Waveform Level Synthesis

Qingyun Dou

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Emmanuel College August 2017

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

Declaration

I, Qingyun Dou of Emmanuel College, being a candidate for the M.Phil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. This thesis contains 12817 words.

Signed

Date

Qingyun Dou
August 2017

Acknowledgements

I would like to thank Professor Gales for being patient to me, who had almost no experience with speech synthesis before working on this MPhil project. Thank you for giving me advice, not only on specific questions, but also on how to think. I would also like to thank other faculty members for giving great lectures and demonstrated sessions.

Working on with people in the speech group has been a pleasure. I would like to thank Gilles and Moquan in particular, for their encouragement and advice. Thank you so much for having discussions with me on sometimes very basic questions.

It has been a great experience to work with the MLSALT cohort through out the year. People in this cohort are simply amazing! Thanks for working with me and showing me how talented people can be. You know who I am talking about.

Last but not least, a lot of thanks to my family and friends, including those who are not geographically close to me this year. Your love means every thing to me! As for those who are in town, you make Cambridge great again.

Abstract

This thesis investigates waveform-level synthesis models, which directly generate audio waveforms. In contrast, traditional feature-level synthesis models generate vocoder feature sequences, which are then converted to waveforms. This type of synthesis is limited by several factors, including the quality of the vocoder, the fixed-length analysis window and the lack of expressiveness.

Waveform-level synthesis models is not limited by these factors, as vocoder is not used to generate waveforms. In this thesis, both unconditional synthesis and conditional synthesis are investigated. For unconditional waveform-level synthesis, the major challenge is to model a long history. Two models are investigated: Hierarchical Recurrent Neural Network (HRNN) and Dilated Convolutional Neural Network (DCNN). HRNN models a long history with a stack of RNNs, each operating at a different time scale, while DCNN uses stack of CNNs, each dilated to a different extent. Experiments are performed with HRNN, using both music and speech data. It is found that the model performs well in both cases, and that the structure of the network should be designed according to the time scales of different tiers.

For conditional waveform-level synthesis, the major challenge is to incorporate extra information into the unconditional models. The analysis focuses on adding text information, which is more complicated and more general than other information such as music style. Two approaches are investigated: using standard text labels and using text labels generated by a neural network with attention mechanism. A new conditional synthesis model is developed, combining HRNN and standard standard text labels. Experiments are performed with both the new model and a feature-level synthesis model. The results are analyzed in both time-domain and feature-domain. It is found that the waveform-level synthesis model achieves performance comparable to the feature-level synthesis model, even with very limited tuning, and that having multiple tiers is essential to good performance.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Speech/music synthesis

Systems that can generate natural-sounding speech/music are highly valuable. For example, speech synthesis systems can be applied to give mute people a voice and to read text for blind people. This thesis studies waveform-level synthesis, which can be applied to both music and speech. As speech synthesis is generally more complicated and music can be considered a special type of speech, most analysis takes speech synthesis as example. Depending on the goal, speech synthesis can be categorized into unconditional [15, 25] speech synthesis and conditional speech synthesis [22, 25]. For unconditional speech synthesis, except raw waveform for training no extra information is available. The goal is to generate waveforms that sound like mumbling. For conditional speech synthesis, extra information such as speaker identity and the text to be spoken is available. The goal is to generate meaningful speech corresponding to the given text.

Depending on the approach used, speech synthesis can be categorised into non-parametric speech synthesis [16, 9, 20] and parametric speech synthesis [27, 31, 29, 14]. Non-parametric speech synthesis is also known as concatenative speech synthesis; it generates speech by concatenating units of recorded speech. Parametric speech synthesis is also known as statistical speech synthesis; it generates speech by optimizing a cost function. Although non-parametric speech synthesis is generally more natural, it is less flexible and requires a large corpus to cover the target domain [13]. In contrast, parametric speech synthesis is more flexible. For example, in parametric speech synthesis speaker identity can be changed by changing a small part of the system, whereas in non-parametric speech synthesis the entire system has to be retrained on another corpus.

Parametric speech synthesis comes in two categories: feature-level synthesis [28, 11, 30] and waveform-level synthesis [25, 22]. In feature-level synthesis, a models generates a sequence of vocoder features. A vocoder then converts the sequence of features to a sequence of audio signal points, forming a waveform [13]. Feature-level synthesis is limited by several factors, including the quality of the vocoder, the fixed-length shifting analysis window and the lack of expressiveness caused by deterministic mapping. In contrast, waveform-level synthesis models not limited by these factors: waveforms are not generated by a vocoder; the analysis window can have varying lengths; audio signal points are typically generated in a probabilistic fashion [25, 22].

