio quality [170].

6.3.2 Waveform Distributions

Categorical Distributions

To model a waveform sample, it is most common to use a mixture model or categorical distribution [126, 174]. In general, categorical distributions are more flexible than mixture models. Although a mixture model could approximate any distribution given an infinite number of components, in practice the number of components is limited by data and training. Waveform samples are typically quantized as 8-bit integers, and μ-law is used to reduce quantization errors. This quantization approach is a trade-off between accuracy and model size, and is empirically validated in [174]. Let z denote such an integer, following a categorical distribution represented by $\boldsymbol{q}$, the probability of z can be written as

$$p(z|\boldsymbol{q}) = \boldsymbol{q}[z] \tag{6.8}$$

The downside of using a standard categorical distribution is that, to increase the encoding resolution, the number of model parameters must increase accordingly. For example, to increase the encoding resolution from 8-bit to 16-bit, the number of the parameters in the output layer must increase by 255 times. To deal with this issue, dual categorical distribution can be applied. As the name indicates, it uses two softmax output layers to model an individual waveform sample. The first layer represents the coarse bits, and conditions the second layer, which represents the fine bits. For example, achieving 16-bit resolution requires two output layers each having 256 nodes, instead of a single output layer with 65536 nodes. This can be formulated as

$$p(z|\boldsymbol{q}^c, \boldsymbol{q}^f) = p(z^c|\boldsymbol{q}^c)p(z^f|\boldsymbol{q}^f, z^c) = \boldsymbol{q}^c[z^c]\boldsymbol{q}^f[z^f] \tag{6.9}$$

where z^c and z^f denote the coarse and fine parts of a 16-bit integer; $\boldsymbol{q}^c$ and $\boldsymbol{q}^f$ denote the coarse and fine parts of its distribution. This idea can easily be extended to using more than two softmax output layers.

Discretized Mixture of Logistics

Discretized Mixture of Logistics (MoL) is a typical mixture model approach for waveform representation. Empirically, it achieves comparable performance to categorical distributions [126]. A single logistic distribution $\mathbb{L}$ can be defined as:

$$\mathbb{L}(u; \mu, \lambda) = \frac{\exp((u - \mu)/\lambda)}{\lambda(1 + \exp((u - \mu)/\lambda))^2} \tag{6.10}$$

u is a continuous random variable, μ is the mean of the distribution, and $\lambda > 0$ is the scale parameter. Its Cumulative Distribution Function (CDF) is a sigmoid function:

$$\begin{aligned} &\int_{-\infty}^{a} \mathbb{L}(u; \mu, \lambda) du = \sigma(\frac{a - \mu}{\lambda}) \\ &\sigma(u) = \frac{1}{1 + \exp(-u)}; \frac{d\sigma(u)}{du} = \sigma(u)(1 - \sigma(u)) \end{aligned} \tag{6.11}$$

A mixture of logistics can be defined as:

$$p(u|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \sum_{m=1}^{M} \pi_m \mathbb{L}(u|\mu_m, \lambda_m) \tag{6.12}$$

$\boldsymbol{\pi}$ denotes the weights of the M mixture components. In this approach, it is assumed that there is a latent waveform sample u with a continuous distribution, which is then rounded to its nearest discrete representation to give the observed waveform sample z [126]. Taking 8-bit encoding as an example, the probability of the observed discretized value z can be computed as:

$$p(z|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \sum_{m=1}^{M} \pi_m [\sigma(\frac{z + 0.5 - \mu_m}{\lambda_m}) - \sigma(\frac{z - 0.5 - \mu_m}{\lambda_m})] \tag{6.13}$$

For the edge case of 0, $z - 0.5$ is replaced by $-\infty$; for the edge case of 255, $z + 0.5$ is replaced by ∞. The advantage of this approach is that it is computationally efficient [155]. In addition, it penalizes classification errors differently depending on the distance between the reference bin and the predicted bin. Previous experiments have shown that only a relatively small number of mixture components is needed [155].

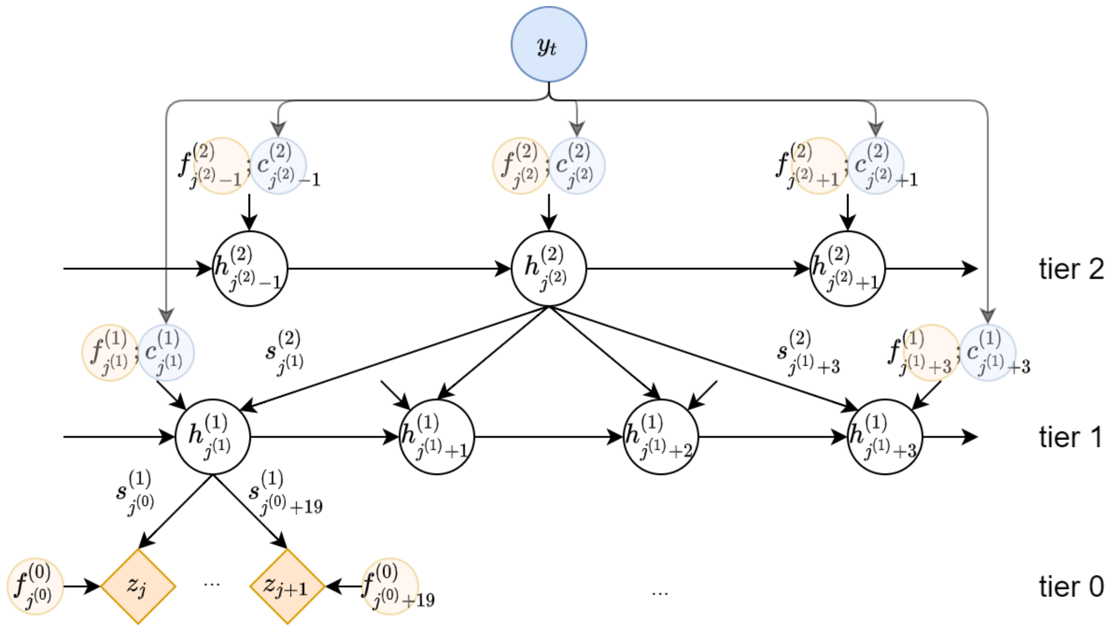

Fig. 6.4 Illustration of a hierarchical RNN [120] with 3 tiers; $\boldsymbol{y}_t$ is a frame of acoustic feature; z_j is a discretized waveform sample, whose distribution is estimated.

6.3.3 SampleRNN

Recall that neural vocoders model the distribution of a waveform sequence conditioned on an acoustic feature sequence, as shown in equation 6.7. It is essential to estimate $p(z_j|z_{1:j-1}, \boldsymbol{y}_{1:T}; \boldsymbol{\phi})$, which is challenging due to waveform sequences being very long. Here $z_{1:J}$ denotes a sequence of discretized waveform samples.

SampleRNNs [81], also referred to as hierarchical RNNs, take into account that waveform samples contain structures at different time scales. It is assumed that the conditional distribution of each waveform sample is a categorical distribution, described in section 6.3.2. To model long-term dependencies, SampleRNNs use a hierarchy of tiers, each operating at a different frequency. Figure 6.4 illustrates the model structure; a 3-tier model is shown but the configuration can be tuned for different tasks. The lowest tier operates at waveform-level frequency, and outputs distributions of waveform samples. Each preceding tier operates at a lower frequency, and supervises the tier following it.

Frame-level Tier(s)

Except the lowest tier, all tiers operate below waveform-level frequency. These tiers are RNNs operating on non-overlapping frames of waveform samples, hence they are also referred to as frame-level tiers. Each frame-level tier summarizes the history of its inputs into a supervising vector for the next tier downward. These tiers can be formulated as

$$\boldsymbol{e}^{(k)}_{j^{(k)}} = \begin{cases} [\boldsymbol{c}^{(k)}_{j^{(k)}}; \boldsymbol{f}^{(k)}_{j^{(k)}}] & \text{if } k = K \\ [\boldsymbol{c}^{(k)}_{j^{(k)}}; \boldsymbol{f}^{(k)}_{j^{(k)}}; \boldsymbol{s}^{(k+1)}_{j^{(k)}}] & \text{if } 0 < k < K \end{cases} \tag{6.14}$$

$$\boldsymbol{h}^{(k)}_{j^{(k)}} = f(\boldsymbol{h}^{(k)}_{j^{(k)}-1}, \boldsymbol{e}^{(k)}_{j^{(k)}}; \boldsymbol{\phi}_h) \tag{6.15}$$

$$\boldsymbol{s}^{(k)}_{(j^{(k)}-1)R^{(k)}:j^{(k)}R^{(k)}} = f(\boldsymbol{h}^{(k)}_{j^{(k)}}, R^{(k)}; \boldsymbol{\phi}_s) \tag{6.16}$$

$K + 1$ is the number of tiers and k the tier index. $[;]$ denotes concatenation. $\boldsymbol{\phi}_h$ and $\boldsymbol{\phi}_s$ are parts of of neural vocoder $\boldsymbol{\phi}$. In this section, the convention is that $\boldsymbol{c}$, $\boldsymbol{f}$, $\boldsymbol{s}$ and $\boldsymbol{W}$ denote the conditioning vector, frame vector, supervising vector and weight matrix; $\boldsymbol{e}$ and $\boldsymbol{h}$ denote the input and history vector of the RNN. The letters $\boldsymbol{c}$ and $\boldsymbol{s}$ have been used when describing attention-based sequence-to-sequence models in section 3.1. They are reused here because in both cases, $\boldsymbol{c}$ contains information about the input, and $\boldsymbol{s}$ contains information about the output history.

A frame vector $\boldsymbol{f}^{(k)}_{j^{(k)}}$ includes $F^{(k)}$ previous waveform samples. $F^{(k)}$ is the frame size, and is equal to the number of waveform samples covered by a frame in tier k. At each time step $j^{(k)}$, the RNN makes a history update as a function of the previous history $\boldsymbol{h}^{(k)}_{j^{(k)}-1}$ and the current input $\boldsymbol{e}^{(k)}_{j^{(k)}}$, as shown in equation 6.15. For the top tier ($k = K$), the current input is a linear combination of a frame $\boldsymbol{f}^{(k)}_{j^{(k)}}$ and a conditioning vector $\boldsymbol{c}^{(k)}_{j^{(k)}}$. For intermediate tiers ($0 < k < K$), it also includes a supervising vector $\boldsymbol{s}^{(k+1)}_{j^{(k)}}$ from the next tier upward.

To condition the next tier downward, the history vector is upsampled into $R^{(k)}$ supervising vectors, where $R^{(k)}$ is the ratio between the frame sizes of the two tiers. If $k = 0$, $R^{(k)} = F^{(k)}$; otherwise $R^{(k)} = F^{(k)}/F^{(k-1)}$. This upsampling is realized by a set of $R^{(k)}$ different linear mappings, hence equation 6.16 can be written as:

$$\boldsymbol{s}^{(k)}_{(j^{(k)}-1)R^{(k)}:j^{(k)}R^{(k)}} = \boldsymbol{W}^{(k)}_r \boldsymbol{h}^{(k)}_{j^{(k)}}; \quad 1 \leq r \leq R^{(k)} \tag{6.17}$$

Table 6.1 Relation of time steps at different tiers, taking the configuration shown in figure 6.4 as an example; F and R denote the frame size and upsampling ratio of a tier.

tier	F	R	time step			
2	80	4	1			
1	20	20	1	2	3	4
0	20	N/A	1 | ... | 20	21 | ... | 40	41 | ... | 60	61 | ... | 80

As described in section 6.1.3, each input token is upsampled and corresponds to a fixed number of output tokens. In this work, the conditioning vectors are upsampled from the acoustic features $\boldsymbol{y}_{1:T}$ using linear interpolation, and there are no learnable parameters:

$$\boldsymbol{c}^{(k)}_{1:J^{(k)}} = f(\boldsymbol{y}_{1:T}, F^{(k)}) \tag{6.18}$$

The upsampling rate depends on the frame size, the number of waveform samples covered by a frame in tier k, and the number of waveform samples covered by an acoustic feature vector.

For each tier, the time step $j^{(k)}$ is related to a different frequency. Each time step $j^{(k)}$ at tier k corresponds to $R^{(k)}$ time steps in tier $k-1$. Table 6.1 illustrates how time steps at different tiers are related, taking the configuration shown in figure 6.4 as an example.

Waveform-level Tier

The last tier operates at waveform-level frequency, i.e. the waveform sampling frequency; hence it is referred to as waveform-level tier. Unlike other tiers, this tier uses a DNN with a softmax output layer, denoted by $\boldsymbol{\phi}_z$:

$$\boldsymbol{e}^{(0)}_{j^{(0)}} = [\boldsymbol{f}^{(0)}_{j^{(0)}}; \boldsymbol{s}^{(1)}_{j^{(0)}}] \tag{6.19}$$

$$\hat{z}_{j^{(0)}} \sim p(\cdot|\boldsymbol{e}^{(0)}_{j^{(0)}}; \boldsymbol{\phi}_z) \tag{6.20}$$

At each time step, the input $\boldsymbol{e}^{(0)}_{j^{(0)}}$ is a linear combination of the supervising vector from tier 1 and an overlapping frame including $F^{(0)}$ previous waveform samples; the output is a vector corresponding to a categorical distribution. $\hat{z}_{j^{(0)}}$ is a waveform sample drawn from the categorical distribution. For this tier, the time step is the same as the waveform $z_{1:J}$.

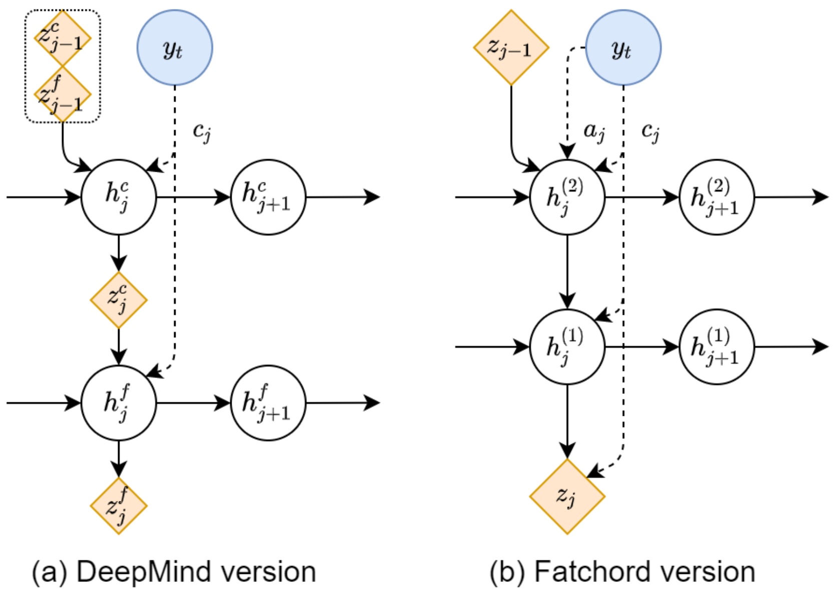

Fig. 6.5 Illustration of WaveRNN [81]; $\boldsymbol{y}_t$ is a frame of acoustic features; z^c_j and z^f_j are the coarse and fine bits of a discretized waveform sample, whose distribution is estimated.

6.3.4 WaveRNN

WaveRNN [81] is a light-weight RNN for audio generation that is designed to efficiently predict 16-bit audio samples. Recall that neural vocoders model the distribution of a waveform sequence conditioned on an acoustic feature sequence, as shown in equation 6.7. At each time step of the waveform, the model estimates the distribution of a waveform sample $p(z_j|z_{1:j-1}, \boldsymbol{y}_{1:T}; \boldsymbol{\phi})$.

As described in section 6.1.3, a neural vocoder does not handle the alignment between the output and input sequences. For WaveRNN, a conditioning network upsamples the acoustic feature sequence, and produces a conditioning sequence with the same frequency as the waveform:

$$\boldsymbol{c}_{1:J} = f(\boldsymbol{y}_{1:T}; \boldsymbol{\psi}) \tag{6.21}$$

This conditioning network is trained as part of the neural vocoder. In the original paper introducing WaveRNN [81], the conditioning network is not described. This section will later describe a CNN-based option, which is used in the experiments.

The left side of figure 6.5 depicts a WaveRNN. it has two RNN layers, followed by DNN layers. Each 16-bit audio sample is split into two 8-bit integers, respectively coding the coarse and fine parts of the audio sample. The distribution of the audio sample can be factorized into two parts:

$$\begin{aligned} p(z_j|z_{1:j-1}, \boldsymbol{y}_{1:T}; \boldsymbol{\phi}) &\approx p(z_j|z_{1:j-1}, \boldsymbol{c}_{1:J}; \boldsymbol{\phi}) && (6.22)\\ &= p(z_j^c, z_j^f|z_{1:j-1}^c, z_{1:j-1}^f, \boldsymbol{c}_{1:J}; \boldsymbol{\phi}) && (6.23)\\ &= p(z_j^c|z_{1:j-1}^c, z_{1:j-1}^f, \boldsymbol{c}_{1:J}; \boldsymbol{\phi})\, p(z_j^f|z_{1:j}^c, z_{1:j-1}^f, \boldsymbol{c}_{1:J}; \boldsymbol{\phi}) && (6.24) \end{aligned}$$

The two parts are respectively modeled by two RNNs, both using GRU cells. For the RNN modeling the coarse part, the history vector tracks the conditioning vector, and the coarse and fine parts of the previous sample:

$$\begin{aligned} p(z_j^c|z_{1:j-1}^c, z_{1:j-1}^f, \boldsymbol{c}_{1:J}; \boldsymbol{\phi}) &= f(z_j^c, \boldsymbol{h}_j^c; \boldsymbol{\phi}) && (6.25)\\ \boldsymbol{h}_j^c &= f([\boldsymbol{h}_{j-1}^c; \boldsymbol{h}_{j-1}^f], [z_{j-1}^c; z_{j-1}^f], \boldsymbol{c}_j; \boldsymbol{\phi}) && (6.26)\\ \hat{z}_j^c &\sim p(\cdot|\boldsymbol{h}_j^c; \boldsymbol{\phi}) && (6.27) \end{aligned}$$

For the RNN modeling the fine part, the history vector also tracks the coarse part of the current sample:

$$\begin{aligned} p(z_j^f|z_{1:j}^c, z_{1:j-1}^f, \boldsymbol{c}_{1:J}; \boldsymbol{\phi}) &= f(z_j^f, \boldsymbol{h}_j^f; \boldsymbol{\phi}) && (6.28)\\ \boldsymbol{h}_j^f &= f([\boldsymbol{h}_{j-1}^c; \boldsymbol{h}_{j-1}^f], [z_j^c; z_{j-1}^c; z_{j-1}^f], \boldsymbol{c}_j; \boldsymbol{\phi}) && (6.29)\\ \hat{z}_j^f &\sim p(\cdot|\boldsymbol{h}_j^f; \boldsymbol{\phi}) && (6.30) \end{aligned}$$

It is assumed that both the coarse and fine parts follow a categorical distribution. Each RNN hidden state is fed into a DNN, which has two layers. The activation functions are respectively ReLU and softmax. During training, the reference output sequence is available, so the two recurrent layers can be run in parallel. At the inference stage, however, this is not the case. At each time step, the coarse part must be generated before the fine part.

Alternative Version

The original DeepMind version of WaveRNN is an existence proof that a light-weight RNN can achieve high-quality efficient waveform synthesis [81]. Following this line of research, there are several alternative versions. To further improve the synthesis speed, reference [81] introduced sparse WaveRNN and subscale WaveRNN, respectively leveraging weight pruning and subscale sampling techniques. This thesis uses the

ResembleAI/Fatchord [119] version of WaveRNN, which does not depend on the weight pruning or subscale sampling.

The right side of figure 6.5 depicts the Fatchord version of WaveRNN. Similar to the original DeepMind version, it has two recurrent layers with GRU cells, followed by some feedforward layers. At each time step, the input is the concatenation of the previous audio sample, the current conditioning vector and the augmentation vector, which can be viewed as additional conditioning. This input is processed by a linear layer, to adjust its dimensionality. The first recurrent layer takes the processed input, and updates its history vector. This history vector is concatenated with the conditioning vector, and fed into the second recurrent layer. The second history vector is then processed by three feedforward layers to generate the final output. The first two feedforward layers use ReLU activations, and concatenates the input with the conditioning vector. The last feedforward layer is linear, and does not use any activation function or concatenation. Residual connections are applied to the two recurrent layers, and the first feedforward layer, linking the pre-concatenation inputs to the outputs.

A key difference between this version and the original DeepMind version is that the output of this model corresponds to a discretized MoL distribution, instead of a dual categorical distribution. The output vector can be split into three parts: $[\boldsymbol{\pi}; \boldsymbol{\mu}; \boldsymbol{\lambda}]$. As shown in equation 6.13, these three parts respectively correspond to the weight, mean and scale of the mixture.

Conditioning Networks

The goal of a conditioning network is to upsample the acoustic feature sequence, producing a conditioning sequence with the same frequency as the waveform. For WaveRNN, the conditioning network is based on CNNs, and is depicted in figure 6.6. It has several blocks of 1D convolutions followed by a repetition operation, which performs the upsampling. The first block consists of a 1D convolution followed by batch normalization and ReLU activation; the last block is just a 1D convolution. The blocks in between have the same residual network [64] structure: two 1D convolutions, each followed by batch normalization; ReLU is applied after the first batch normalization; the input is added to the output via a residual connection. All the 1D convolutions are along the time axis; the stride is 1 so the time resolution is maintained.

In addition, another network produces a sequence of augmentation vectors, which are used as part of the input to the neural vocoder. As shown in the right side of figure

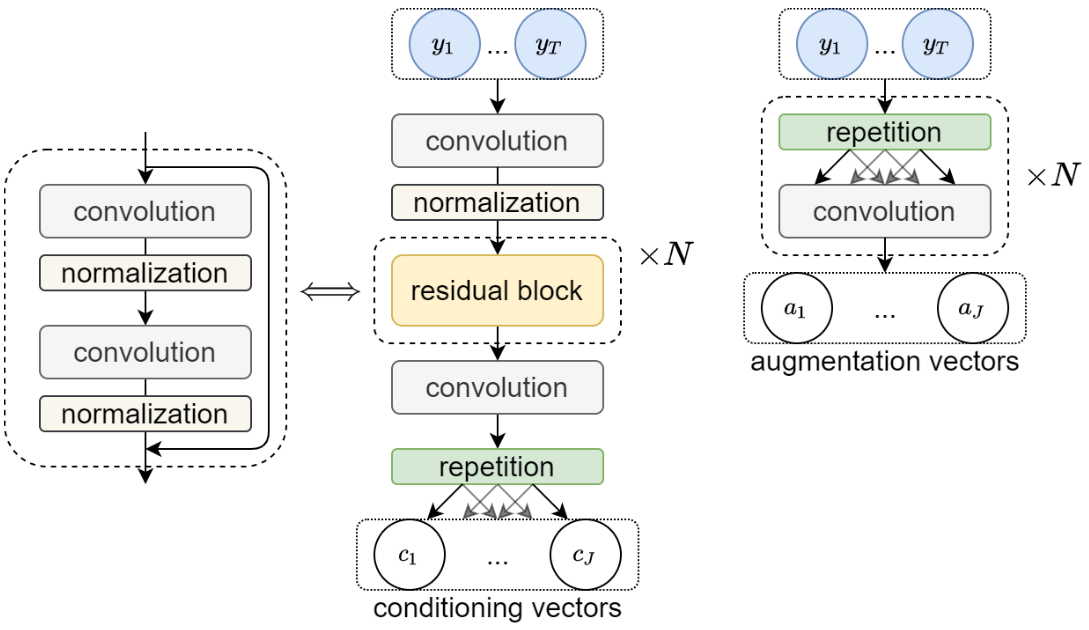

Fig. 6.6 Illustration of conditioning networks; $\boldsymbol{y}_t$ is a frame of acoustic features; $\boldsymbol{c}_j$ is a conditioning vector.

6.6, this network performs the repetition operation in multiple stages, instead of at once. The number of stages and the upsampling factors are hyperparameters. After each repetition, there is a 2D convolution. The filter size along the time axis is one plus twice the upsampling factor, and padding is used so that the time resolution is maintained. The filter size along the acoustic feature axis is set to one, so each dimension of the acoustic feature is processed independently of the others.

6.3.5 Inference and Training

As described in section 6.1.3, neural vocoders can be viewed as special sequence-to-sequence models, where the output and input sequences are aligned. During inference, greedy search or beam search can be used, as described in section 3.2. The models run in an autoregressive fashion: the sampled output at one time step will be included in the input at future time steps. The training approaches described in section 3.3 can also be applied to neural vocoders. Similar to the case of acoustic models, teacher forcing is typically used during training [174, 120, 126, 81], leveraging the reference output history to make training stable.

A standard TTS dataset contains pairs of text and waveform:

$$\mathrm{D} = \{\boldsymbol{x}_{1:L}^{(n)}, \boldsymbol{z}_{1:J}^{(n)}\}_1^N \tag{6.31}$$

where D denotes the dataset of size N; n is the data index; $\boldsymbol{x}_{1:L}^{(n)}$ and $\boldsymbol{z}_{1:J}^{(n)}$ respectively denote a text sequence and a waveform sequence. To simplify the notation, the data index is omitted for the length of the sequences, although they also vary with the index. As described in section 6.7, for each reference waveform $\boldsymbol{z}_{1:J}^{(n)}$, a sequence of reference vocoder features $\boldsymbol{y}_{1:T}^{(n)}$ can be extracted, using a conventional vocoder. Iterating through all the waveform sequences yields the dataset D_ϕ, which is commonly used to train the neural vocoder $\boldsymbol{\phi}$ [115]:

$$\mathrm{D}_\phi = \{\boldsymbol{y}_{1:T}^{(n)}, \boldsymbol{z}_{1:J}^{(n)}\}_1^N \tag{6.32}$$

Pretraining

Conceptually, a neural vocoder should be able to map any sequence of vocoder features to a waveform. It does not even matter if the waveform is speech or other types of audio such as singing. Therefore, this work proposes a pretraining approach, leveraging unlabeled data. Using a conventional vocoder, vocoder features can be extracted for large amounts of unlabeled speech $\{\boldsymbol{z}_{1:J}^{(n)}\}_{N+1}^{N+Q}$, where $Q \gg N$. This yields a new dataset:

$$\mathrm{D}_\phi^Q = \{\boldsymbol{y}_{1:T}^{(n)}, \boldsymbol{z}_{1:J}^{(n)}\}_{N+1}^{N+Q} \tag{6.33}$$

Here the unlabeled speech is not necessarily in-domain: factors such as speaker, speaking style and recording environment can be different from the speech in the target dataset D_ϕ. The neural vocoder can be pretrained with D_ϕ^Q, and then trained with D_ϕ. Empirically, we found that this pretraining approach significantly reduces the need for in-domain data [41]. The corresponding experiments are described in appendix B. The pretraining approach is original, and was developed around the same time as reference [115], which found that while the amount of pretraining data matters, the diversity of the data is also important.

Adaptation

The TTS pipeline consists of a sequence-to-sequence acoustic model $\boldsymbol{\theta}$, followed by a neural vocoder $\boldsymbol{\phi}$, which does not handle the alignment between its input and output. This perfectly matches the scenario described in section 4.3.2. Once the acoustic model is trained, the neural vocoder can be adapted to it. This is done by training the neural vocoder with acoustic features generated from the acoustic model.

For each input text sequence $\boldsymbol{x}_{1:L}^{(n)}$, a vocoder feature sequence $\hat{\boldsymbol{y}}_{1:\hat{T}}^{(n)}$ can be generated. The length of $\hat{\boldsymbol{y}}_{1:\hat{T}}^{(n)}$ is usually from different from that of the reference sequence $\boldsymbol{y}_{1:T}^{(n)}$. Here $\hat{T}$ denotes the length of the generated sequence, and T the reference sequence. In most parts of this thesis, the hat is omitted because the length is not essential to the discussion. More importantly, $\hat{\boldsymbol{y}}_{1:\hat{T}}^{(n)}$ is not aligned with $\boldsymbol{y}_{1:T}^{(n)}$ and $\boldsymbol{z}_{1:J}^{(n)}$. Therefore, it is unsuitable for training the neural vocoder. However, if $\hat{\boldsymbol{y}}_{1:\hat{T}}^{(n)}$ is generated under teacher forcing mode or attention forcing mode, it will have the same length as the reference, i.e. $\hat{T} = T$. The two output sequences will also be aligned. This forms a new dataset $\hat{\mathrm{D}}_{\phi}$:

$$\hat{\mathrm{D}}_{\phi} = \{\hat{\boldsymbol{y}}_{1:T}^{(n)}, \boldsymbol{z}_{1:J}^{(n)}\}_1^N \tag{6.34}$$

From this point on, the length of the generated sequence will be denoted by T. $\hat{\mathrm{D}}_{\phi}$ can be used to train the neural vocoder. In addition, it allows the neural vocoder to fix some mistakes made by the acoustic model.

Conventionally, $\hat{\boldsymbol{y}}_{1:T}$ is generated in teacher forcing mode. This thesis proposes an alternative approach: to use attention forcing instead of teacher forcing. Training the acoustic model with attention forcing has the potential to improve its performance, which is analyzed in section 4.1, and will be empirically demonstrated in section 6.4.3. Furthermore, in attention forcing mode, each output is predicted based on generated back-history, and is more likely than in teacher forcing mode to contain errors made at the inference stage.

6.4 Experiments

The experiments in this chapter investigate attention forcing and deliberation networks in the context of TTS. Sections 6.4.1 and 6.4.2 respectively describe the data and evaluation in general. Each of the following sections focuses on one line of experiments, describing the setup, results and analysis. Section 6.4.3 investigates attention forcing, introduced in chapter 4. The TTS system used in this line was developed before 2018, so section 6.4.4 shifts to a stronger system, which is comparable to state-of-the-art performance in 2021. Section 6.4.5 leverages the new system, and investigates deliberation networks, introduced in chapter 5. The source code and speech samples are available online.[3]

6.4.1 Data

The TTS experiments are conducted on the LJ dataset [77]. This is a public domain speech dataset consisting of 13,100 short audio clips of a single speaker reading passages from 7 non-fiction books. A transcription is provided for each clip. Clips vary in length from 1 to 10 seconds and have a total length of approximately 24 hours. The original sampling rate is 22050 Hz with 16 bits per sample.

The raw text, which is already normalized by the data provider, is split into a character sequence, and embedded as integers ranging from zero to the number of different characters minus one. The text normalization involves converting special symbols and numbers to their verbal form. Three types of acoustic features are extracted from waveforms: PML vocoder features, linear spectrograms, and mel spectrograms. For PML vocoder features, the features are 163D vectors, and the default frequency is 200 Hz, which may be decreased in some experiments. The linear spectrograms are computed through a Short-Time Fourier transform (STFT) using 50 ms frame size, 12.5 ms frame hop, and Hann window function. These features are 1024D vectors, and the frequency is 80 Hz. The linear spectrograms are transformed to the mel-scale using an 80 channel mel filterbank spanning 125 Hz to 7.6 KHz, followed by log dynamic range compression. Prior to log compression, the filterbank output magnitudes are stabilized to a floor of 0.01 in order to limit dynamic range in the logarithmic domain. To train neural vocoders, the waveform samples are discretized into 8-bit integers, following WaveNet [174].

[3] The following web page is made for the experiments in this thesis. Please feel free to contact the author if the page cannot be accessed. `http://mi.eng.cam.ac.uk/~qd212/phdthesis`

6.4.2 Performance Metrics

For TTS, subjective metrics are the gold-standard for overall speech quality, but objective metrics can still indicate speech quality in various ways [179]. This work uses two types of subjective metrics: Mean Opinion Score (MOS) tests and AB preference tests. For both types, a group of participants, also referred to as workers, evaluate speech samples from one or multiple TTS systems. In a MOS test, each worker rates the overall quality of audio samples on the 5-point scale. In a AB preference test, each worker listens to pairs of samples, and indicates which has better overall quality or give no preference. The samples are randomly selected from the test data. The MOS tests measure the overall quality of the TTS models, and helps benchmarking the performance. The AB preference tests show a more direct comparison between two systems. In this work, the MOS tests and AB preference tests are conducted using Amazon Mechanical Turk. Each type of test is taken by at least 30 workers based in the US. Example listening test pages are shown in section B.2 of appendix B.

To objectively measure the overall speech quality, Dynamic Time Warpped (DTW) L_1 distance between the reference and the generated feature sequences is computed. The distance is normalized by the length of the reference and is averaged over the test set and feature dimensions. In addition, Global Variance (GV) is computed and averaged over the test set and feature dimensions. GV reflects the dynamic range of feature trajectories, and roughly indicates the expressiveness [125] of speech. High-quality samples should have low DTW distance and high GV.

6.4.3 Attention Forcing - Initial Development System

The experiments in this section investigate attention forcing, introduced in chapter 4. The following points are empirically tested. First, recall that section 6.2.4 described general training considerations for acoustic models. In particular, exposure bias is more severe due to the high frame rate of the acoustic feature sequence. Hence for teacher forcing, it is essential to use a reduced frame rate. In contrast, attention forcing addresses exposure bias, and allows the use of a higher frame rate. Second, as introduced in section 4.1, attention forcing has the potential to improve the performance of sequence-to-sequence models, i.e. acoustic models in the context of TTS. Finally, section 4.3 introduced how to leverage attention forcing to help down-stream tasks. Section 6.3.5 elaborates on this idea, applying it to neural vocoder adaptation. The corresponding experiments will be described at the end of this section.

Experimental Setup

The experiments are conducted on the LJ dataset [77], introduced in section 6.4.1. The training-validation-test split is 13000-50-50, and the data is shuffled before being split. The speech is down-sampled to 16 KHz, following WaveNet [174]. This sampling rate is a trade-off between accuracy and sequence length, and is empirically validated in [174]. The reference vocoder features are extracted with a PML vocoder [33], as a study comparing various vocoders shows that PML has the best overall performance [1]. Each feature vector has 163 elements. The default frame rate is 200 Hz, which is common for TTS, and is sometimes reduced to 100 Hz for comparison.

As described in section 6.1, the TTS pipeline includes an acoustic model and a neural vocoder. Here the acoustic model is based on Tacotron[182], described in section 6.2.2. Table 6.2 lists its hyperparameters. There are a few differences from the original Tacotron model. First, both the decoder and the post-net predict PML features. Second, the reduction factor is 5 instead of 2, i.e. each decoding step predicts 5 frames. Finally, the attention mechanism is the location-sensitive attention [28], described in section 2.2.1, which is the same as in Tacotron 2 [158]. The attention scores of all past steps are accumulated, convolved with 32 filters of size 31, and then used by the score function.

The neural vocoder is based on SampleRNN [120], described in section 6.3.3. Table 6.3 lists its hyperparameters. Compared with the model used in our previous work [41] investigating SampleRNN as a neural vocoder, the main differences are as follows. First, the Gated Recurrent Unit (GRU) [30] dimension is 512. Second, tiers 1 to 3 each have one layer, instead of two. Finally, the frequencies for tiers 0 to 3 are respectively 16, 8, 2 and 0.4 KHz.

The acoustic models are trained with Teacher Forcing (TF) or Attention Forcing (AF). L_1 loss is used for both the decoder output and the post-net output. The scaling factor γ for the attention loss is 50. The neural vocoders are trained with TF. During training, Adam optimizer [86] is used. $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. By default, for the acoustic models, the batch size is 32; the learning rate is 0.001 and is respectively reduced to 0.0005, 0.0003, and 0.0001 after 500K, 1M and 2M steps. For the neural vocoders, the batch size is 40, and the learning rate is 0.001. As described in section 6.3.5, the neural vocoders are pretrained with reference vocoder features, and then finetuned with generated vocoder features from a TF or AF acoustic model. During

Table 6.2 Hyperparameters of the Tacotron acoustic model. "conv-g-c-ReLU" denotes convolution with width g and c output channels with ReLU activation. "FC" stands for "fully connected".

Character embedding	256D
Encoder CBHG	Conv1D: g=1...16, conv-g-128-ReLU Max pooling: stride=1, width=2 Conv1D: conv-3-128-ReLU → conv-3-128-Linear Highway net: 4 layers of FC-128-ReLU Bidirectional GRU: 128 units
Encoder pre-net	FC-256-ReLU → Dropout(0.5) → FC-128-ReLU → Dropout(0.5)
Decoder pre-net	FC-256-ReLU → Dropout(0.5)→ FC-128-ReLU → Dropout(0.5)
Decoder RNN	2-layer residual GRU (256 units)
Attention RNN	1-layer GRU (256 units)
Attention CNN	Conv1D: conv-31-32-Linear
Post-net CBHG	Conv1D: g=1...8, conv-g-128-ReLU Max pooling: stride=1, width=2 Conv1D: conv-3-256-ReLU → conv-3-80-Linear Highway net: 4 layers of FC-128-ReLU Bidirectional GRU: 128 units
Reduction factor	5

Table 6.3 Hyperparameters of SampleRNN neural vocoder. "FC" stands for "fully connected".

Tier	Frequency	
3	400 Hz	1-layer GRU (512 units)
2	2000 Hz	1-layer GRU (512 units)
1	8000 Hz	1-layer GRU (512 units)
0	16000 Hz	FC-1024-ReLU → FC-1024-ReLU → FC-256-ReLU → FC-256-Softmax

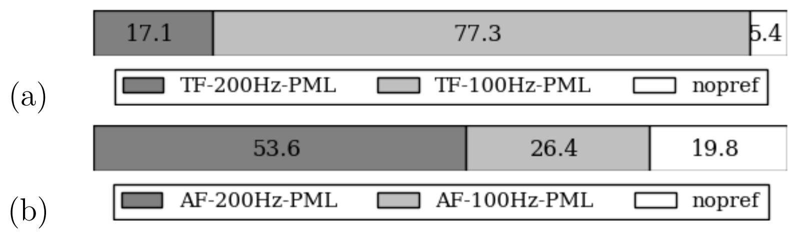

Fig. 6.7 Listening tests comparing 100 Hz and 200 Hz models; acoustic models trained with (a) teacher forcing, (b) attention forcing; waveform generated using a PML vocoder; each number indicates a percentage of preference.

inference, all the models operate in free running mode. The acoustic models adopt greedy search, and the neural vocoders draw samples from the estimated distributions.

The models are compared in AB preference tests, described in section 6.4.2. Over 30 workers from Amazon Mechanical Turk took part in each comparison, which includes five pairs of utterances from the test set. So the test set, which has 50 utterances, is subjectively evaluated about three times. In addition, global variance is computed for each model, to objectively measure the expressiveness [125]; it is averaged over the test set and feature dimensions.

Results and Analysis

To see the impact of AF on frame rate, two pairs of sequence-to-sequence models are trained. The first pair (TF-100Hz, TF-200Hz) is trained with TF; TF-100Hz and TF-200Hz operate at 100 Hz and 200 Hz respectively. The second pair (AF-100Hz, AF-200Hz) has the same frame rates, but is trained with AF. PML vocoders map the features to speech. Figure 6.7 (a) shows the result of the listening test comparing TF-100Hz and TF-200Hz, and figure 6.7 (b) is the equivalence for AF-100Hz and AF-200Hz. Each number indicates a percentage of preference. The results show that reducing the frame rate is beneficial for TF, despite the introduction of some noise. In contrast, AF allows the use of a higher frame rate, which improves the speech quality.