1.2 Contribution of the thesis

This thesis mainly investigates waveform-level synthesis, and compares it to feature-level synthesis. For unconditional waveform-level synthesis, the two models [25, 15] achieving state-of-the-art performance are investigated, namely Hierarchical Recurrent Neural Network (HRNN) and Dilated Convolutional Neural Network (DCNN). Experiments are performed with HRNN. In addition to the same music data as in the reference paper [15], the model is investigated with the Nick speech data.

For conditional synthesis, two approaches of conditioning are investigated: using standard labels and using labels generated by a neural network with attention mechanism. A new conditional waveform-level synthesis model is developed, combining HRNN with standard labels and achieving performance comparable to feature-level synthesis. Experiments are performed with the new model and a feature-level synthesis model. In addition to analyzing the results in time-domain as the existing research [25, 22], the generated audio samples are analyzed in feature-domain.

1.3 Organization of the thesis

The rest of this thesis is organized as follows. Chapter 2 introduces the general process of feature-level synthesis and waveform-level synthesis. Chapter 3 investigates unconditional waveform-level synthesis, and presents experimental results for unconditional speech and music synthesis. Chapter 4 investigates conditional waveform-level synthesis. Chapter 5 presents experimental results for conditional speech synthesis. Chapter 6 concludes the thesis and presents ideas about future work.

Chapter 2

Background

2.1 Feature-level synthesis

2.1.1 Standard pipeline

The standard pipeline of feature-level synthesis is as follows. In the first step, a sequence of vocoder features $\mathbf{v}_{1:N} = \{v_1, ..., v_N\}$ is extracted from a sequence of audio signal points $x_{1:T} = \{x_1, ..., x_T\}$ forming a waveform, and a sequence of text labels $\mathbf{l}_{1:N}$ is extracted from raw text. Vocoder features are typically extracted at 200Hz, i.e. every 5 milliseconds. For each time step there is a high-dimensional feature vector, which includes mel-cepstrum, fundamental frequency F0 and aperiodicity. Mel-cepstrum represents vocal tract transfer function, and aperiodicity represents characteristics of vocal source excitation signals [25]. In the second step, a set of generative models, such as hidden Markov models (HMMs)[27], feed-forward neural networks (DNNs)[28] and recurrent neural networks (RNNs)[24, 5], is trained by maximizing the probability of the vocoder features given the text labels as a function of model parameters. This can be formulated as in equation (2.1), where θ denotes model parameters. After training, θ will be fixed, and vocoder features can be generated as shown in equation (2.2). Finally, a vocoder maps the features to a waveform consisting of audio signal points; this mapping is highly non-linear and is denoted as $f^{(voc)}$.

$$\hat{\theta} = \underset{\theta}{argmax}\, p(\mathbf{v}_{1:N}|\mathbf{l}_{1:N}, \theta) \tag{2.1}$$

$$\hat{\mathbf{v}}_{1:N} = \underset{\mathbf{v}_{1:N}}{argmax}\, p(\mathbf{v}_{1:N}|\mathbf{l}_{1:N}, \hat{\theta}) \tag{2.2}$$

$$\hat{x}_{1:T} = f^{(voc)}(\hat{\mathbf{v}}_{1:N}) \tag{2.3}$$

2.1.2 Limitations

Feature-level synthesis has several limitations. First, instead of raw waveform data, features are extracted to train the models. This causes some loss of information, which cannot be recovered by the synthesis model. Second, when extracting features, a fixed-length shifting window is used. However, speech units such as phones have very different lengths, and can be much longer or shorter than the window. Moreover, as all audio signal points within a window share the same feature, the generated waveforms is often overly smooth. Finally, the features are typically generated and mapped to waveforms in a deterministic fashion, which makes the generated waveforms less expressive. If two words in a sentence have the same vocoder features, they will be spoken in exactly the same way, which is odd for a real person.