To see the impact of AF on sequence-to-sequence models, the best TF model (TF-100Hz) is compared with the best AF model (AF-200Hz). Figure 6.8 (a) shows the result: AF yields better performance. As for expressiveness, table 6.4 shows the global variances of all the models. It can be seen that AF yields more expressiveness than TF. Doubling the frame rate results in less expressiveness for TF, but not for AF. One likely reason is that AF prevents the model from copying the output history. Note

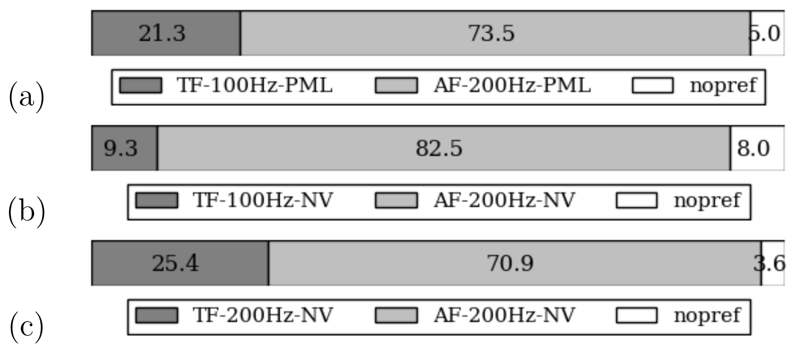

Fig. 6.8 Listening tests comparing teacher forcing and attention forcing; waveform generated using (a) PML vocoder (b,c) neural vocoder; each number indicates a percentage of preference.

the consistency between table 6.4 and figure 6.7, i.e. expressiveness and preference, indicating that expressiveness is important for overall quality. While speed is not the focus of this work, it can be important for TTS, and is reported here for information. For the 200 Hz models, one iteration takes about 3.0s with TF, and 2.6s with AF; generating one second of feature sequence takes about 0.64s. For the 100 Hz models, the time is halved thanks to shorter sequences.

Next, completely neural TTS systems are built, to investigate the synergy between AF and neural vocoders. For a TF system (TF-100Hz-NV or TF-200Hz-NV), a neural vocoder is trained with the vocoder features generated by a TF acoustic model in TF mode. For an AF system (AF-100Hz-NV or AF-200Hz-NV) , a neural vocoder is trained with the vocoder features generated by an AF acoustic model in AF mode. Figure 6.8 (a) and (b) show that AF works better with neural vocoders: when PML vocoders are replaced by neural vocoders, the AF system outperforms the TF system even further. Figure 6.8 (b) and (c) show that AF results in the best neural TTS system. An extra finding is that for TF systems, neural vocoders can fix issues caused by high frame rate. While figure 6.7 (a) shows that TF-100Hz outperforms TF-200Hz when using PML vocoders, figure 6.8 (b) and (c) show that a neural vocoder can help TF-200Hz surpass TF-100Hz. One likely reason is that the neural vocoder alleviates the loss of expressiveness caused by the high frame rate.

6.4.4 Attention Forcing

Since the implementation of the above Tacotron + SampleRNN system, significant progress has been made in the field of TTS. The performance has improved greatly

Table 6.4 Global variance of vocoder features generated by different models, computed over the test set, averaged over all sequences and dimensions; the first row correspond to TF-200Hz and TF-100Hz, and the second row AF-200Hz and AF-100Hz.

	Global variance	
Training	200 Hz	100 Hz
Teacher Forcing (TF)	0.39	0.54
Attention Forcing (AF)	0.71	0.70

in audio quality [158, 183, 195] and efficiency [81, 45]. An increasing number of third-party implementations [119, 62] of Tacotron are reaching comparable performance to that reported by Google.[4] Therefore, the rest of the experiments in this chapter are conducted with a new TTS system, based on the open-source project by ResembleAI [119], which is among the first to reach the reported performance of Tacotron. The approaches introduced in this thesis are implemented on top of that, and compared with a state-of-the-art system in ESPnet[62], an open-source project that implements more up-to-date systems.

This section describes the new TTS system, and conducts further experiments on attention forcing. First, attention forcing is compared to teacher forcing, as a sanity check. Second, scheduled sampling is added to the comparison. Finally, comparisons are made to a system which achieves state-of-the-art performance in 2021. The link to the source code of the new system and its speech samples is given in footnote 3, at the beginning of section 6.4.

Experimental Setup

The data used here is still the LJ dataset, but the pre-processing is different. The previous system uses 163D PML vocoder features as the output of the acoustic model, and the frame rate is 100 Hz or 200 Hz. For the new system, data processing is the same as Tacotron [182]: 80D mel spectrograms and 1024D linear spectrograms are extracted at a frame rate of 80 Hz. Following ESPnet [62], the training-validation-test split is 12600-250-250, and the data is not shuffled before being split.

[4]The official implementation of Tacotron is not publicly available, and it was difficult for third-party implementations to reach the performance reported by Google at https://google.github.io/tacotron/publications/tacotron/index.html

The differences between the previous system and the new system are as follows. First, the Tacotron-based [182] acoustic model is implemented differently, and adopts more design considerations from Tacotron 2 [158]. To be specific, the decoder RNN uses LSTM cells, and has 512 units. The output of the decoder is 80D mel spectrogram, and that of the post-net is 1024D linear spectrogram. The rest of the hyperparameters are the same as listed in table 6.2. On top of that, the neural vocoder is WaveRNN [81], which is more efficient than SampleRNN [41]. This work adopts the ResembleAI version of WaveRNN and the networks upsampling the acoustic features, described in section 6.3.4. Table 6.5 lists their hyperparameters.

The training and inference setup is very similar to the previous experiments, and the differences are as follows. For both the acoustic model and the neural vocoder, the batch size is increased to 100, and the models are initialized with the parameters from the ResembleAI implementation. This is found to greatly speed up convergence without degrading the final performance. For the acoustic model, the learning rate is 10^{-3} for the first 40K iterations, and 10^{-4} afterwards. In addition to teacher forcing and attention forcing, scheduled sampling is investigated. The neural vocoder is trained with teacher forcing, and the learning rate is 10^{-4}. Unlike in the previous experiments, only one neural vocoder is used in this section, in order to save the time of building multiple neural vocoders. This neural vocoder is trained with the vocoder features generated from the baseline acoustic model, which is trained with teacher forcing. With the upgraded TTS system, this neural vocoder is enough for generating high quality speech. Although the baseline has an advantage from neural vocoder adaptation, the novel approaches are able to outperform it, as the following experiments will show.

As is done in the previous experiments, the expressiveness of audio samples is objectively measured by global variance [125]. In addition, Mean Opinion Score (MOS) tests and AB preference tests are conducted. Each type of test is taken by 54 Amazon Mechanical Turk workers based in the US. In the MOS test, each worker rates, for each system, the overall quality of 5 audio samples, randomly selected from the test set for that worker. In the AB preference test, each worker listens to 10 pairs of samples, and indicates which has better overall quality. Following ESPnet [62], for each worker, the samples are randomly selected from the first 100 test sentences. The objective metrics are used over all 250 test sentences.

Table 6.5 Hyperparameters of WaveRNN neural vocoder and upsampling networks. "conv-g-c-ReLU" denotes convolution with width g and c output channels with ReLU activation. "FC" stands for "fully connected". Both the augmentation network and the conditioning network upsample the acoustic features to audio frequency.

WaveRNN	2-layer residual GRU (512 units) $\rightarrow$ FC-512-ReLU $\rightarrow$ FC-512-ReLU $\rightarrow$ FC-30-Linear
Augmentation network	Upsample via repetition: 5 times Conv2D: (1×11)-1-Linear Upsample via repetition: 5 times Conv2D: (1×11)-1-Linear Upsample via repetition: 11 times Conv2D: (1×23)-1-Linear
Conditioning network	Conv1D: conv-5-128-Linear 10 Residual blocks: conv-1-128-ReLU $\rightarrow$ conv-1-128-Linear Conv1D: conv-5-128-Linear Upsample via repetition: 275 times

Results and Analysis

Before moving on to the key experiments, table 6.6 shows the performance gap between the upgraded system and the initial development system, described in section 6.4.3. The two systems use different acoustic features, so the global variance is not comparable. For each system, the performance is measured by the baseline MOS; the acoustic model is trained with teacher forcing, which does not introduce any hyperparameters; the neural vocoder generates the audio samples. The upgraded system adopts the train-validation-test data split of ESPnet[62], and is thus trained with slightly less data: about 97% that of the initial development system. However, its MOS is significantly higher, demonstrating the effect of the system upgrade. Following recent TTS literature [149, 45], the MOS is reported with 95% confidence intervals. Note that this only a convention of the TTS community, and the MOS scores do not necessarily follow a normal distribution.

As is done in section 6.4.3, teacher forcing is compared with attention forcing. A standard baseline model is trained with teacher forcing. A group of attention forcing models are trained. The performance is robust to the scale of the attention loss γ (equation 4.7) being 1, 2 or 4. The model with $\gamma = 1$ is selected for comparison with

Table 6.6 Baseline MOS of the upgraded system and the initial development system.

	MOS↑
Initial Development System	2.87$\pm$0.17
Upgraded System	**3.67**$\pm$0.11

Table 6.7 MOS and global variance of various approaches; upgraded system.

	MOS↑	GV↑
Reference	4.42$\pm$0.09	0.0235
Teacher Forcing (TF)	3.67$\pm$0.11	0.0171
Scheduled Sampling (SS)	3.70$\pm$0.12	0.0167
Attention Forcing (AF)	**3.89**$\pm$0.10	**0.0219**

other approaches. Table 6.7 shows the MOS and the global variances of the models. As observed in the previous experiments: attention forcing yields performance gains in overall quality and expressiveness, on top of the upgraded baseline.

In this section, attention forcing is also compared with scheduled sampling. As analyzed in section 4.4, both approaches address exposure bias, and are between teacher forcing and free running. In previous research applying scheduled sampling to TTS, both positive [110] and negative [60] results have been reported. In this work, a group of acoustic models are trained with scheduled sampling, using different schedules. Based on informal listening tests, the best model is obtained with a linear schedule with ϵ (equation 3.20) decaying from 1 to 0.8 in 20K steps. We found that an ϵ below 0.7 yields noisy speech. The result is shown in table 6.7. While scheduled sampling yields slightly better performance than teacher forcing, it is considerably below attention forcing in both MOS and global variance.

MOS tests depend largely on the design of the test: the platform, the payment, the number of utterances in a test, etc. Therefore AB preference tests are used for more direct comparisons. To demonstrate this point, figure 6.9 shows the AB preference test comparing teacher forcing and attention forcing. While the gap in MOS may not seem significant, attention forcing is strongly preferred over teacher forcing.

Finally, the attention forcing system shown in table 6.7 is compared with a more advanced system. Considering the above analysis, an AB preference test is conducted. The system selected for this comparison is implemented in ESPNet. The acoustic model is based on the Conformer [58] and FastSpeech 2 [149], and the neural vocoder

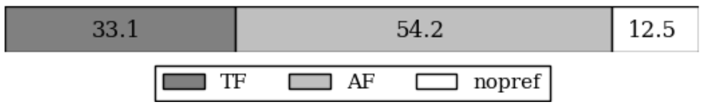

Fig. 6.9 Listening tests comparing Teacher Forcing (TF) and Attention Forcing (AF); upgraded system; each number indicates a percentage of preference.

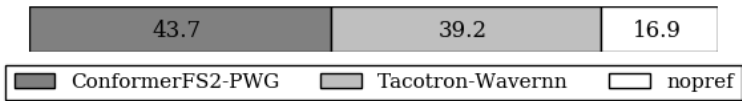

Fig. 6.10 Listening tests comparing a relatively simple system (Tacotron-WaveRNN), which leverages attention forcing, and a state-of-the-art system (ConformerFS2-PWG) in ESPnet; each number indicates a percentage of preference.

is Parallel WaveGAN [190]. The reported MOS for the system is 4.16 ± 0.09, which is at the same level with state-of-the-art in 2021. Figure 6.10 shows the result of the AB preference test. With attention forcing, our relatively simple model achieved comparable performance to a much more complicated model.

6.4.5 Deliberation Networks

The experiments in this section investigate deliberation networks, introduced in chapter 5. The following points are empirically tested. First, as introduced in section 5.1, deliberation networks improve the performance of sequence-to-sequence modeling. Second, addressing exposure bias is an essential element of deliberation networks. This is why all the training approaches in section 5.2 train the second-pass model to correct the free running output from the first-pass model. Finally, recall that section 6.2.4 described general training considerations for acoustic models. In particular, exposure bias is more severe due to the high frame rate of the acoustic feature sequence. Hence for teacher forcing, it is essential to use a reduced frame rate. In contrast, attention forcing and deliberation networks address exposure bias, enabling the use of a higher frame rate. The link to the source code and speech samples is given in footnote 3, at the beginning of section 6.4.

Experimental Setup

The experimental setup is same as described in section 6.4.4. The data is LJ [77], described in section 6.4.1; the acoustic model is based on Tacotron [182], and the neural vocoder is based on WaveRNN [81].

In terms of whether the acoustic features are generated in multiple passes, two types of acoustic models are used: the standard models described in section 6.2 and the corresponding deliberation networks built as introduced in sections 5.1 and 5.3. They will be respectively referred to as single-pass and multi-pass models. The single-pass models and their training are the same as described in section 6.4.4. Teacher forcing, scheduled sampling and attention forcing are respectively used to train the models.

For the multi-pass system, the first-pass model is the standard single-pass baseline. The second-pass model is as introduced in sections 5.1 and 5.3. Its additional encoder, attention mechanism, and the first layer of decoder are randomly initialized, and the rest are initialized with the baseline. The attention RNN state is concatenated with the first-pass output, forming the input to the additional encoder. The separate training approach, introduced in section 5.2.2, is adopted. For the guided attention loss $\mathcal{L}_\alpha$ (equation 5.26), the scale γ is 10, and the sharpness coefficient g (equation 5.25) is 0.4. With these techniques, the second-pass model can be trained in less than 4K steps. During inference, all the models operate in free running mode.

MOS tests and AB preference tests are conducted as described in section 6.4.4. The MOS tests measure the overall quality of the TTS models, and helps benchmarking the performance. The AB preference tests show a more direct comparison between two systems. The expressiveness of audio samples is measured by Global Variance (GV) [125]. In this section, the Dynamic Time Warpped (DTW) L_1 distance between the reference and the generated feature sequences is also computed. The distance is normalized by the length of the reference and is averaged over the test set and feature dimensions. High-quality samples should have low DTW distance and high GV.

Results and analysis

Table 6.8 shows the MOS, GV and DTW L_1 distance of various TTS systems. In terms of MOS, SS is marginally better than TF, and AF outperforms SS. The proposed multi-pass system (FR-TF) outperforms all single-pass systems. Figure 6.11 shows some more direct AB preference comparisons. It is clear that both AF and FR-TF

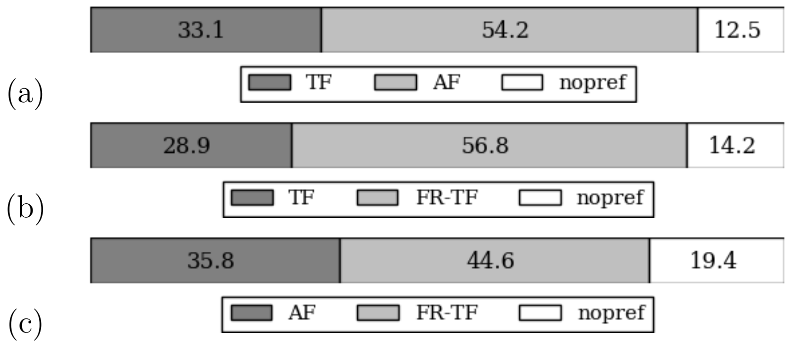

Fig. 6.11 AB preference tests comparing Teacher Forcing (TF), Attention Forcing (AF) and Deliberation Networks (FR-TF); each number indicates a percentage of preference.

Table 6.8 MOS, GV and DTW L_1 distance of various approaches.

	MOS↑	GV↑	DTW↓
Reference	4.42±0.09	0.0235	0
Teacher Forcing (TF)	3.67±0.11	0.0171	6.29
Scheduled Sampling (SS)	3.70±0.12	0.0167	6.02
Attention Forcing (AF)	3.89±0.10	0.0219	**5.59**
FR-TF	**4.03**±0.10	**0.0223**	5.64
TF-TF	—	0.0136	6.73

outperform TF, and that FR-TF is slightly better than AF. The objective metrics show similar trends, meaning that they can be good indicators of human perception. In terms of expressiveness, FR-TF achieves slightly higher GV than AF, although the difference is less obvious in the samples. In general, we find FR-TF less expressive than AF, but more stable. For AF and FR-TF, the empirical frequency of attention failure is respectively 4% and 2%, while the frequency for TF is 1%. Most of the attention failures are due to wrong End-Of-Sequence (EOS) prediction, which is likely to be addressed by introducing a binary EOS predictor [158].

To see what contributes more to the success of the multi-pass system, the training or the deeper model, we run a control experiment, denoted as TF-TF. During training, the first-pass model runs in TF, instead of FR mode. The performance of TF-TF is not nearly as good as FR-TF in GV and DTW. This shows that the performance gain of FR-TF results from fixing errors of the FR output. In other words, it is not enough to simply use a multiple-pass acoustic model, which is deeper and more powerful. Considering its objective performance, TF-TF is excluded in the subjective tests.

An interesting finding was that for the second pass, the initial input seems to be more important than the previous FR output. In an additional experiment, the input text is masked out, and only the FR output is given to the second-pass model. Here the GV and DTW distance are 0.0130 and 7.46, much worse than the baseline. This shows that mapping FR speech to its reference is more difficult than mapping text to the reference speech. In other words, the initial input is more important for the second pass model. This is consistent with the observation that the second-pass model tends to ignore the FR output when no guided attention loss is used.

Following the investigation on frame rate, in sections 6.2.4 and 6.4.3, a TF baseline and a FR-TF system are trained at 200 Hz, where the exposure bias is more severe. While the GV and DTW distance of TF degrade considerably to 0.012 and 7.42, those of FR-TF remain at a similar level (0.2117 and 5.768). This further demonstrates the ability of FR-TF to address the issue.

6.5 Chapter Summary

This chapter investigated speech synthesis as an example sequence-to-sequence task, in order to validate the effectiveness of attention forcing and deliberation networks. Section 6.1 described the speech synthesis pipeline, including text analysis, acoustic modeling and vocoder synthesis. The advantages of state-of-the-art TTS systems, consisting of a sequence-to-sequence acoustic model and a neural vocoder, were analyzed. Section 6.2 reviewed a range of acoustic models, and described in detail the models used in the experiments. In particular, the impact of frame rate was investigated. While it is essential to reduce the frame rate when using teacher forcing, attention forcing and deliberation networks do not have such requirements. Section 6.3 took the same structure to describe neural vocoders. An adaptation technique was proposed, leveraging attention forcing to generate outputs aligned with the references. Section 6.4 reported and analyzed the experimental results. The key findings are as follows. Attention forcing outperforms scheduled sampling and teacher forcing, yielding more natural and expressive speech. Deliberation networks, trained with the proposed separate training approach, outperform the standard single-pass models, including the ones trained with attention forcing. The performance increases result from addressing exposure bias, instead of the models being deeper. In tasks like speech synthesis, the output space is continuous and the attention is monotonic. As a result, applying attention forcing is simpler than applying deliberation networks, where a guided attention loss is required. This can be better demonstrated by comparing to the machine translation experiments in the next chapter. In the future, it would be interesting to extend attention forcing and deliberation networks to duration-based acoustic models, where the attention mechanisms are replaced by duration-based aligners.

Chapter 7

Neural Machine Translation

This chapter investigates machine translation as an example sequence-to-sequence task, in order to validate the effectiveness of attention forcing and deliberation networks. For machine translation, the output space is discrete and multi-modal in the sense that the given an input, the distribution of the corresponding output can be multi-modal. Similar tasks include grammatical error correction and text summarization. The structure of this chapter follows the previous chapter. Section 7.1 describes the machine translation pipeline, including text segmentation, translation and language model rescoring. Section 7.2 reviews a range of translation models, and describes in more depth the models used in the experiments, namely Google's RNN-based model and Transformer. Section 7.3 reports and analyzes the experimental results.

7.1 Pipeline

The aim of machine translation is to convert text in one language to that in another. This can be modeled at different levels, such as document-level, paragraph-level, or sentence-level. This work focuses on sentence-level translation, and view the input and output sentences as sequences. The early machine translation systems are mainly based on hand-crafted translation rules and linguistic knowledge [74]. The pipeline can be viewed as a hand-crafted function that maps an input sequence to an output sequence. Due to the complexity of natural languages, it is difficult to cover all language irregularities with manual translation rules [171]. This work takes a probabilistic approach to machine translation, centering it around the sequence-to-sequence task described at the beginning of chapter 3. As shown in figure 7.1, the pipeline consists of

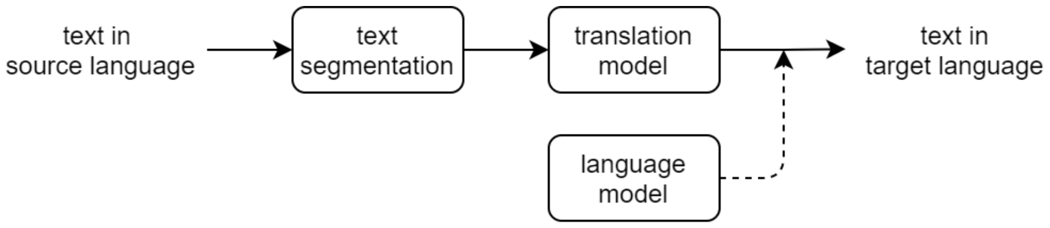

Fig. 7.1 Illustration of a neural machine translation pipeline.

text segmentation, translation, and optionally language model rescoring. The following subsections will respectively discuss each of these elements.

7.1.1 Text Segmentation

Text Segmentation, also referred to as vocabularization, mainly involves splitting text sequences in the training data, and using the obtained tokens to build a vocabulary. Its counterpart in TTS is text analysis, which embeds raw text and optionally leverages other sources of information such as duration models and lexicons, as described in section 6.1.1. In NMT, the output tokens are modeled by discrete distributions. So it is important to define a vocabulary, although translation is fundamentally an open vocabulary problem (names, numbers, dates etc.) [186]. A key challenge then is how to handle out-of-vocabulary words. In this regard, there are three types of approaches: word-level, character-level and sub-word level.

A straight-forward approach is to use a word-level vocabulary, where each token corresponds to a word, space or punctuation. Initially, most NMT models adopted this approach. Despite promising results, these models typically operate with a fixed vocabulary, and have problems handling rare words [188]. Due to practical reasons such as memory constraints, the vocabulary size often ranges from 30k to 50k [171]. The limited size results in a large number of unknown words. Therefore, word-level NMT is unable to translate these words and performs poorly in open-vocabulary settings [168, 6].

Character-level NMT [29, 102, 132] does not have this problem. By splitting words into characters, the vocabulary size is much smaller and every rare word can be represented. On the other hand, splitting words into characters results in longer sequences in which each symbol contains less information. This creates both modeling and computational challenges [26].

Other than word-level and character-level vocabularies, subword-level vocabulary is an option that keeps a balance between vocabulary size and sequence length. Byte-Pair-Encoding (BPE) [157] is a commonly used subword-level approach. The general idea is to merge pairs of frequent character sequences to create sub-word units. BPE makes the NMT model capable of open-vocabulary translation by encoding rare and unknown words as sequences of subword units. This approach consistently achieves better performance over word-level and character-level approaches [171]. Compared to word-level vocabularies, it is more capable of handling rare words. Compared to character-level vocabularies, it has shorter sentence lengths. Moreover, it is an unsupervised approach with few hyperparameters. Several extensions have been proposed to further improve BPE, such as subword regularization [94], BPE-dropout [144] and SentencePiece [95].

7.1.2 Translation

As previously described, the early machine translation systems are mainly rule-based. Such systems can be viewed as a function that maps an input sequence $\boldsymbol{x}_{1:L}$ to an output sequence $\boldsymbol{y}_{1:T}$: $\boldsymbol{y}_{1:T} = f(\boldsymbol{x}_{1:L})$. The function is pre-defined and there are no parameters to train. Due to the complexity of natural languages, it is difficult to cover all language irregularities with manual translation rules [171].

With the availability of large-scale parallel corpora, Statistical Machine Translation (SMT) [90] has gained increasing attention. SMT is based on a probabilistic model, which estimates the conditional probability $p(\boldsymbol{y}_{1:T}|\boldsymbol{x}_{1:L};\boldsymbol{\theta})$. $\boldsymbol{\theta}$ denotes the parameters to be learned. This fits the sequence-to-sequence framework described in chapter 3. During training, $\boldsymbol{\theta}$ is updated to optimize a loss function. At inference stage, $\boldsymbol{\theta}$ is fixed, $\boldsymbol{x}_{1:L}$ is given, and $\boldsymbol{y}_{1:T}$ is generated based on its conditional probability. Example models include N-gram models [114] and log-linear models [152]. In general, SMT struggles to model long-range dependencies, which limits its performance [124].

With the development of deep learning, Neural Machine Translation (NMT) [81, 6, 27] has emerged as a new paradigm and quickly replaced SMT as the mainstream approach to machine translation [171]. NMT adopts the same probabilistic model as SMT, but has some fundamental differences. On the one hand, NMT employs continuous representations instead of discrete symbolic representations in SMT. On the other hand, NMT uses a single large neural network to model the entire translation process, freeing the need for excessive feature engineering. Besides its simplicity, NMT has achieved state-of-the-art performance on various language pairs [171, 175].

7.1.3 Language Model Rescoring

A language model estimates the distribution of a sequence, which could be in either the source or the target language. More formally, let $\boldsymbol{\phi}_x$ and $\boldsymbol{\phi}_y$ denote language models for the source and the target, they respectively estimate $p(\boldsymbol{x}_{1:L}; \boldsymbol{\phi}_x)$ and $p(\boldsymbol{y}_{1:T}; \boldsymbol{\phi}_y)$. While a translation model is usually trained with labeled data, a language model can be trained with unlabeled data, which are much easier to obtain.

There are several ways to integrate a language model into a neural machine translation pipeline. For example, the model can rescore a group of candidates generated by a translation model, e.g. using beam search [113]. The rescoring process can also be fused with the translation. At each decoding step, the probability predicted by the language model and the translation model can be combined, which is referred to as shallow fusion. In contrast, deep fusion concatenates the hidden states of the two models, and fine tunes the down-stream parameters. This approach is more flexible, and gating can be used to decide which model contributes more, depending on the state of the language model [59].

Apart from rescoring, language models are often used in the pretrain-finetune framework, which has improved the performance of a variety of tasks, including machine translation, text summarization, and question answering [146]. Here a language model leverages large amounts of unlabeled data, and serves as a good starting point for another model. Examples include the Generative PreTraining (GPT) series of models [147, 18], which can be used to initialize a decoder generating text.

The pretraining criterion can be extended beyond language modeling, i.e. learning the distribution of sentences. For machine translation, commonly used pretraining criteria include masked language modeling and denoising. Masked language models, e.g. Bidirectional Encoder Representations from Transformers (BERT) [35, 112], are trained to recover masked tokens in text, and can be used as text encoders. Denoising auto-encoders, e.g. Bidirectional and Auto-Regressive Transformers (BART) [103, 111], are trained to recover corrupted text and can be used to initialize a complete translation model [111].

7.2 Translation Models

This thesis does not leverage language models to boost the performance, and focuses on translation models adopting the encoder-attention-decoder architecture described in chapter 3. This section reviews a range of translation models, and then elaborates on the models that will be used in the experiments. Inference and training techniques will be discussed at the end of the section.

7.2.1 General Review

Section 3.1.3 generally discussed the pros and cons of various building blocks. The discussion applies to models used in NMT, and has been recapitulated in section 6.2.1. The first NMT model [168] that achieves comparable performance to SMT is based on RNNs. It consists of an encoder and a decoder, but no attention mechanism, as described in section 3.1.1. More recent RNN-based models adopt attention mechanisms, and achieve significantly better performance. Google's NMT (GNMT) model [186] is a prominent example. Its encoder has both bidirectional and unidirectional RNN layers; its decoder only uses unidirectional RNN layers. Section 7.2.2 will describe this model in detail. RNMT+ [24] adopt GNMT as the starting point, and introduces the following changes to boost the performance. First, RNMT+ makes more extensive use of bidirectional RNN layers, instead of unidirectional RNN layers, trading efficiency for performance. Second, inspired by Transformer [175], RNMT+ uses multi-head attention and layer normalization. To improve convergence speed, a synchronous training strategy [22] is adopted.

Convolutional Sequence-to-Sequence (ConvS2S) is a typical CNN-based translation model [49], achieving similar performance as RNN-based model at that time [192]. Here both the encoder and decoder are constructed by stacking multiple convolutional layers, each layer followed by Gated Linear Units (GLU) [31]. ByteNet [82] is another CNN-based model, where the decoder is dynamically unfolded over the encoder outputs, and dilation is used to increase the receptive field. This model achieved state-of-the-art performance in character-level translation but failed at word-level translation. Hybrid models [80, 27], which use both recurrent and convolutional layers, have also been investigated, but did not outperform fully recurrent or convolutional models.

The original Transformer model achieved state-of-the-art NMT performance at the time [175], and a lot of works follow this line of research. For example, transparent attention

[8] has been introduced to facilitate the training of deeper Transformer models. This attention extended its connection to all encoder layers, which allows the model to flexibly adjust the gradient flow to different layers of the encoder. Universal Transformer [34] uses an adaptive, instead of constant, number of blocks, which helps the model generalize. Semi-autoregressive Transformer [180] produces several consecutive words per time step, in order to improve the translation speed without considerable quality degradation. In terms of training, it is shown that reduced precision and large batch training can speedup training by nearly five times [127]. Section 7.2.3 will describe the Transformer model used in the experiments in this thesis, and section 7.2.4 will describe reduced precision training.

7.2.2 GNMT

GNMT [186] is Google's RNN-based NMT model, which adopts the encoder-attention-decoder architecture described in section 3.1. This thesis uses GNMT in several experiments, considering its high performance across many datasets [24]. Figure 7.2 depicts GNMT. The encoder consists of one bidirectional LSTM layer, followed by seven unidirectional LSTM layers. The decoder has eight unidirectional LSTM layers. Both the encoder and the decoder use residual connections between consecutive layers. The attention mechanism is the additive attention [6], described in section 2.2.1, and is realized with a feed forward network with one hidden layer. It connects the bottom layer of the decoder and the top layer of the encoder.

The design considerations are as follows. Empirically, it is found that both the encoder and decoder RNNs have to be deep enough to capture subtle irregularities in the source and target languages [186]. However, simply stacking more layers of LSTM works only to a certain number of layers [186]. Therefore residual connections are adopted. The decoding process is autoregressive, so the decoder only uses unidirectional layers. Although bidirectional layers can help the encoder extract contextual information related to any position in the input sequence, only the first encoder layer is bidirectional, and the rest are unidirectional. This helps parallelization across depth: the computation for the unidirectional encoder layers can be parallelized after the first few time steps. Only using the bottom decoder layer to compute the attention helps parallelizing the decoder. If the top decoder layer were used, the lower layers would not be able to proceed until the top layer state is updated. In reference [186], the encoder and decoder are partitioned along the depth dimension and are placed on multiple GPUs.

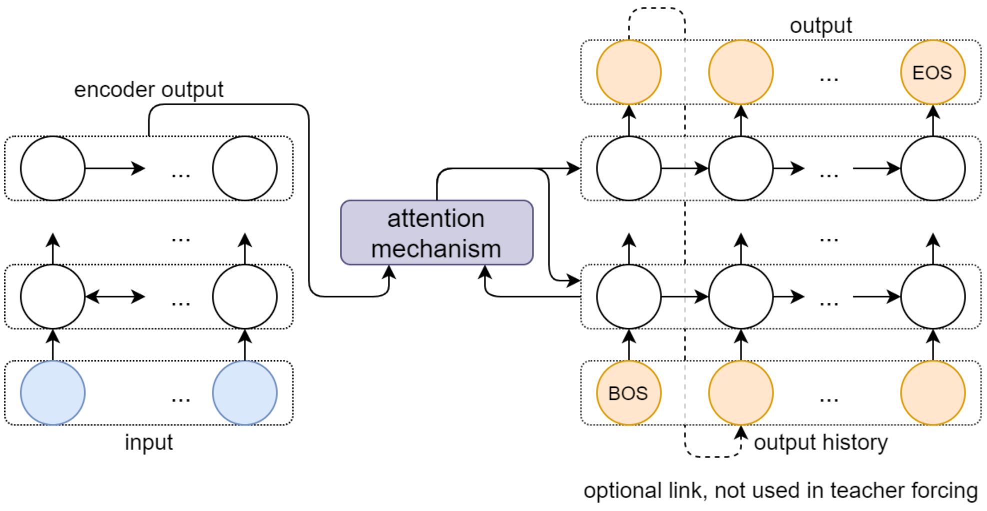

Fig. 7.2 Illustration of GNMT [186], Google's RNN-based NMT model.

7.2.3 Transformer

For Transformer models, [175], the encoder and decoder are construted by stacking multiple Transformer blocks, described in section 2.3. Figure 7.3 depicts the model structure. Each encoder block contains a multi-head self-attention layer, followed by two fully connected feedforward layers with a ReLU activation between them. Each decoder block masks the self-attention to prevent positions from attending to subsequent positions, and inserts a multi-head cross attention after it. In other words, the encoder output is connected with all the decoder layers by a group of cross attentions, integrated into the decoder. All the attention layers adopt scaled dot-product attention, and details about these layers are in sections 2.2.2 and 2.2.3. The attention and feedforward layers are each followed by dropout [165], residual connections [63] and layer normalization [4].

There are no recurrent connections in Transformer during training, hence the model can be trained in parallel across time. The self-attention allows each position in the current layer to access information from all other positions in the previous layer. To compensate for the absence of recurrence, positional encoding is added to the input and output embeddings.

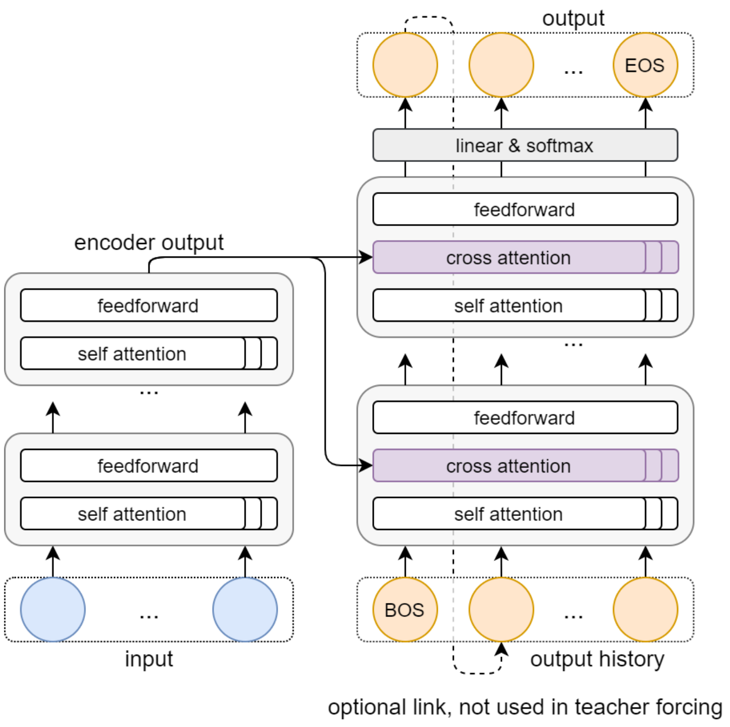

Fig. 7.3 Illustration of Transformer [175], used as a translation model.

7.2.4 Inference and Training

Section 3.2 generally described the inference process for sequence-to-sequence models. For translation models, the output tokens follow discrete distributions, hence both greedy search and beam search can be used during inference. The translation stops when the a special ending token is generated.

Section 3.3 generally described a range of training approaches for sequence-to-sequence models. To the author's knowledge, teacher forcing is the standard training approach for NMT. Sometimes the approaches addressing exposure bias are used to finetune models trained with teacher forcing. In reference [186], GNMT is finetuned with reinforcement learning. However, this requires running the model sequentially during training, which is very expensive for Transformer. Reference [43] applied parallel scheduled sampling to Transformer, but did not observe performance improvements in NMT. To address exposure bias, this thesis adopts attention forcing or deliberation networks, and the application considerations for NMT are respectively introduced in sections 4.3 and 5.3. In this section, we describe reduced precision training, and discuss its impact on attention forcing. Although not specific to NMT, reduced precision training is described in this chapter, because it was originally proposed for Transformer-based NMT [127].

Reduced Precision Training

NVIDIA Volta GPUs enable efficient half precision Floating Point (FP) computations that are several times faster than full precision operations. However, half precision drastically reduces the range of floating point values, which can lead to numerical underflows and overflows [121]. Reference [127] introduced scaling techniques to mitigate this issue. Here all forward-backward computations are performed in FP16. The model weights are available in both FP16 and FP32. The loss is computed in FP32, and then scaled to fit into the FP16 range before the backward pass. The gradients are converted into FP32 before updating the weights. In addition, the loss is automatically scaled down when overflow is detected, and up if no overflows have been detected over many (e.g. 2000) updates.

This thesis adopts half precision training when using Transformer models. In this case, if attention forcing is adopted, the KL loss (equation 4.9) between two attention maps will be replaced by an averaged L_2 loss to improve numerical stability. The reason is as follows. The KL loss may involve taking the log of small numbers, and is more sensitive than L_2 to precision reduction. In addition, Transformer models have multiple attention maps, which makes numerical instability more likely to occur.

7.3 Experiments

The experiments in this chapter investigate attention forcing and deliberation networks in the context of NMT. Sections 6.4.1 and 6.4.2 respectively describe the data and performance metrics. Each of the following sections focuses on one line of experiments, describing the setup, results and analysis. Section 7.3.3 mainly investigates scheduled attention forcing, introduced in chapter 4. The NMT system is based on GNMT, described in section 7.2.2. Using the same system, section 7.3.5 investigates deliberation networks, introduced in chapter 5. Section 7.3.4 investigates parallel attention forcing, introduced in chapter 4, with a different NMT system. The system is based on Transformer, described in section 7.2.3. The source code is available online.[1]

[1] The following web page is made for the experiments in this thesis. Please feel free to contact the author if the page cannot be accessed. `http://mi.eng.cam.ac.uk/~qd212/phdthesis`

Table 7.1 Data used in the machine translation experiments.