2.2 Waveform-level synthesis

2.2.1 Standard pipeline

For waveform-level synthesis, waveforms are generated without first generating vocoder features. To facilitate comparison to feature-level synthesis, waveform-level synthesis can be formulated as in equations (2.4, 2.5), although in practice some details are different.

$$\hat{\theta} = \underset{\theta}{argmax}\, p(\mathbf{x}_{1:T}|\mathbf{l}_{1:T}, \theta) \tag{2.4}$$

$$\hat{\mathbf{x}}_{1:T} = \underset{\mathbf{x}_{1:T}}{argmax}\, p(\mathbf{x}_{1:T}|\mathbf{l}_{1:T}, \hat{\theta}) \tag{2.5}$$

Typically, to make the model flexible, raw audio data are discretized, and the distribution of an individual audio signal point is modeled as a categorical distribution. At each time step, a neural network with a softmax output layer generates a K-dimensional vector $\mathbf{y}_t$ describing the distribution, where K denotes the number of categories. $0 \leq \mathbf{y}_{tk} \leq 1$, and $\sum_{k=1}^{K} \mathbf{y}_{tk} = 1$.

To train the neural network, the target audio signal point x_t is encoded as a K-dimensional "one-hot" vector $\mathbf{z}_t$[1]. To simplify notation, suppose there is only one utterance $x_{1:T}$, so the target is $\mathbf{z}_{1:T}$, and the neural network generates $\mathbf{y}_{1:T}$. Typically the training criterion is cross-entropy and can be formulated as follows, where $f_{\theta}^{(NN)}$ denotes the neural network and $\mathbf{e}_{1:T}$ denotes its input sequence, which will be introduced with more details in section 2.2.2.

[1]Note that x_t is a scalar, while $\mathbf{y}_t$ and $\mathbf{z}_t$ are vectors. In this thesis vectors are denoted by bold lower case letters, unless otherwise mentioned.

Note that in this case where $\mathbf{z}_t$ is a "one-hot" vector, minimum cross-entropy is equivalent to maximum likelihood.

$$\mathbf{y}_{1:T} = f_{\theta}^{(NN)}(\mathbf{e}_{1:T}) \tag{2.6}$$

$$\hat{\theta} = \underset{\theta}{argmin} E(\theta) = -\sum_{t=1}^{T}\sum_{k=1}^{K} \mathbf{z}_{tk} log(\mathbf{y}_{tk}) \tag{2.7}$$

At synthesis time, the neural network generates the distribution of each audio signal point, and each the actual value of each signal point is sampled from the distribution. This can be formulated as follows, where $Cat(\hat{\mathbf{y}}_t)$ denotes categorical distribution. If the input $\mathbf{e}_t$ includes previously generated signal points, the neural network will be run in an auto-regressive fashion. Note that this makes synthesis different from training, because at training time, the complete sequence of audio signal points $x_{1:T}$ is available and can be used at any time step.

$$\hat{\mathbf{y}}_{1:T} = f_{\hat{\theta}}^{(NN)}(\mathbf{e}_{1:T}) \tag{2.8}$$

$$\hat{x}_t \sim Cat(\hat{\mathbf{y}}_t) \tag{2.9}$$

2.2.2 Advantages and challenges

Waveform-level synthesis models are free of the the limitations mentioned in the previous section. As vocoder is not used to generate waveforms, the quality of the audio generated will not be limited by the quality of the vocoder. In waveform-level synthesis, each audio signal point x_t can have a different context label $\mathbf{l}_t$, which solves the over-smoothing problem. Moreover, waveform-level synthesis models are typically probabilistic as shown in equation (2.9), hence even the same words might be spoken differently, as a real person would do.

For unconditional waveform-level synthesis, the input at each time step $\mathbf{e}_t$ includes previous audio signal points $x_{t-L:t-1}$, where L denotes the length of history. The major challenge is to model a long history: how $\mathbf{y}_t$ should depend on $x_{t-L:t-1}$. In the following chapters, two models will be investigated: Hierarchical Recurrent Neural Network (HRNN) and Dilated Convolutional Neural Network (DCNN). HRNN models a long history with a stack of RNNs, each operating at a different time scale [15], while DCNN uses stack of CNNs, each dilated to a different extent [25].

For conditional waveform-level synthesis, the input at each time step $\mathbf{e}_t$ can include both previous audio signal points $x_{t-L:t-1}$ and extra information $\mathbf{l}_t$ such as text label. The major challenge is to incorporate extra information into the unconditional models. The analysis focuses on adding text information, because it is more complicated and more general than other types of information such as music style and speaker identity. In the following chapters, two approaches will be investigated: using standard text labels [25] and using text labels generated by a neural network with attention mechanism [22]. In addition, a new conditional waveform-level synthesis model will be developed, combining HRNN and standard standard text labels.