	Languages	Number of sentence pairs Training	 Validation	 Test
IWSLT'15	En→Fr	208K	1026	1305
	En→Vi	133K	1553	1268
WMT'16	En→De	4.5M	3000	3003

7.3.1 Data

Table 7.1 lists the information about the datasets used in this thesis. There are two sources of data: IWSLT'15 [19, 20] and WMT'16 [14]. For all the NMT data used in this work, only one reference output is provided for each input. The IWSLT datasets correspond to subtitle translation tasks, where the sentences are from TED talks. Two translation directions are investigated: English-to-French (EnFr) and English-to-Vietnamese (EnVi). These datasets are relatively small, and are used to train RNN-based models. For EnFr, the training set contains 208K sentence pairs. The validation set (tst2013) and test set (tst2014) respectively contain 1026 and 1305 sentence pairs. For EnVi, the training set contains 133K sentence pairs. The validation set (tst2012) and test set (tst2013) respectively contain 1553 and 1268 sentence pairs. The data preprocessing follows reference [116]. The vocabularies are at the word-level, i.e. the units are words. For EnFr, both English and French vocabularies are limited to 50K. For EnVi, the vocabulary sizes are 17K and 7.7K for English and Vietnamese.

The WMT datasets correspond to news translation tasks, where the sentences are from newspaper articles. Here English-to-German (EnDe) translation is investigated. The dataset is considerably bigger, and is used to train Transformer-based models. The training set contains 4.5M sentence pairs. The validation set (newstest13) and test set (newstest14) respectively contain 3000 and 3003 sentence pairs. The data preprocessing follows reference [127]. A joint source and target sub-word vocabulary is built using byte pair encoding, as described in section 7.1.1. The vocabulary is 32K BPE tokens. For all the translation directions, the Moses tokenizer [89] is adopted, and the translations are detokenized before evaluation. The checkpoints are selected based on the validation set, and the results are compared on the test set.

7.3.2 Performance Metrics

The overall translation quality is measured by BLEU [129]. The average of 1-to-4 gram BLEU scores are computed and a 0.6 brevity penalty is applied. Higher BLEU indicates higher quality. For IWSLT and WMT data, the BLEU score is computed using the Moses toolkit [89] and the SacreBLEU toolkit [141], respectively.

In NMT, there can be multiple valid output sequences for a single input sequence. Under the condition that the overall translation quality is the same, it is desirable for an NMT model output to be diverse, i.e. different, translations for the same input. This work measures the diversity of the candidate translations by pairwise BLEU [160] and entropy. For a translation model $\boldsymbol{\theta}$, we use sampling search M times with different random seeds, obtaining a group of translations $\{\hat{\boldsymbol{y}}^{(m)}\}_{m=1}^{M}$. $\hat{\boldsymbol{y}}^{(m)}$ denotes all the output sentences in the dev or test set. Then we compute the average BLEU between all pairs: $\frac{1}{M(M-1)}\sum_{n=1}^{M}\sum_{m=1}^{M}\texttt{BLEU}(\hat{\boldsymbol{y}}^{(n)},\hat{\boldsymbol{y}}^{(m)})_{n\neq m}$. In our experiments, M is set to 5. The more diverse the translations, the lower the pairwise BLEU. In addition to pairwise BLEU, we use greedy search and save the entropy e_t of the output token's distribution at each time step. Let $e_{1:T}$ cover all the model output steps, we compute the average value: $e = \frac{1}{T}\sum_{t=1}^{T} e_t$. Higher entropy means that the model is less certain, and thus more likely to produce diverse outputs. This process is deterministic, and is not repeated with different random seeds.

7.3.3 Scheduled Attention Forcing

The experiments in this section investigate attention forcing. First, attention forcing is applied to machine translation. As analyzed in section 4.3, this is more challenging than applying attention forcing to speech synthesis. Therefore, scheduled attention forcing, introduced in section 4.2.1, is adopted. Furthermore, for speech synthesis, attention forcing is observed to increase the variance of the generated samples, as evidenced by the experimental results in section 6.4.3. Therefore, the translation diversity is investigated, in addition to the overall translation quality. The link to the source code is given in footnote 1, at the beginning of section 7.3.

Table 7.2 Hyperparameters of the RNN-based translation model [186].

Word embedding	200D
Encoder	2-layer bidirectional LSTM (200 units)
Attention	General dot-product attention [117]
Decoder	4-layer unibidirectional LSTM (200 units)

Experimental Setup

The experiments are conducted with English-to-French (EnFr) and English-to-Vietnamese (EnVi) data in IWSLT'15, described in section 7.3.1. As described in section 7.3.2, the overall translation quality is measured by BLEU, and the diversity is measured by pairwise BLEU and entropy.

The model is based on GNMT [186], described in section 7.2.2. The differences are as follows. The model is simplified with a smaller number of LSTM layers due to the small scale of data: the encoder has 2 layers of bidirectional LSTM and the decoder has 4 layers of unidirectional LSTM; the attention mechanism is the general form of dot-product attention [117] described in section 2.2.1; both English and Vietnamese word embeddings have 200 dimensions and are randomly initialized. Table 7.2 summarizes the hyperparameters.

The Adam optimiser is used with a learning rate of 0.002; $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. The maximum gradient norm is set to be 1. If there is a finetuning phase, the learning rate will be halved. The batch size is 50. Dropout is used with a probability of 0.2. By default, the baseline models are trained with Teacher Forcing (TF) for 60 epochs. Starting from the baseline, models are finetuned with Attention Forcing (AF) for 30 epochs. For AF, the scale γ, introduced in section 4.2, of the attention loss is 10. The default inference approach is greedy search, in order to reduce the turnaround time. When investigating diversity, sampling search is also adopted, which replaces the argmax operation by sampling. The checkpoints are selected based on the validation BLEU. For all the training approaches, the effective number of epochs is smaller than the maximum, i.e. training goes on until convergence.

Task	Training	λ	BLEU ↑
EnFr	TF	-	30.70
	AF	-	22.93
	SAF	2.5	31.44
	SAF	3.0	**31.66**
	SAF	3.5	31.34
EnVi	TF	-	25.57
	AF	-	18.27
	SAF	2.5	26.02
	SAF	3.0	25.71
	SAF	3.5	**26.72**

Table 7.3 BLEU of Teacher Forcing (TF), Attention Forcing (AF) and Scheduled Attention Forcing (SAF) with different values of λ; the higher λ is, the more likely the generated output history is used; the models are based on GNMT, and trained with data from IWSLT'15.

Results and Analysis

Overall Performance First, we compare Teacher Forcing (TF) and the default form of Attention Forcing (AF). The preliminary experiments show that when starting from scratch, the TF model performs much better than the AF model. When pretraining with TF is adopted, the BLEU of AF increases from 21.77 to 22.93 for EnFr, and from 13.92 to 18.27 for EnVi. However, it does not outperform TF, as shown by the first two rows in each section of table 7.3. In addition, we observed that AF decreases the BLEU of the pretrained TF model. This result is expected, considering the discrete and multi-modal nature of the NMT output space, analyzed in section 4.3.

Next, TF is compared with Scheduled Attention Forcing (SAF). To speed up convergence, pretraining is used by default. The hyper parameter λ, introduced in section 4.2.1, controls the tendency to use generated outputs. The higher λ is, the more likely the generated output history is used. Table 7.3 shows the BLEU scores of SAF, resulting from different values of λ. With very limited tuning, SAF outperforms TF. The performance is robust in a certain range of λ. In the following experiments, λ is set to 3.0 for EnFr and 3.5 for EnVi.

To reduce the randomness of the experiments, both TF and SAF are run $R = 5$ times with different random seeds. Let $\{\boldsymbol{\theta}^{(r)}\}_{r=1}^{R}$ denote the group of TF models, and $\{\hat{\boldsymbol{\theta}}^{(r)}\}_{r=1}^{R}$ the SAF models. For both groups, the BLEU's mean ± standard deviation

Task	Training	BLEU↑
EnFr	TF	31.10 ± 0.27
	SAF	**31.54** ± 0.14
EnVi	TF	25.86 ± 0.44
	SAF	**26.41** ± 0.33

Table 7.4 BLEU of Teacher Forcing (TF) and Scheduled Attention Forcing (SAF); each approach is run 5 times, and the mean ± std is shown; the models are based on GNMT, and trained with data from IWSLT'15.

Task	Training	Entropy↑	Pairwise BLEU↓
EnFr	TF	1.060 ± 0.047	27.43 ± 0.75
	SAF	1.034 ± 0.013	27.82 ± 0.67
EnVi	TF	1.508 ± 0.012	22.11 ± 0.34
	SAF	**1.582** ± 0.017	**20.75** ± 0.29

Table 7.5 Diversity of Teacher Forcing (TF) and Scheduled Attention Forcing (SAF); each approach is run 5 times, and the mean ± std is shown; the models are based on GNMT, and trained with data from IWSLT'15.

is computed. Table 7.4 shows the results. In terms of mean BLEU, SAF yields a 0.44 gain for EnFr, and a 0.55 gain for EnVi.

Diversity To measure the diversity among the translations, the entropy and pairwise BLEU are computed for $\{\boldsymbol{\theta}^{(r)}\}_{r=1}^{R}$ and $\{\hat{\boldsymbol{\theta}}^{(r)}\}_{r=1}^{R}$. The results are shown in table 7.5. Focusing on the entropy column, SAF leads to higher entropy for EnVi, which indicates higher diversity. For EnFr, SAF and TF lead to similar levels of diversity, especially when the standard deviation is considered. We believe that the difference is due to the nature of the tasks. While English and French have similar syntax and lexicon, English and Vietnamese are more different. When trained with SAF, the EnVi model benefits more from using generated back-history, which is more likely to be different from the reference back-history. The pairwise BLEU shows similar trends. For EnVi, SAF leads to lower pairwise BLEU, i.e. higher diversity. For EnFr, the difference between SAF and TF is negligible.

7.3.4 Parallel Attention Forcing

The experiments in this section continue to investigate attention forcing, using a NMT system based on Transformer, instead of RNN. For this system, parallel training is essential. Therefore, parallel attention forcing, introduced in section 4.2.2, is adopted. As analyzed in section 4.3.3, attention forcing may over regularize Transformer-based models, which have multiple attention mechanisms. Hence the remedy of only forcing selected attention heads is investigated. In this section, parallel attention forcing is compared with both teacher forcing and parallel scheduled sampling. The link to the source code is given in footnote 1, at the beginning of section 7.3.

Experimental Setup

The experiments in this section are conducted with WMT'16 English-to-German (EnDe) data, described in section 7.3.1. Compared with the IWSLT data used in section 7.3.3, the WMT data is more suitable, in terms of the amount of data, for Transformer-based models, which usually have many more parameters than RNN-based models. The overall translation quality is measured by BLEU, and the diversity is measured by pairwise BLEU, as described in section 7.3.2.

The translation models have the same structure as the "big" Transformer in reference [175]. Table 7.6 shows the hyperparameters. The models are optimized with Adam using $\beta_1 = 0.9$, $\beta_2 = 0.98$, and $\epsilon = 1e^{-8}$. Following reference [127], large batches are built to have a maximum of 3584 tokens. The learning rate increases linearly for 4,000 steps to $5e^{-4}$, after which it is decayed proportionally to the inverse square root of the number of steps. Label smoothing [136] is applied with 0.1 weight for the uniform prior distribution over the vocabulary. Dropout is applied with probability 0.3 after each attention or feedforward module. Half precision optimization techniques, described in section 7.2.4, are adopted to speed up training.

The baseline models are trained with Teacher Forcing (TF). Starting from the baseline, other models are finetuned respectively with sequence-level Scheduled Sampling (SS) and Attention Forcing (AF). To keep the benefit of parallel training across time, SS and AF are approximated by their parallel version, as described in section 4.2.2 and reference [43]. The number of iterations is two. The first iteration is conditioned on the reference output, and generates a complete output history, which is then used to condition the second iteration. For SS, the probability of using the reference output

Table 7.6 Hyperparameters of the Transformer-based translation model [175]; "FC" stands for "fully connected".

Sub-word embedding	1024D
Encoder	(Multi-head self-attention $\rightarrow$ Feedforward) $\times$ 6
Decoder	(Multi-head self-attention $\rightarrow$ Multi-head cross attention $\rightarrow$ Feedforward) $\times$ 6
Multi-head attention	16 heads $\times$ (64D query / key / value) $\rightarrow$ 1024D output Scaled dot-product attention [175]
Feedforward	FC-4096-ReLU $\rightarrow$ FC-1024-Linear

decreases linearly from 1 to 0.7; more aggressive schedules are found to degrade the performance. For AF, the scale γ (equation 4.7) of the attention loss is 1000. The default inference approach is beam search with beam size 4. The validation BLEU is monitored to select checkpoints and to stop training when no performance increased is observed after 10 epochs.

Results and Analysis

Table 7.7 lists some preliminary results of TF, parallel SS and parallel AF. Here all the attention heads are constantly forced, regardless of the alignment between the reference and generated output history. Recall that the hyper parameter λ, introduced in section 4.2.1, controls the tendency to use the generated output history. The higher λ is, the more likely the generated output history is used. Compared with TF, parallel SS yields lower BLEU as well as pairwise BLEU. It is difficult to conclude whether the decrease in pairwise BLEU results from higher diversity or lower translation quality. In its parallel version, AF performs similarly to TF. This is probably because the back-history is generated in TF mode. In contrast, when the sequential version of AF is applied, as described in section 7.3.3, the performance is considerably lower than TF.

As analyzed in section 4.2.1, when applying AF to NMT, it is important to turn AF on and off based on the alignment between the reference and the generated outputs. Hence unless otherwise mentioned, a schedule is added to parallel AF in the following experiments. Table 7.8 lists the performance of parallel AF, where the hyperparameter λ of the schedule is tuned. It can be seen that the BLEU remains at the same level.

Table 7.7 Preliminary results comparing Teacher Forcing (TF), Parallel Scheduled Sampling (PSS) and Parallel Attention Forcing (PAF) without a schedule; the models are based on Transformer, trained with WMT'16 EnDe, tested on newstest14.

	λ	BLEU↑	Pairwise BLEU↓
TF	-	28.68	31.12
PSS	-	28.19	**30.17**
PAF	$+\infty$	**28.74**	31.90

Table 7.8 BLEU and Pairwise BLEU of Teacher Forcing (TF), Parallel Scheduled Sampling (PSS) and Parallel Attention Forcing (PAF); higher λ means higher tendency to use AF; the models are based on Transformer, trained with WMT'16 EnDe, tested on newstest14; the bold numbers correspond to the hyperparameter selected ($\lambda = 1.2$), based on the overall performance, for further experiments.

	λ	BLEU↑	Pairwise BLEU↓
TF	-	28.68	31.12
PSS	-	28.19	30.17
PAF	1.1	28.75	31.60
PAF	1.2	**28.57**	**30.62**
PAF	1.3	28.48	31.89
PAF	1.4	28.56	31.99
PAF	1.5	28.47	32.06

However, the pairwise BLEU decreases when the percentage of AF decreases, signaling that AF regularizes the translation model to operate in a safe zone.

Recall that the translation model is a big Transformer. As analyzed in section 4.3.3, the encoder and decoder are connected by multiple attention mechanisms, instead of just one. It is likely that too much information is passed from the TF baseline to the AF model. Therefore, to reduce this information, we only force selected attention heads. To be specific, the first two decoder layers are selected, and reason will be discussed in the next paragraph. In each layer, the number of heads forced are 8, 12 or 16 out of 16. Table 7.9 lists the results of these experiments. It can be seen that when only two layers are forced, the performance of parallel AF surpasses TF in both BLEU and pairwise BLEU. The best performance (first row) is achieved when 8 heads are forced in each layer. To reduce the randomness of hyperparameter tuning, we run another experiment where the other 8 heads are forced, and the result (second row) is comparable to the best performance.

Table 7.9 BLEU and Pairwise BLEU of Teacher Forcing (TF), Parallel Scheduled Sampling (PSS) and Parallel Attention Forcing (PAF), where only selected attention heads are forced; the models are based on Transformer, trained with WMT'16 EnDe, tested on newstest14.

	λ	Layers	Heads	BLEU↑	Pairwise BLEU↓
TF	-	-	-	28.68	31.12
PSS	-	-	-	28.19	30.17
PAF	1.2	1-2	1-8	**29.04**	**30.47**
PAF	1.2	1-2	9-16	**28.91**	**30.05**
PAF	1.2	1-2	1-12	28.86	30.87
PAF	1.2	1-2	1-16	28.64	30.79

Table 7.10 Ablation study on forcing selected attention heads; BLEU and Pairwise BLEU of Teacher Forcing (TF), Parallel Scheduled Sampling (PSS) and Parallel Attention Forcing (PAF) without a schedule; the models are based on Transformer, trained with WMT'16 EnDe, tested on newstest14.

	λ	Layers	Heads	BLEU↑	Pairwise BLEU↓
TF	-	-	-	28.68	31.12
PSS	-	-	-	28.19	30.17
PAF	$+\infty$	1-2	1-4	28.38	31.43
PAF	$+\infty$	1-2	1-8	28.53	31.28
PAF	$+\infty$	1-2	1-12	28.80	31.85
PAF	$+\infty$	1-2	1-16	28.74	31.31
PAF	$+\infty$	3-4	1-16	27.94	31.61
PAF	$+\infty$	5-6	1-16	27.46	32.29

Table 7.8 has shown that adding a schedule itself is not enough for parallel AF to surpass TF. Another series of experiments show that limiting the information passed from the TF baseline is also not enough. In other words, the two techniques must be combined. Table 7.10 shows the results of forcing selected heads, without using a schedule. As analyzed in section 4.3.3, different layers in a Transformer model perform different roles. The last three rows show that forcing layers 1 and 2 yields the best performance. The first four rows show that once the layers are selected, forcing more than four heads generally leads to better performance in BLEU and pairwise BLEU. This is the motivation behind the setup of the experiments described in the previous paragraph.

7.3.5 Deliberation Networks

The experiments in this section investigate deliberation networks, introduced in chapter 5. The following points are empirically tested. First, as introduced in section 5.1, deliberation networks improve the performance of sequence-to-sequence modeling. Second, addressing exposure bias is an essential element of deliberation networks. In other words, the performance increases come not from using deeper models, but training the second-pass model to correct the free running output from the first-pass model, as is done by all the training approaches in section 5.2. Finally, various NMT systems are compared, including the teacher forcing baseline, scheduled sampling, attention forcing and deliberation networks. The link to the source code is given in footnote 1, at the beginning of section 7.3.

Experimental Setup

The experiments are conducted with the English-to-French data in IWSLT [19] 2015. BLEU [129] is used to measure the overall translation quality. Each model is trained five times with different random seeds and the mean ± standard deviation is reported.

Regarding whether the final output is generated in multiple passes, two types of models are trained: the standard models described in section 7.2 and the corresponding deliberation networks built as described in section 5.3. They will be respectively referred to as single-pass and multi-pass models. The single-pass baseline model is based on GNMT, and is the same as in section 7.3.3. This model is trained with Teacher Forcing (TF). Starting from the baseline, two stronger single-pass models are finetuned respectively with sequence-level Scheduled Sampling (SS) and Attention Forcing (AF). For SS, the probability of using the reference output decreases linearly from 1 to 0.9 during training; more aggressive schedules are found to degrade the performance. For AF, the scale γ, introduced in section 4.2, of the attention loss is 10. Details about the training of the single-pass models are described in section 7.3.3.

As for the multi-pass system, the first-pass model is the same as the baseline, trained with TF and then used to generate a Free Running (FR) output. The second-pass model is constructed on top of the first-pass model, by adding an additional encoder and attention over the free running output. The dimension of the decoder's input layer is increased by 400, because an extra context vector is taken. The second-pass model is randomly initialized and trained with TF for 50 epochs. For all the models, the inference approach is greedy search. The checkpoints are selected based on the validation BLEU. For all the training approaches, the effective number of epochs is smaller than the maximum, i.e. training goes on until convergence.

Results and Analysis

Table 7.11 shows the BLEU scores of various translation systems. TF, SS and AF denote single-pass models trained with different approaches. SS and AF outperform TF in BLEU, as they address the exposure bias. SS yields slightly higher translation diversity than AF, as indicated by the pairwise BLEU. FR-TF denotes the proposed multi-pass system. It outperforms all of the above in both BLEU and pairwise BLEU. To see if this results from fixing the errors in the free running output, instead of using a bigger model, an extra experiment is run. TF-TF denotes this experiment, where the first-pass output is generated in TF instead of FR mode. TF-TF has the same number of parameters as FR-TF, but its BLEU score is considerably lower. This indicates that the performance gain of FR-TF mainly results from fixing the errors in the free running output. It can be observed that the multi-pass systems have considerably higher translation diversity than the single-pass systems. This is to some extent because the multi-pass systems has two stages of decoding, where the output tokens are sampled from their distributions. In terms of pairwise BLEU, FR-TF outperforms TF-TF, indicating that addressing exposure bias improves translation diversity. Similar results have been observed in the speech synthesis experiments in section 6.4.5.

Compared with AF, FR-TF yields slightly higher BLEU, and considerably lower pairwise BLEU, i.e. higher diversity. While both approaches outperform the TF baseline, FR-TF is easier than AF to apply to the RNN-based machine translation model. As analyzed in section 4.3, it is challenging to apply AF to tasks where the output space is discrete and multi-modal. Recall from section 7.3.3 that the basic form of AF does not outperform TF, and it is essential to use the selection scheme introduced in section 4.2.1. The speech synthesis experiments in section 6.4.5 tell a slightly different story. When applied to the RNN-based speech synthesis model, both AF and FR-TF outperform TF. However, here the basic form of AF is enough, while FR-TF requires the guided attention loss introduced in section 5.3. This is because FR-TF tends to ignore the additional input sequence when it is long and continuous.

An interesting finding was that for the second pass, the initial input seems to be more important than the previous free running output. In an additional experiment, the

Table 7.11 Performance of various translation systems; the single-pass models are based on GNMT; the multi-pass models are the corresponding deliberation networks; all the models are trained with IWSLT'15 EnFr, tested on tst14.

	Training	BLEU↑	Pairwise BLEU↓
Single-pass	Teacher Forcing (TF)	31.10 ± 0.27	27.43 ± 0.75
	Scheduled Sampling (SS)	31.45 ± 0.45	27.16 ± 0.60
	Attention Forcing (AF)	31.54 ± 0.14	27.82 ± 0.67
Multi-pass	FR-TF	**31.74** ± 0.27	**23.66** ± 0.57
	TF-TF	31.29 ± 0.05	25.59 ± 0.90

input text is masked, and only the free running output is given to the second-pass model. The BLEU dropped to 29.31, even lower than the baseline number 31.10. This indicates that mapping French text with errors to its clean version is more difficult than mapping English to French. Similar results have also been observed in the speech synthesis experiments in section 6.4.5: learning to refine generated speech is more difficult than learning to generate speech from text.

The differences between the experiments in machine translation and speech synthesis are also interesting. In the speech synthesis experiments, it is essential to initialize the second-pass models from their corresponding first-pass models. The additional attention over the generated speech is randomly initialized, but is regularized with the guided attention loss proposed in section 5.3. In contrast, in the machine translation experiments, the second-pass models are randomly initialized, and the additional attention over the generated text is not regularized. This confirms the analysis in section 5.3 that it is more difficult to learn the attention over longer sequences.

7.4 Chapter Summary

This chapter investigated machine translation as an example sequence-to-sequence task, in order to validate the effectiveness of attention forcing and deliberation networks. Section 7.1 described the machine translation pipeline, including text segmentation, translation and language model rescoring. Section 7.2 reviewed a range of translation models, and described in more depth the models used in the experiments, namely GNMT and Transformer. Half precision training techniques were described, and an alternative alignment loss was proposed to address the numerical issues in half precision attention forcing. Section 7.3 reported and analyzed the experimental results.

The key findings are as follows. Briefly speaking, applying deliberation networks to machine translation is simpler than applying attention forcing, which requires techniques such as the selection scheme. For GNMT, attention forcing and scheduled sampling yield similar performance, and both of them outperform the teacher forcing baseline. Deliberation networks, trained with the proposed separate training approach, outperform the standard single-pass models. The performance increases result from addressing exposure bias, instead of the models being deeper. For Transformer, parallel scheduled sampling has similar translation quality but higher diversity than teacher forcing. Parallel attention forcing outperforms parallel scheduled sampling in translation quality, while maintaining the level of diversity. In the future, it would be interesting to apply deliberation networks to Transformer-based models, leveraging parallel training.

Chapter 8

Conclusions

This thesis investigated attention-based sequence-to-sequence models, with a focus on training approaches. Attention forcing and deliberation networks were introduced to improve the performance of sequence-to-sequence modeling. Attention forcing trains a model with the reference attention and the generated output history, so that the model learns to recover from its mistakes. Deliberation networks use multiple sequence-to-sequence models to generate the final output through multiple stages of refinement. Each stage learns to refine the erroneous output of the preceding stage.

When applying the novel approaches to specific sequence-to-sequence tasks, it is essential to address the application-specific challenges. Therefore, several extensions were introduced to attention forcing and deliberation networks. In this thesis, speech synthesis and machine translation were investigated as example tasks. Speech synthesis represents tasks where the output space is continuous and the attention is monotonic. Machine translation represents tasks where the output space is discrete and the attention is more complicated.

In terms of thesis structure, chapter 2 reviewed the fundamentals of deep learning, covering commonly used building blocks and general training techniques. Chapter 3 described sequence-to-sequence models that adopt the encoder-attention-decoder architecture. In particular, a range of existing training approaches were analyzed. Chapter 4 introduced attention forcing, covering the general framework and application considerations. Chapter 5 adopted the same structure to introduce deliberation networks. Chapter 6 investigated speech synthesis, describing the pipeline and analyzing the experimental results. In a similar fashion, chapter 7 investigated machine translation.

The rest of this chapter will recapitulate the major contributions of this work in more depth, and introduce several directions for future work.

8.1 Review of Contributions

Attention forcing, introduced in chapter 4, is a novel training approach, which guides the sequence-to-sequence model with the generated output history and reference attention. The training criterion is a combination of the log-likelihood of the reference output and the KL-divergence between the reference attention and the generated attention. An advantage of this approach is that it does not rely on a heuristic schedule or a classifier to stabilize training. In addition, it does not require the sequence-to-sequence task to have a well-established sequence-level criterion. In contrast, the standard training approach, teacher forcing suffers from exposure bias: during training the model is guided with the reference output, but the generated output must be used at inference stage. To address exposure bias, scheduled sampling and professor forcing guide a model with both the reference and the generated output history. However, they depend on a heuristic schedule or an auxiliary classifier, which can be difficult to tune. Alternatively, sequence-level training guides a model with the generated output history, and optimizes a sequence-level criterion. The generation process is often sequential, which is undesirable for parallelizable models.

The speech synthesis experiments in section 6.4.3 demonstrate that attention forcing outperforms both teacher forcing and scheduled sampling. It improves both the overall quality and the expressiveness of speech. However, the machine translation experiments in section 7.3.3 indicate that additional techniques are needed when the output space is discrete and multi-modal. This problem is analyzed in section 4.3, and scheduled attention forcing is introduced in section 4.2 to address the problem. Here a selection scheme automatically turns attention forcing on and off depending on the mode of attention, making sure that the training criterion is sensible.

Parallel attention forcing, also introduced in section 4.2, is another extension, which approximates the sequential generation of the output history with a parallel generation process. This facilitates applying attention forcing to models that can be trained in parallel across time, such as the Transformer. When using Transformer-based models, a lot of information is available in the attention maps. To avoid over regularization, it is important to limit the information passed to the attention forcing model. Therefore, section 4.3 introduced the trick of only forcing selected attention heads. The experiments in section 7.3.4 demonstrate that parallel attention forcing, combined with the previously mentioned selection scheme, improves the overall quality and translation diversity in Transformer-based machine translation.

As discussed in sections 4.3, and 6.3.5, attention forcing can be used to generate outputs aligned with their references, which can be helpful for down-stream tasks, such as neural vocoder adaptation. In addition, a pretraining technique is introduced for neural vocoders in section 6.3.5. A conventional neural vocoder is used to extract vocoder features from large amounts of unlabeled speech. The corresponding experiments are described in appendix B.

Deliberation networks, introduced in chapter 5, involve novel ideas about modeling and training. Here the output sequence is generated in multiple passes, each one conditioned on the initial input and the free running output of the previous pass. As analyzed in section 5.2, the models can be trained either jointly or separately. This work proposes the separate training approach, which is more suitable for parallelizable models. Here the multiple passes are trained in turn with teacher forcing. For all passes but the last, the model is fixed, after training, to generate free running outputs, which will be stored for the training of the next pass. Deliberation networks were originally proposed as deeper networks that leverage additional attention mechanisms to access both past and future context when decoding [187]. In this thesis, it is argued that deliberation networks address exposure bias, which is essential for performance gains. This analysis and the effectiveness of the separate training approach, are empirically demonstrated by both the machine translation experiments in section 7.3.5 and the speech synthesis experiments in section 6.4.5.

As described in section 5.3, for tasks such as speech synthesis and voice conversion, the output sequences are relatively long. As a result, the multi-pass model tends to converge to the standard single-pass model, ignoring the previous output. To tackle this issue, a guided attention loss is proposed to encourage more extensive use of the free running output. The speech synthesis experiments in section 6.4.5 demonstrate the issue and the effectiveness of the guided attention loss.

8.2 Future work

As discussed in section 4.4, attention forcing can be extended to sequence-to-sequence models that do not have attention mechanisms. For convolutional neural networks, for example, attention maps can be defined based on the activation or gradient [199]. Some recent work on TTS [150, 149, 195, 39] uses a duration model instead of attention. In this case, the duration can be forced in the place of attention. During training, the reference duration can be obtained from a external aligner and given to the decoder. The duration model is trained to predict the reference duration, and the predicted duration will be given to the decoder at inference stage.

In terms of application, parallel attention forcing has been applied to Transformer-based models in the machine translation experiments. It is natural to conduct corresponding experiments in speech synthesis. Deliberation networks can also be applied to Transformer-based models, which can leverage the efficiency of the proposed separate training approach.

For deliberation networks, the various training approaches explored in section 5.2 can be further investigated. The investigation can follow the research on minimum risk training, discussed in section 3.3. In particular, it would be helpful to mathematically characterize different ways to approximate the expected risk.

For both attention forcing and deliberation networks, it would be interesting to see how they interact with pretrained models. In other words, it is unknown whether these approaches can improve the sequence-to-sequence models initialized with pretrained models, which leverage large amounts of unlabeled data. Finally, as general approaches introduced for sequence-to-sequence tasks, attention forcing and deliberation networks can be applied to more tasks such as voice conversion, speech recognition, and text summarization.

References

[1] Airaksinen, M., Juvela, L., Bollepalli, B., Yamagishi, J., and Alku, P. (2018). A comparison between straight, glottal, and sinusoidal vocoding in statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1658–1670.

[2] Arık, S. Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., Li, X., Miller, J., Ng, A., Raiman, J., et al. (2017). Deep voice: Real-time neural text-to-speech. In *International Conference on Machine Learning*, pages 195–204. PMLR.

[3] Arık, S. Ö., Jun, H., and Diamos, G. (2018). Fast spectrogram inversion using multi-head convolutional neural networks. *IEEE Signal Processing Letters*, 26(1):94–98.

[4] Ba, L. J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450.

[5] Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. (2017). An actor-critic algorithm for sequence prediction. *5th International Conference on Learning Representations, ICLR.*

[6] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *CoRR.*

[7] Bao, Y., Chang, S., Yu, M., and Barzilay, R. (2018). Deriving machine attention from human rationales. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP.*

[8] Bapna, A., Chen, M. X., Firat, O., Cao, Y., and Wu, Y. (2018). Training deeper neural machine translation models with transparent attention. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3028–3033.

[9] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179.

[10] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[11] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. (2007). Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160.

[12] 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.

[13] Bińkowski, M., Donahue, J., Dieleman, S., Clark, A., Elsen, E., Casagrande, N., Cobo, L. C., and Simonyan, K. (2019). High fidelity speech synthesis with adversarial networks. In *International Conference on Learning Representations*.

[14] Bojar, O., Chatterjee, R., Federmann, C., Graham, Y., Haddow, B., Huck, M., Yepes, A. J., Koehn, P., Logacheva, V., Monz, C., et al. (2016). Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198.

[15] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

[16] Bottou, L. and Bousquet, O. (2008). The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168.

[17] Britz, D., Goldie, A., Luong, M.-T., and Le, Q. (2017). Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451.

[18] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T. J., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*.

[19] Cettolo, M., Girardi, C., and Federico, M. (2012). Wit3: Web inventory of transcribed and translated talks. In *Conference of european association for machine translation*, pages 261–268.

[20] Cettolo, M., Niehues, J., Stüker, S., Bentivogli, L., Cattoni, R., and Federico, M. (2015). The IWSLT 2015 evaluation campaign. In *Proceedings of the 12th International Workshop on Spoken Language Translation: Evaluation Campaign*.

[21] Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. (2016). Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

[22] Chen, J., Monga, R., Bengio, S., and Jozefowicz, R. (2016). Revisiting distributed synchronous SGD. In *International Conference on Learning Representations Workshop Track*.

[23] Chen, M., Tan, X., Ren, Y., Xu, J., Sun, H., Zhao, S., and Qin, T. (2020). Multispeech: Multi-speaker text to speech with Transformer. *Proc. Interspeech 2020*, pages 4024–4028.

[24] Chen, M. X., Firat, O., Bapna, A., Johnson, M., Macherey, W., Foster, G., Jones, L., Schuster, M., Shazeer, N., Parmar, N., et al. (2018). The best of both worlds: Combining recent advances in neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 76–86.

[25] Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561.

[26] Cherry, C., Foster, G., Bapna, A., Firat, O., and Macherey, W. (2018). Revisiting character-based neural machine translation with capacity and compression. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4295–4305.

[27] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). 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.

[28] Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, pages 577–585.

[29] Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1693–1703.

[30] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS 2014 Workshop on Deep Learning.*

[31] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.

[32] Degottex, G., Lanchantin, P., and Gales, M. (2017). A log domain pulse model for parametric speech synthesis. *IEEE/ACM Transactions on audio, speech, and language processing*, 26(1):57–70.

[33] Degottex, G. A., Lanchantin, P. K., and Gales, M. J. (2016). A pulse model in log-domain for a uniform synthesizer. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 230–236. IEEE.

[34] Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., and Kaiser, L. (2018). Universal Transformers. In *International Conference on Learning Representations.*

[35] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional Transformers for language understanding. 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 4171–4186.

[36] Dinh, L., Krueger, D., and Bengio, Y. (2015). NICE: Non-linear independent components estimation. *CoRR*, abs/1410.8516.

[37] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017). Density estimation using real NVP. In *ICLR (Poster)*. OpenReview.net.

[38] Donahue, C., McAuley, J., and Puckette, M. (2018). Adversarial audio synthesis. In *International Conference on Learning Representations*.

[39] Donahue, J., Dieleman, S., Bińkowski, M., Elsen, E., and Simonyan, K. (2021). End-to-end adversarial text-to-speech. *9th International Conference on Learning Representations, ICLR*.

[40] Dou, Q. (2017). Waveform level synthesis. Mphil thesis, University of Cambridge, department of engineering.

[41] Dou, Q., Wan, M., Degottex, G., Ma, Z., and Gales, M. J. (2018). Hierarchical RNNs for waveform-level speech synthesis. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 618–625. IEEE.

[42] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[43] Duckworth, D., Neelakantan, A., Goodrich, B., Kaiser, L., and Bengio, S. (2019). Parallel scheduled sampling. *arXiv preprint arXiv:1906.04331*.

[44] Elias, I., Zen, H., Shen, J., Zhang, Y., Jia, Y., Skerry-Ryan, R., and Wu, Y. (2021a). Parallel Tacotron 2: A Non-Autoregressive Neural TTS Model with Differentiable Duration Modeling. In *Proc. Interspeech 2021*, pages 141–145.

[45] Elias, I., Zen, H., Shen, J., Zhang, Y., Jia, Y., Weiss, R. J., and Wu, Y. (2021b). Parallel Tacotron: Non-autoregressive and controllable TTS. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5709–5713. IEEE.

[46] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

[47] Fan, Y., Qian, Y., Xie, F.-L., and Soong, F. K. (2014). TTS synthesis with bidirectional lstm based recurrent neural networks. In *Fifteenth annual conference of the international speech communication association*.

[48] Gales, M. (2016). Lecture notes on deep learning.

[49] 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.

[50] Gibiansky, A., Arik, S., Diamos, G., Miller, J., Peng, K., Ping, W., Raiman, J., and Zhou, Y. (2017). Deep voice 2: Multi-speaker neural text-to-speech. *Advances in neural information processing systems*, 30.

[51] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[52] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

[53] Graves, A. (2012). Sequence transduction with recurrent neural networks. *29th International Conference on Machine Learning, ICML*, abs/1211.3711.

[54] Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *CoRR*.

[55] Grefenstette, E., Hermann, K. M., Suleyman, M., and Blunsom, P. (2015). Learning to transduce with unbounded memory. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 1828–1836.

[56] Griffin, D. and Lim, J. (1984). Signal estimation from modified short-time Fourier transform. *IEEE Transactions on acoustics, speech, and signal processing*, 32(2):236–243.

[57] Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. (2018). Non-autoregressive neural machine translation. In *International Conference on Learning Representations*.

[58] 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. In *Proc. Interspeech 2020*.

[59] Gulcehre, C., Firat, O., Xu, K., Cho, K., and Bengio, Y. (2017). On integrating a language model into neural machine translation. *Computer Speech & Language*, 45:137–148.

[60] Guo, H., Soong, F. K., He, L., and Xie, L. (2019). A new GAN-based end-to-end TTS training algorithm. *Interspeech*.

[61] Hayashi, T., Watanabe, S., Toda, T., Takeda, K., Toshniwal, S., and Livescu, K. (2019). Pre-trained text embeddings for enhanced text-to-speech synthesis. In *INTERSPEECH*, pages 4430–4434.

[62] Hayashi, T., Yamamoto, R., Inoue, K., Yoshimura, T., Watanabe, S., Toda, T., Takeda, K., Zhang, Y., and Tan, X. (2020). ESPnet-TTS: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7654–7658. IEEE.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[68] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[69] Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Netw.*, 3(5):551–560.

[70] Hsu, Y.-T., Garg, S., Liao, Y.-H., and Chatsviorkin, I. (2020). Efficient inference for neural machine translation. In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*, pages 48–53.

[71] Hu, K., Pang, R., Sainath, T. N., and Strohman, T. (2021). Transformer based deliberation for two-pass speech recognition. In *2021 IEEE Spoken Language Technology Workshop (SLT)*, pages 68–74. IEEE.

[72] Hu, K., Sainath, T. N., Pang, R., and Prabhavalkar, R. (2020). Deliberation model based two-pass end-to-end speech recognition. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7799–7803. IEEE.

[73] Hunt, A. J. and Black, A. W. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 1, pages 373–376. IEEE.

[74] Hutchins, J. (2007). Machine translation: A concise history. *Computer aided translation: Theory and practice*, 13(29-70):11.

[75] Hwang, M.-J., Song, E., Yamamoto, R., Soong, F., and Kang, H.-G. (2020). Improving LPCNet-based text-to-speech with linear prediction-structured mixture density network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7219–7223. IEEE.