Chapter 3

Unconditional synthesis

3.1 Target representation

In unconditional waveform-level synthesis, audio waveforms are modeled without extra information. The joint probability of a waveform $x_{1:T} = \{x_1, ..., x_T\}$ can be factorized as a product of conditional probabilities [25], as shown in equation (3.1). To model the distribution $p(x_t|x_1, ..., x_{t-1})$ for an individual audio signal point, there are mainly two approaches. The first approach is to use a mixture model such as density network [2] and conditional Gaussian scale mixtures [23]. The second approach is to use a categorical distribution [18].

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t|x_1, ..., x_{t-1}) \tag{3.1}$$

3.1.1 Categorical distribution

In this thesis, the approach using categorical distribution is adopted. One reason is that categorical distribution is generally more flexible than mixture models. Although in theory, a mixture model can describe any distribution if it has an infinite number of components, in practice the number of components for a mixture model is limited by the number of data available. Previous experiments have shown that categorical distribution yields better performance than Gaussian or Gaussian mixture distribution [25, 15].

Another reason for adopting the second approach is that standard mixture models tend to result in distributions that are overly smooth for $p(x_t|x_1, ..., x_{t-1})$. Given a long history, the distribution of the next audio signal point is typically very sharp. Figure 3.1 shows an example distribution $p_{eg}(x_t|x_1, ..., x_{t-1})$ generated by a well performing synthesis system. After normalization, each signal point takes a value between -1 and 1. The blue curve shows the learned distribution, and the orange cross shows the actual position of the next point (its probability is 1). As analyzed previously, this distribution is sharp and can be better modeled by a categorical distribution. For this thesis, the categorical distribution used typically has 256 categories, which has been empirically shown sensible [25, 15].

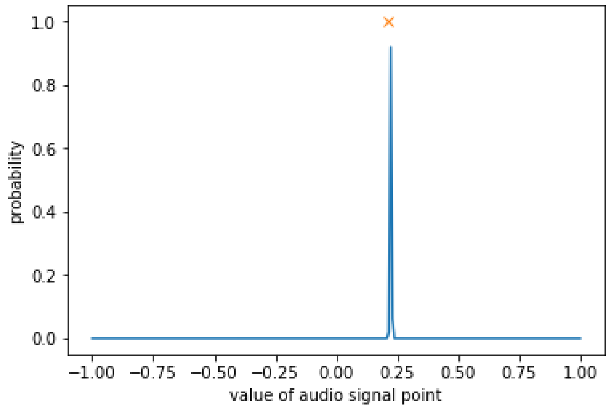


Fig. 3.1 example distribution over an individual signal point

3.1.2 Data quantization

While categorical distribution is discrete, audio is inherently continuous[1]. To apply categorical distribution to audio signal points, raw audio data should be quantized. The simplest approach is to directly use linear quantization. An alternative approach is to first apply a μ-law transformation to the data, as shown in equation (3.2), where $\mu = 255$, and then use linear quantization. Correspondingly, for the first approach, it can be considered that the transformation $f(x_t) = x_t$ is applied before linear quantization.

$$f(x_t) = sign(x_t)\frac{ln(1+\mu|x_t|)}{ln(1+\mu)} \tag{3.2}$$

In general, μ-law transformation reduces the dynamic range of audio data, and hence reduces the quantization error. Figure 3.2 shows the effect of μ-law transformation. Without loss of generality, it can be assumed that all data are linearly normalized to the range $(-1,1)$. At the range where x_t is close to 0, μ-law transformation will result in higher resolution. A small change in x_t can lead to a big change in $f(x_t)$. For audio data, the number of signal points close to 0 is typically higher than the number of signal points far away from to 0, as shown in figure 3.3, therefore using μ-law transformation is sensible. In this thesis, both data quantization approaches are investigated.

[1]Typically, raw audio is stored as a sequence of 16-bit integer values. At each time step, there are 65536 probabilities, which is significantly more than the number of categories. Hence raw audio data can be considered continuous.