[76] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

[77] Ito, K. (2017). The LJ speech dataset. https://keithito.com/LJ-Speech-Dataset/.

[78] Jang, E., Gu, S., and Poole, B. (2017). Categorical reparameterization with Gumbel-softmax. *stat*, 1050:5.

[79] Jiao, Y., Gabryś, A., Tinchev, G., Putrycz, B., Korzekwa, D., and Klimkov, V. (2021). Universal neural vocoding with parallel WaveNet. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6044–6048. IEEE.

[80] Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1700–1709.

[81] Kalchbrenner, N., Elsen, E., Simonyan, K., Noury, S., Casagrande, N., Lockhart, E., Stimberg, F., Oord, A., Dieleman, S., and Kavukcuoglu, K. (2018). Efficient neural audio synthesis. In *International Conference on Machine Learning*, pages 2410–2419. PMLR.

[82] Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *ArXiv*, abs/1610.10099.

[83] Kanagawa, H. and Ijima, Y. (2020). Lightweight LPCNet-based neural vocoder with tensor decomposition. *Proc. Interspeech 2020*, pages 205–209.

[84] Kawahara, H. (2006). Straight, exploitation of the other aspect of vocoder: Perceptually isomorphic decomposition of speech sounds. *Acoustical science and technology*, 27(6):349–353.

[85] Kim, S., Lee, S.-G., Song, J., Kim, J., and Yoon, S. (2019). FloWaveNet: A generative flow for raw audio. In *International Conference on Machine Learning*, pages 3370–3378. PMLR.

[86] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR.*

[87] Kingma, D. P. and Dhariwal, P. (2018). Glow: generative flow with invertible 1×1 convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 10236–10245.

[88] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29:4743–4751.

[89] 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.

[90] Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133.

[91] Kong, J., Kim, J., and Bae, J. (2020). HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*, 33.

[92] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.

[93] Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. (2017). Zoneout: Regularizing RNNs by randomly preserving hidden activations. *5th International Conference on Learning Representations, ICLR.*

[94] Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75.

[95] Kudo, T. and Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

[96] Kumar, K., Kumar, R., de Boissiere, T., Gestin, L., Teoh, W. Z., Sotelo, J., de Brébisson, A., Bengio, Y., and Courville, A. C. (2019). MelGAN: Generative adversarial networks for conditional waveform synthesis. *Advances in Neural Information Processing Systems*, 32.

[97] Lamb, A. M., Goyal, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

[98] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations.*

[99] Larsen, A. B. L., Sønderby, S. K., Larochelle, H., and Winther, O. (2016). Autoencoding beyond pixels using a learned similarity metric. In *International conference on machine learning*, pages 1558–1566. PMLR.

[100] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436.

[101] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[102] Lee, J., Cho, K., and Hofmann, T. (2017). Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378.

[103] 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.

[104] Li, N., Liu, S., Liu, Y., Zhao, S., and Liu, M. (2019). Neural speech synthesis with Transformer network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6706–6713.

[105] Libovickỳ, J. and Helcl, J. (2018). End-to-end non-autoregressive neural machine translation with connectionist temporal classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3021.

[106] Lim, J. H. and Ye, J. C. (2017). Geometric GAN. *arXiv preprint arXiv:1705.02894*.

[107] Lin, C.-Y. and Hovy, E. (2003). Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 150–157.

[108] Ling, Z.-H., Kang, S.-Y., Zen, H., Senior, A., Schuster, M., Qian, X.-J., Meng, H. M., and Deng, L. (2015). Deep learning for acoustic modeling in parametric speech generation: A systematic review of existing techniques and future trends. *IEEE Signal Processing Magazine*, 32(3):35–52.

[109] Liu, L., Utiyama, M., Finch, A., and Sumita, E. (2016). Neural machine translation with supervised attention. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3093–3102, Osaka, Japan. The COLING 2016 Organizing Committee.

[110] Liu, R., Sisman, B., Li, J., Bao, F., Gao, G., and Li, H. (2020a). Teacher-student training for robust Tacotron-based TTS. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6274–6278. IEEE.

[111] Liu, Y., Gu, J., Goyal, N., Li, X., Edunov, S., Ghazvininejad, M., Lewis, M., and Zettlemoyer, L. (2020b). Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

[112] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2021). RoBERTa: A robustly optimized BERT pretraining approach. *Proceedings of the 20th Chinese National Conference on Computational Linguistics.*

[113] Liu, Z., Xu, Y., Winata, G. I., and Fung, P. (2019). Incorporating word and subword units in unsupervised machine translation using language model rescoring. In *Proceedings of the Fourth Conference on Machine Translation (Volume 2: Shared Task Papers, Day 1)*, pages 275–282.

[114] Lopez, A. (2008). Statistical machine translation. *ACM Computing Surveys (CSUR)*, 40(3):1–49.

[115] Lorenzo-Trueba, J., Drugman, T., Latorre, J., Merritt, T., Putrycz, B., Barra-Chicote, R., Moinet, A., and Aggarwal, V. (2019). Towards achieving robust universal neural vocoding. *Proc. Interspeech 2019*, pages 181–185.

[116] Luong, M.-T., Manning, C. D., et al. (2015a). Stanford neural machine translation systems for spoken language domains. In *Proceedings of the international workshop on spoken language translation.*

[117] Luong, M.-T., Pham, H., and Manning, C. D. (2015b). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing.*

[118] Mavandadi, S., Sainath, T. N., Hu, K., and Wu, Z. (2021). A deliberation-based joint acoustic and text decoder. In *Proc. Interspeech 2021.*

[119] McCarthy, O. (2018). Pytorch implementation of Deepmind's WaveRNN model.

[120] Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. (2017). SampleRNN: An unconditional end-to-end neural audio generation model. *5th International Conference on Learning Representations, ICLR.*

[121] Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., et al. (2018). Mixed precision training. In *International Conference on Learning Representations.*

[122] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[123] Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362.

[124] Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619.*

[125] Nose, T. and Kobayashi, T. (2013). An intuitive style control technique in HMM-based expressive speech synthesis using subjective style intensity and multiple-regression global variance model. *Speech Communication*, 55(2):347–357.

[126] Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G., Lockhart, E., Cobo, L., Stimberg, F., et al. (2018). Parallel WaveNet: Fast high-fidelity speech synthesis. In *International conference on machine learning*, pages 3918–3926. PMLR.

[127] Ott, M., Edunov, S., Grangier, D., and Auli, M. (2018). Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9.

[128] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S., and Lakshminarayanan, B. (2021). Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64.

[129] 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 on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

[130] Parikh, A., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255.

[131] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318.

[132] Passban, P., Liu, Q., and Way, A. (2018). Improving character-based decoding using target-side morphological information for neural machine 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 58–68.

[133] Paul, D., Pantazis, Y., and Stylianou, Y. (2020). Speaker conditional WaveRNN: Towards universal neural vocoder for unseen speaker and recording conditions. *Proc. Interspeech 2020*, pages 235–239.

[134] Paulus, R., Xiong, C., and Socher, R. (2018). A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.

[135] Peng, K., Ping, W., Song, Z., and Zhao, K. (2020). Non-autoregressive neural text-to-speech. In *International conference on machine learning*, pages 7586–7598. PMLR.

[136] Pereyra, G., Tucker, G., Chorowski, J., Kaiser, L., and Hinton, G. E. (2017). Regularizing neural networks by penalizing confident output distributions. *CoRR*, abs/1701.06548.

[137] Ping, W., Peng, K., and Chen, J. (2018a). ClariNet: Parallel wave generation in end-to-end text-to-speech. In *International Conference on Learning Representations*.

[138] Ping, W., Peng, K., Gibiansky, A., Arik, S. Ö., Kannan, A., Narang, S., Raiman, J., and Miller, J. (2018b). Deep voice 3: Scaling text-to-speech with convolutional sequence learning. In *ICLR (Poster)*.

[139] Ping, W., Peng, K., Zhao, K., and Song, Z. (2020). WaveFlow: A compact flow-based model for raw audio. In *International Conference on Machine Learning*, pages 7706–7716. PMLR.

[140] Popov, V., Kudinov, M., and Sadekova, T. (2020). Gaussian LPCNet for multisample speech synthesis. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6204–6208. IEEE.

[141] Post, M. (2018). A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191.

[142] Prabhavalkar, R., Sainath, T. N., Wu, Y., Nguyen, P., Chen, Z., Chiu, C.-C., and Kannan, A. (2018). Minimum word error rate training for attention-based sequence-to-sequence models. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4839–4843. IEEE.

[143] Prenger, R., Valle, R., and Catanzaro, B. (2019). WaveGlow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE.

[144] Provilkov, I., Emelianenko, D., and Voita, E. (2020). BPE-dropout: Simple and effective subword regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892.

[145] Qiu, J., Ma, H., Levy, O., Yih, W.-t., Wang, S., and Tang, J. (2020a). Blockwise self-attention for long document understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2555–2565.

[146] Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., and Huang, X. (2020b). Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, pages 1–26.

[147] Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training. *OpenAI Blog.* at https://openai.com/blog/language-unsupervised/.

[148] Ranzato, M., Chopra, S., Auli, M., and Zaremba, W. (2016). Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR.*

[149] Ren, Y., Hu, C., Qin, T., Zhao, S., Zhao, Z., and Liu, T.-Y. (2021). Fastspeech 2: Fast and high-quality end-to-end text-to-speech. *9th International Conference on Learning Representations, ICLR.*

[150] Ren, Y., Ruan, Y., Tan, X., Qin, T., Zhao, S., Zhao, Z., and Liu, T.-Y. (2019). Fastspeech: Fast, robust and controllable text to speech. In *Advances in Neural Information Processing Systems*, pages 3171–3180.

[151] Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR.

[152] Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modelling. *Computer Speech & Language*, 3(10):187–228.

[153] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[154] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

[155] Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. (2017). PixelCNN++: Improving the pixelCNN with discretized logistic mixture likelihood and other modifications. *5th International Conference on Learning Representations, ICLR*.

[156] Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.

[157] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

[158] Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerrv-Ryan, R., et al. (2018). Natural TTS synthesis by conditioning WaveNet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783. IEEE.

[159] 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.

[160] Shen, T., Ott, M., Auli, M., and Ranzato, M. (2019). Mixture models for diverse machine translation: Tricks of the trade. In *ICML*.

[161] 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.

[162] Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *ICLR*.

[163] Skerry-Ryan, R., Battenberg, E., Xiao, Y., Wang, Y., Stanton, D., Shor, J., Weiss, R., Clark, R., and Saurous, R. A. (2018). Towards end-to-end prosody transfer for expressive speech synthesis with Tacotron. In *international conference on machine learning*, pages 4693–4702. PMLR.

[164] Sotelo, J., Mehri, S., Kumar, K., Santos, J. F., Kastner, K., Courville, A. C., and Bengio, Y. (2017). Char2wav: End-to-end speech synthesis. In *5th International Conference on Learning Representations, ICLR (Workshop).*

[165] 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.

[166] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Training very deep networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2.*

[167] Sun, H., Tan, X., Gan, J.-W., Liu, H., Zhao, S., Qin, T., and Liu, T.-Y. (2019). Token-level ensemble distillation for grapheme-to-phoneme conversion. *Proc. Interspeech 2019*, pages 2115–2119.

[168] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[169] Tachibana, H., Uenoyama, K., and Aihara, S. (2018). Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4784–4788. IEEE.

[170] Tan, X., Qin, T., Soong, F., and Liu, T.-Y. (2021). A survey on neural speech synthesis. *arXiv preprint arXiv:2106.15561.*

[171] Tan, Z., Wang, S., Yang, Z., Chen, G., Huang, X., Sun, M., and Liu, Y. (2020). Neural machine translation: A review of methods, resources, and tools. *AI Open*, 1:5–21.

[172] Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2020). Efficient Transformers: A survey. *arXiv preprint arXiv:2009.06732.*

[173] Valin, J.-M. and Skoglund, J. (2019). LPCNet: Improving neural speech synthesis through linear prediction. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5891–5895. IEEE.

[174] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). WaveNet: A generative model for raw audio. *CoRR abs/1609.03499.*

[175] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems.*

[176] Vig, J. and Belinkov, Y. (2019). Analyzing the structure of attention in a Transformer language model. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 63–76.

[177] Vipperla, R., Park, S., Choo, K., Ishtiaq, S., Min, K., Bhattacharya, S., Mehrotra, A., Ramos, A. G. C., and Lane, N. D. (2020). Bunched LPCNet: Vocoder for low-cost neural text-to-speech systems. *Proc. Interspeech 2020*, pages 3565–3569.

[178] Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808.

[179] Wagner, P., Beskow, J., Betz, S., Edlund, J., Gustafson, J., Eje Henter, G., Le Maguer, S., Malisz, Z., Székely, É., Tånnander, C., et al. (2019). Speech synthesis evaluation—state-of-the-art assessment and suggestion for a novel research program. In *Proceedings of the 10th Speech Synthesis Workshop (SSW10)*.

[180] Wang, C., Zhang, J., and Chen, H. (2018a). Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 479–488.

[181] Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *Deep learning reviews*.

[182] Wang, Y., Skerry-Ryan, R. J., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q. V., Agiomyrgiannakis, Y., Clark, R. A. J., and Saurous, R. A. (2017). Tacotron: Towards end-to-end speech synthesis. In *INTERSPEECH*.

[183] Wang, Y., Stanton, D., Zhang, Y., Ryan, R.-S., Battenberg, E., Shor, J., Xiao, Y., Jia, Y., Ren, F., and Saurous, R. A. (2018b). Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. In *International Conference on Machine Learning*, pages 5180–5189. PMLR.

[184] Weiss, R. J., Skerry-Ryan, R., Battenberg, E., Mariooryad, S., and Kingma, D. P. (2021). Wave-Tacotron: Spectrogram-free end-to-end text-to-speech synthesis. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5679–5683. IEEE.

[185] Wu, L., Xia, Y., Tian, F., Zhao, L., Qin, T., Lai, J., and Liu, T.-Y. (2018). Adversarial neural machine translation. In *Asian Conference on Machine Learning*, pages 534–549. PMLR.

[186] 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. *arXiv preprint arXiv:1609.08144*.

[187] 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. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1782–1792.

[188] Xu, J., Zhou, H., Gan, C., Zheng, Z., and Li, L. (2021). Vocabulary learning via optimal transport for neural machine translation. *Entropy*, 27(28.0):28–5.

[189] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR.

[190] Yamamoto, R., Song, E., and Kim, J.-M. (2020). Parallel waveGAN: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6199–6203. IEEE.

[191] Yang, J. and He, L. (2020). Towards universal text-to-speech. *Proc. Interspeech 2020*, pages 3171–3175.

[192] Yang, S., Wang, Y., and Chu, X. (2020). A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv:2002.07526*.

[193] 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.

[194] Yoshimura, T., Tokuda, K., Masuko, T., Kobayashi, T., and Kitamura, T. (1999). Simultaneous modeling of spectrum, pitch and duration in HMM-based speech synthesis. In *Sixth European Conference on Speech Communication and Technology*.

[195] Yu, C., Lu, H., Hu, N., Yu, M., Weng, C., Xu, K., Liu, P., Tuo, D., Kang, S., Lei, G., et al. (2020). DurIAN: Duration informed attention network for multimodal synthesis. *Proc. Interspeech 2020*.

[196] Yu, F. and Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. In *ICLR (Poster)*.

[197] Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017a). SeqGAN: sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 2852–2858.

[198] Yu, Y., Choi, J., Kim, Y., Yoo, K., Lee, S.-H., and Kim, G. (2017b). Supervising neural attention models for video captioning by human gaze data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 490–498.

[199] Zagoruyko, S. and Komodakis, N. (2017). Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *5th International Conference on Learning Representations, ICLR*.

[200] Zen, H. and Sak, H. (2015). Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4470–4474. IEEE.

[201] Zen, H. and Senior, A. (2014). Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 3844–3848. IEEE.

[202] Zen, H., Senior, A., and Schuster, M. (2013). Statistical parametric speech synthesis using deep neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 7962–7966. IEEE.

[203] Zen, H., Tokuda, K., and Black, A. W. (2009). Statistical parametric speech synthesis. *speech communication*, 51(11):1039–1064.

[204] Zhang, W., Feng, Y., Meng, F., You, D., and Liu, Q. (2019). Bridging the gap between training and inference for neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4334–4343.

[205] Zhang, Z., Liu, S., Li, M., Zhou, M., and Chen, E. (2018). Bidirectional generative adversarial networks for neural machine translation. In *Proceedings of the 22nd conference on computational natural language learning*, pages 190–199.

Appendix A

Supplementary Material on Deep Learning

A.1 Cells for Recurrent Neural Networks

A.1.1 Gated Recurrent Units

Figure A.1 illustrates the structure of vanilla RNN, focusing on the activation function f. Figure A.2 illustrates the structure of Gated Recurrent Units (GRU), which uses gating to improve the network's memory. Gating can be viewed as an extension to activation function. The standard form is sigmoid function, in which case a gating function outputs a vector that acts as a probabilistic gate on network values. Each element of the vector is between 0 and 1. Intuitively, 0 means to remember nothing, and 1 means to remember everything. GRU can be formulated as equations A.1 to A.4.

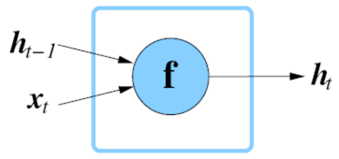

Fig. A.1 Vanilla recurrent unit [48]

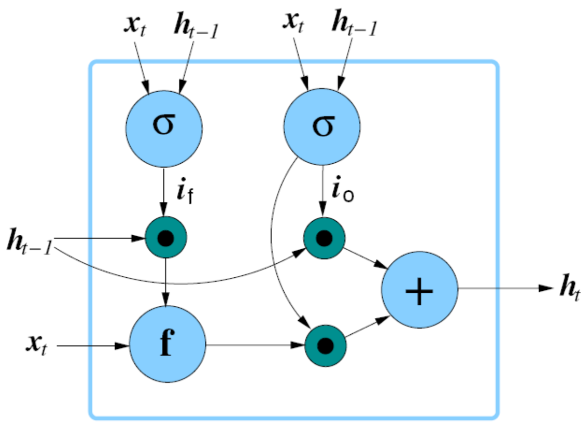

Fig. A.2 Gated recurrent unit [48]

$$\boldsymbol{i}_f = \sigma(\boldsymbol{W}_f^f \boldsymbol{x}_t + \boldsymbol{W}_f^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \tag{A.1}$$

$$\boldsymbol{i}_o = \sigma(\boldsymbol{W}_o^f \boldsymbol{x}_t + \boldsymbol{W}_o^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \tag{A.2}$$

$$\boldsymbol{y}_t = f(\boldsymbol{W}_y^f \boldsymbol{x}_t + \boldsymbol{W}_y^r (\boldsymbol{i}_f \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b}_y) \tag{A.3}$$

$$\boldsymbol{h}_t = \boldsymbol{i}_o \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{i}_o \odot \boldsymbol{y}_t) \tag{A.4}$$

σ denotes sigmoid function; $\odot$ denotes element-wise multiplication; $\boldsymbol{i}_f$ and $\boldsymbol{i}_o$ denote forget and output gates. Suppose the output gate $\boldsymbol{i}_o$ has the value 1 for all dimensions, then there is a direct link between $\boldsymbol{h}_t$ and $\boldsymbol{h}_{t-1}$. In contrast, for vanilla RNN the activation function f is always between $\boldsymbol{h}_t$ and $\boldsymbol{h}_{t-1}$.

A.1.2 Long Short-Term Memory

A Long Short-Term Memory (LSTM) network uses special hidden units, the natural behavior of which is to remember inputs for a long time, to augment the network with an explicit memory [67]. A special unit called the memory cell acts like an accumulator or a gated leaky neuron: it has a connection to itself at the next time step that has a weight of one, so it copies its own real-valued state and accumulates the external signal, but this self-connection is multiplicatively gated by another unit that learns to decide when to clear the content of the memory [100]. Figure A.3 illustrates the structure of LSTM, which can be formulated as equations A.5 to A.9.

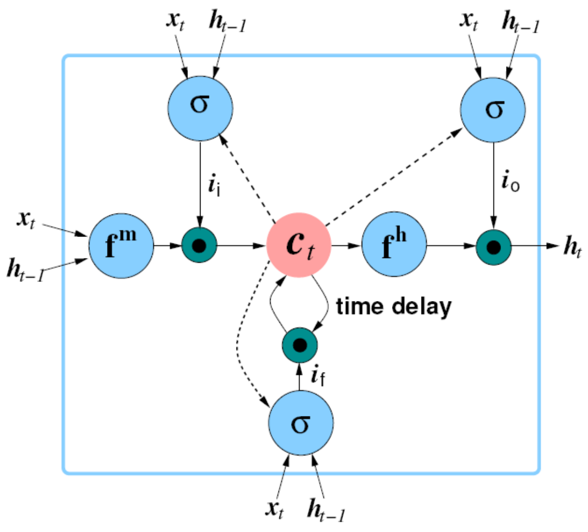

Fig. A.3 Long short-term memory [48]

$$\boldsymbol{i}_f = \sigma(\boldsymbol{W}_f^f \boldsymbol{x}_t + \boldsymbol{W}_f^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_f + \boldsymbol{W}_f^m \boldsymbol{c}_{t-1}) \tag{A.5}$$

$$\boldsymbol{i}_i = \sigma(\boldsymbol{W}_i^f \boldsymbol{x}_t + \boldsymbol{W}_i^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_i + \boldsymbol{W}_i^m \boldsymbol{c}_{t-1}) \tag{A.6}$$

$$\boldsymbol{i}_o = \sigma(\boldsymbol{W}_o^f \boldsymbol{x}_t + \boldsymbol{W}_o^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_o + \boldsymbol{W}_o^m \boldsymbol{c}_t) \tag{A.7}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_f \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_i \odot f^m(\boldsymbol{W}_c^f \boldsymbol{x}_t + \boldsymbol{W}_c^r \boldsymbol{h}_{t-1} + \boldsymbol{b}_c) \tag{A.8}$$

$$\boldsymbol{h}_t = \boldsymbol{i}_o \odot f^h(\boldsymbol{c}_t) \tag{A.9}$$

$\boldsymbol{c}_t$ denotes the memory cell; $\boldsymbol{i}_f, \boldsymbol{i}_i, \boldsymbol{i}_o$ denote the forget, input and output gates. Suppose the forget gate $\boldsymbol{i}_o$ has the value 1 for all dimensions, and the input gate $\boldsymbol{i}_i$ has the value 0 for all dimensions, the current memory cell will be exactly the same as the one at the previous time step, which helps memorizing history.

Appendix B

Supplementary Material on Text-to-speech Synthesis

B.1 Additional Experiments

Section 6.3.5 proposed a pretraining approach for neural vocoders, which leverages large amounts of unlabeled data. The idea is to use a conventional vocoder to extract features from large amounts of unlabeled speech, and then train the the neural vocoder to map the features back to speech. The section describes the corresponding experiments.

B.1.1 Data

The experiments are performed on two datasets with different size, namely Nick and Nancy. Nick dataset contains 2396 utterances from a male British speaker; each utterance is about 2 seconds so there is about 3 hours' speech in total. Nancy dataset contains 12095 utterances from a female American speaker; each utterance is about 5 seconds so there is about 15 hours' speech in total. Regardless of the dataset, 50 utterances are for validation, another 50 utterances are for testing, and the rest utterances are for training.

For waveforms, the sampling frequency is 16kHz, and the samples are quantized into 256 integer values. For conditioning vectors, both linguistic features and vocoder features have a frequency of 200Hz; when necessary they are upsampled using linear interpolation. The linguistic features are 601-dimensional vectors extracted from text;

the first 592 dimensions are binary and the other dimensions are continuous. The vocoder features are 163-dimensional vectors; all dimensions are continuous. Depending on the goal of the experiment, either reference vocoder features or generated vocoder features are used. The reference vocoder features are extracted from waveforms with a PML vocoder. The generated vocoder features are from a conventional acoustic model. As described in section 6.1.2, this model only handles aligned input and output. Here the linguistic features are force-aligned with the corresponding waveform.

B.1.2 Performance Metrics

In this work, both objective and subjective performance metrics are used. When developing a speech synthesis system, we evaluate it with objective metrics, which are free and fast to use. When the system has a reasonable quality, we evaluate it with subjective listening tests. Each listening test compares two systems, and is taken by more than 30 workers from Amazon Mechanical Turk. Participants are instructed to listen to pairs of sentences, and indicate which one they prefer in terms of overall quality. Each comparison includes 5 pairs of utterances randomly selected among all the test utterances.

For objective metrics, vocoder features are extracted from reference and generated waveforms, and root-mean-square error (RMSE) is computed between the feature trajectories. Since the natural duration is always used, the trajectories are synchronized. For PML vocoder, the dimensions of a vocoder feature vector can be separated into three streams: noise mask (NM), MCEP and log(F0). RMSE are computed separately for each stream. Within each stream, the RMSE of all dimensions are added. Although the energy level for each dimension is different, adding them up is sensible because the dimensions with higher energy are more important for the quality of generated speech.

B.1.3 Neural Vocoder Pretraining

Experimental Setup

The data and performance metrics are described in the previous sections. A conventional acoustic model is used to map a linguistic feature sequence to a vocoder feature sequence. $\theta_{\texttt{BLSTM}}$ denotes its parameters. The model has three bidirectional LSTM (BLSTM) layers, and the dimension of each layer is 1024 for Nick dataset, and 512 for Nancy dataset. The acoustic model is trained with linguistic features and reference acoustic features extracted from waveform. After training, this model can be used to generate vocoder features for all utterances of the two datasets, and these generated vocoder features can be used as conditioning vectors for neural vocoders.

For waveform-level synthesis, the model is based on SampleRNN, also referred to as hierarchical RNN (HRNN), and is described in section 6.3.3. Here a 4-tier HRNN is used to map a conditioning vector sequence to speech. There are three types of conditioning vectors: linguistic feature, acoustic feature extracted from waveform, and acoustic feature generated from the acoustic model. $\theta_{\texttt{HRNN}}$ denotes the model parameters. Tier 0 is a 4-layer DNN, including three fully connected layers with ReLU activation and a softmax output layer; the dimension is 1024 for the first two fully connected layers, and is 256 for the other two layers. The other tiers are all 2-layer RNNs; GRU is used and the dimension is 1024 for all layers. The frequencies for tiers 0 to 3 are respectively 16000Hz, 3200Hz, 800Hz and 200Hz. The reason to use this HRNN configuration is as follows. In general, using many tiers operating at diverse frequencies leads to good performance. From experience, the effective history length of GRU is about 100 time steps. For tier 3, each time step corresponds to 5ms, so the effective history length is 500ms, which is at word-level. Similarly, for tier 2, the effective history length is 125ms, which is at phoneme-level. For tier 1, the effective history length is 31.25ms, which is at sub-phoneme-level and keeps the generated waveform from being too smooth. For each RNN tier, the number of layers to use should be consistent with its upsampling rate $R^{(k)}$. A tier with higher upsampling rate outputs more supervising vectors, and should have more layers [40]. In the above configuration, $R^{(3)} = R^{(2)} = 4$ and $R^{(1)} = 5$, therefore all RNN tiers have two layers.

During training, SGD is used to minimize the negative log-likelihood. Gradients are hard-clipped to remain in the range $(-1, 1)$. The Adam optimizer [86] with an initial learning rate of 0.001 ($\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e^{-8}$) is used. The initial RNN state of all the RNN-based models is learnable. Weight normalization [156] is used for all the linear layers to accelerate training. Truncated back propagation through time is also used to accelerate training. Orthogonal weight matrices are used for initializing hidden-to-hidden connections and other weight matrices are initialized in a similar way to the way proposed by He et al. [63].

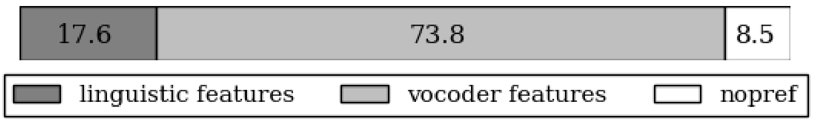

Fig. B.1 Result of the listening test comparing linguistic features and vocoder features; Nick test set

Table B.1 Added RMSE for different conditioning vectors; PML vocoder analysis; Nick test set

Conditioning	MCEP	NM	log(F0)
Acoustic features	7.60	8.15	0.39
Linguistic features	8.14	8.61	0.40

Results and Analysis

Investigation on conditioning vectors As a preliminary experiment, linguistic features and acoustic features are compared as conditioning vectors. Two HRNNs are trained with Nick dataset, using linguistic features and vocoder features respectively. The vocoder features are generated from the acoustic model. It is expected that conditioning on vocoder features yields better performance, because the mapping requires less data and modeling power. Figure B.1 shows the result of the listening test comparing linguistic features and vocoder features. Each number indicates the percentage of participants with a certain preference. It can be seen that most participants prefer conditioning on vocoder features. Table B.1 compares linguistic features and vocoder features using objective metrics. It shows the added RMSE for MCEP, NM and log(F0). The vocoder analysis is performed with a PML vocoder. It can be seen that when conditioning on vocoder features, the model has better performance in all three aspects. This motivates using acoustic features as conditioning vectors in the other experiments in this work.

Investigation on pretraining To investigate the effect of pretraining, three neural vocoders are built: HRNN, HRNNFT10% and HRNNFT100%. The acoustic model $\theta_{\texttt{BLSTM}}$ is shared. It is trained with all the Nick training data. For HRNN, the neural vocoder $\theta_{\texttt{HRNN}}$ is trained with all the Nick training data. The vocoder features used to pretrain $\theta_{\texttt{HRNN}}$ are generated from $\theta_{\texttt{BLSTM}}$. In contrast, HRNNFT10% and HRNNFT100%

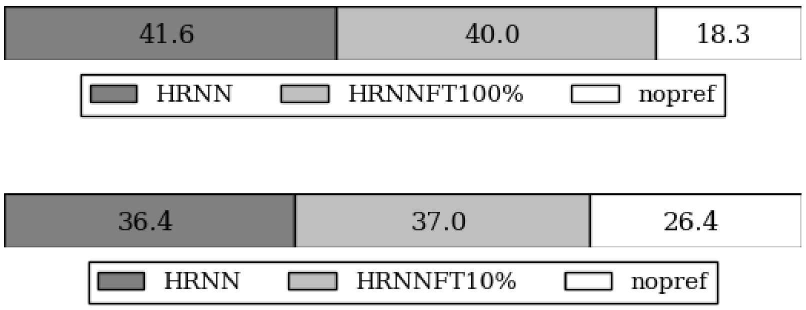

Fig. B.2 Results of the listening tests comparing HRNN, HRNNFT100% and HRNNFT10%; Nick test set

are built as follows. First, the neural vocoder $\theta_{\texttt{HRNN}}$ is pretrained with all the Nancy training data. The vocoder features are extracted with a PML vocoder, so no label is required. Next, the pretrained neural vocoder is finetuned with different amounts of Nick data. Here the vocoder features are generated from the acoustic model $\theta_{\texttt{BLSTM}}$. HRNNFT10% and HRNNFT100% respectively use 10% and 100% of the generated data to finetune the neural vocoder.

As expected, pretraining the neural vocoder is very effective. Figure B.2 shows the results of listening tests comparing the three systems. It can be seen that HRNN and HRNNFT100% have very similar quality. More importantly, 10% of Nick training data, which is about 10 minutes, is enough to finetune the model to be comparable with the counterpart finetuned with 100% of Nick training data. In fact, even when the neural vocoder pretrained with Nancy data (female, American) is not finetuned, it can map a sequence of vocoder features from Nick data (male, British) to speech that sounds like a British male.

B.2 Listening Tests

Section 6.4.2 described the subjective listening tests conducted in this work. The following are some example web pages made for these tests, including a cover page, an AB preference test and a Mean-Opinion-Score test.

Preamble

Participation criteria

Please **check absolutely** that the following criteria are met. **ATTENTION**: If **ANY** of these criteria is not met, your assignment will be rejected.

- You can participate in this test **ONLY ONCE**.
- Only **NATIVE ENGLISH** speakers can take this test (all regional accents are accepted).
- Take the time to **listen!** You have to spend **AT LEAST 4 MINUTES** on this test.
- You **MUST** absolutely use **headphones** or **earphones**

Anonymity

- Your participation in this test is **anonymous**. The data the researchers will work on will not identify you in any manner. If the data of each participant is published for reasons of Open Access the data will still be anonymous. Similarly, any published aggregated results (statistics, histograms, etc.) will not identify any participants.
- In addition to the assessment of the speech, you will be asked for some information to facilitate analysis of the results (mother tongue, age, gender). This data will also be processed anonymously.
- Your data will not be forwarded to any third-party.

Retraction right

- Within **a week**, you have the possibility to withdraw the data you supply. If you want to assert this right, please leave us a message providing your **Assignment ID** (See below) and see the section *Contact* below.
- If you do not withdraw your data within a week, you authorize us to publish the data or statistics of the data aggregated with other participants' data, as detailed above.

Contact

If you are Amazon Mechanical Turk Worker, please use AMTurk messaging service. Otherwise please send us an e-mail. To avoid any possible identification, please avoid mentioning any personnal contact information in your message.

Purpose

The resulting data will be used for the following research projects:

- HQSTS Project
- NST Project

Authors

This listening test is carried out by the Speech Group of The Machine Intelligence Laboratory, Engineering Department, Cambridge University, UK.

Related regulations

This Listening Test complies with EU and UK regulations:

- EU Directive 95/46/EC
- UK Data Protection Act (DPA)

Page loaded on Friday 22nd of October 2021 20:45:14. Test: LJ_TacoWrnn_TF80_AF80_NV
Assignment ID: `C41D1677594114E4FF437E21066F3452A83BCD7C`

Before pressing any button below, please keep a copy of this page (print me!)

I agree.
Show me the test !

I disagree.
Finally, I'll not take this test.

Evaluation of preferences

Please carefully read the following information, even if you are used to doing these type of listening tests.

Instructions

- Do the test in a **quiet place**.
- Verify that the sound level is **loud enough** to hear the sound details properly.
- If there are any **technical problems** with one of the recordings, select "Prob" to indicate a problem.
- If you perceive a **difference of loudness** so that you can't properly evaluate a pair of sounds also select "Prob".
- Do not try to find a pattern among the samples' order. There is none.
- There is no "correct" answer. It is about your subjective preference, but try to be consistent in your choices.

For each pair of recordings below (each line) select one button depending on your **preference** for the two samples in terms of their **overall quality**.

- If the right recording is much better than the left one, select the furthest right button (+3)
- If the right recording is better than the left one, select the second button (+2)
- If the right recording is slightly better than the left one, select the third button (+1)
- If the two recordings sound the same, or, they have differences which are **NOT related to the overall quality**, select the middle button (0)

... and the same on the other way.

If you feel that these instructions lack of clarity, please do not take this listening test.

The test

Pair	File1	-3	-2	-1	0	+1	+2	+3	File2	Prob
1	▶ 0:00 / 0:04	○	○	○	○	○	○	○	▶ 0:00 / 0:04	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
2	▶ 0:00 / 0:05	○	○	○	○	○	○	○	▶ 0:00 / 0:06	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
3	▶ 0:00 / 0:06	○	○	○	○	○	○	○	▶ 0:00 / 0:04	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
4	▶ 0:00 / 0:05	○	○	○	○	○	○	○	▶ 0:00 / 0:05	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
5	▶ 0:00 / 0:02	○	○	○	○	○	○	○	▶ 0:00 / 0:02	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
6	▶ 0:00 / 0:05	○	○	○	○	○	○	○	▶ 0:00 / 0:05	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
7	▶ 0:00 / 0:03	○	○	○	○	○	○	○	▶ 0:00 / 0:03	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
8	▶ 0:00 / 0:04	○	○	○	○	○	○	○	▶ 0:00 / 0:04	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
9	▶ 0:00 / 0:04	○	○	○	○	○	○	○	▶ 0:00 / 0:04	○
Pair	**File1**	**-3**	**-2**	**-1**	**0**	**+1**	**+2**	**+3**	**File2**	**Prob**
10	▶ 0:00 / 0:04	○	○	○	○	○	○	○	▶ 0:00 / 0:04	○

Once finished, you can reassess the comparisons as many times as you want.

Some information about you

What's your mother tongue ? (Select English if English is among your mother tongues)	Please select		
Are you	○ Female	○ Male	○ Other
Your Age			
How did you listen to the sounds ?	○ Headphones	○ Earphones	○ Loudspeakers

Please, check that all pairs are evaluated, then send the answers !

Evaluation of preferences

Please carefully read the following information, even if you are used to doing these type of listening tests.

Instructions

- Do the test in a **quiet place**.
- Verify that the sound level is **loud enough** to hear the sound details properly.
- If there are any **technical problems** with one of the recordings, select "Prob" to indicate a problem.
- If you perceive a **difference of loudness** so that you can't properly evaluate a pair of sounds also select "Prob".
- Do not try to find a pattern among the samples' order. There is none.
- There is no "correct" answer. It is about your subjective preference, but try to be consistent in your choices.

For each recording below (each line) select one button depending on your **evaluation** of their **overall quality**.

- The recordings are grouped by their corresponding text.
- Please feel free to adjust your evaluation after listening to the whole group.

If you feel that these instructions lack of clarity, please do not take this listening test.

The test

	Sounds to assess	Bad(1)	Poor(2)	Fair(3)	Good(4)	Excellent(5)	Prob
1	▶ 0:00 / 0:03	○	○	○	○	○	○
2	▶ 0:00 / 0:03	○	○	○	○	○	○
3	▶ 0:00 / 0:03	○	○	○	○	○	○

	Sounds to assess	Bad(1)	Poor(2)	Fair(3)	Good(4)	Excellent(5)	Prob
4	▶ 0:00 / 0:04	○	○	○	○	○	○
5	▶ 0:00 / 0:04	○	○	○	○	○	○
6	▶ 0:00 / 0:05	○	○	○	○	○	○

	Sounds to assess	Bad(1)	Poor(2)	Fair(3)	Good(4)	Excellent(5)	Prob
7	▶ 0:00 / 0:04	○	○	○	○	○	○
8	▶ 0:00 / 0:04	○	○	○	○	○	○
9	▶ 0:00 / 0:04	○	○	○	○	○	○

	Sounds to assess	Bad(1)	Poor(2)	Fair(3)	Good(4)	Excellent(5)	Prob
10	▶ 0:00 / 0:05	○	○	○	○	○	○
11	▶ 0:00 / 0:07	○	○	○	○	○	○
12	▶ 0:00 / 0:05	○	○	○	○	○	○

	Sounds to assess	Bad(1)	Poor(2)	Fair(3)	Good(4)	Excellent(5)	Prob
13		○	○	○	○	○	○

	▶ 0:00 / 0:06						
14	▶ 0:00 / 0:05	○	○	○	○	○	○
15	▶ 0:00 / 0:07	○	○	○	○	○	○

Once finished, you can reassess the comparisons as many times as you want.

Some information about you

What's your mother tongue ? (Select English if English is among your mother tongues)	Please select		
Are you	○ Female	○ Male	○ Other
Your Age			
How did you listen to the sounds ?	○ Headphones	○ Earphones	○ Loudspeakers

Please, check that all pairs are evaluated, then send the answers !