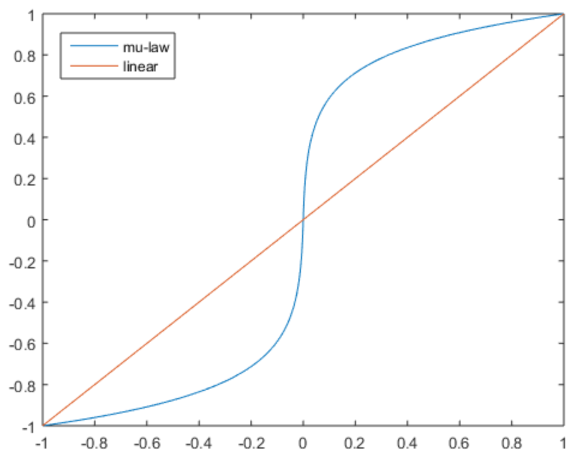


Fig. 3.2 effect of μ-law transformation

3.2 History representation

History representation is very important for synthesis systems, because the audio to be generated depends largely on history (previous input and/or output). Recurrent Neural Network (RNN) is typically used for history representation [6, 15], and can be formulated as follows:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t|x_1,...,x_{t-1}) \tag{3.3}$$

$$p(x_{t+1}|x_1,...,x_t) = f^{(DNN)}(\mathbf{h}_t) \tag{3.4}$$

$$\mathbf{h}_t = f^{(RNN)}(\mathbf{h}_{t-1}, x_t) \tag{3.5}$$

$f^{(DNN)}$ is a feed-forward neural network, and $f^{(RNN)}$ is one of the known RNN memory cells, such as Gated Recurrent Unit (GRU) [3], Long Short Term Memory Unit (LSTM) [8], or their variants [15]. The configurations of vanilla recurrent unit, GRU and LSTM are shown in appendix A.

Compared to vanilla RNNs, RNNs using LSTM or GRU can model longer history. However, due to the vanishing gradient problem [19], even when using LSTM or GRU, RNNs can only model history up to a few hundreds time steps. For feature-level synthesis, a few hundreds time steps can be enough, whereas for waveform-level synthesis, it is far from enough. Even at a relatively low sample rate of 16kHz, there are on average about 6000 audio signal points for each word [15]. Figure 3.3 shows the waveform corresponding to one recorded sentence, "we dress to suit the weather of most days", at two time scales. It can be seen the even for a 30ms range there are hundreds of points. When generating the word "days", the complete history is enormously long. Even if the complete history is truncated and only the last word is left, traditional RNNs will not be able to model the history that is long enough for waveform-level synthesis. To deal with this problem, two models are proposed, namely HRNN and DCNN. Both models will be investigated in the following sections.

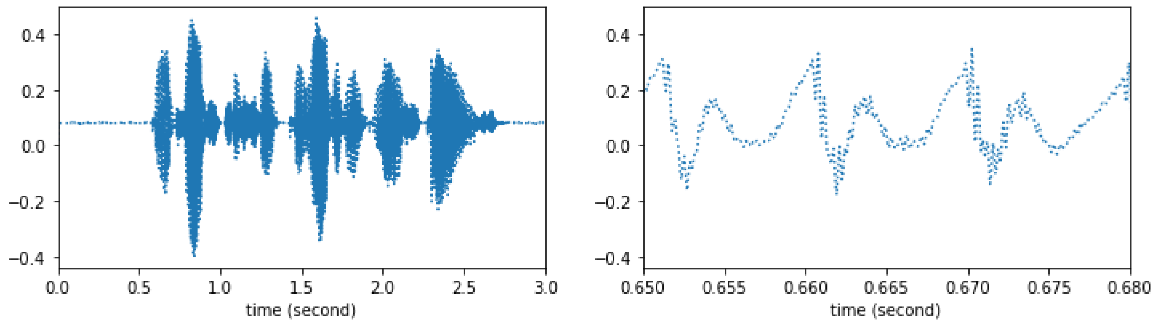


Fig. 3.3 example audio

3.2.1 Hierarchical RNN

HRNN takes into account the fact that audio signal points contain structures at different time scales: dependencies exist not only between neighboring signal points, but also between signal points that are thousands of steps apart [15]. It addresses the challenge of modeling long history by using a hierarchy of tiers, each operating at a different frequency. Figure 3.4 illustrates the structure of HRNN; a 3-tier model is shown but the configuration can be tuned to suit different tasks. The lowest tier processes individual audio signal points and operates on the highest frequency, which is 16000Hz for this thesis. Each higher tier operates on a longer timescale and hence a lower frequency. Each tier conditions the tier below it, and the lowest tier outputs waveform-level predictions.

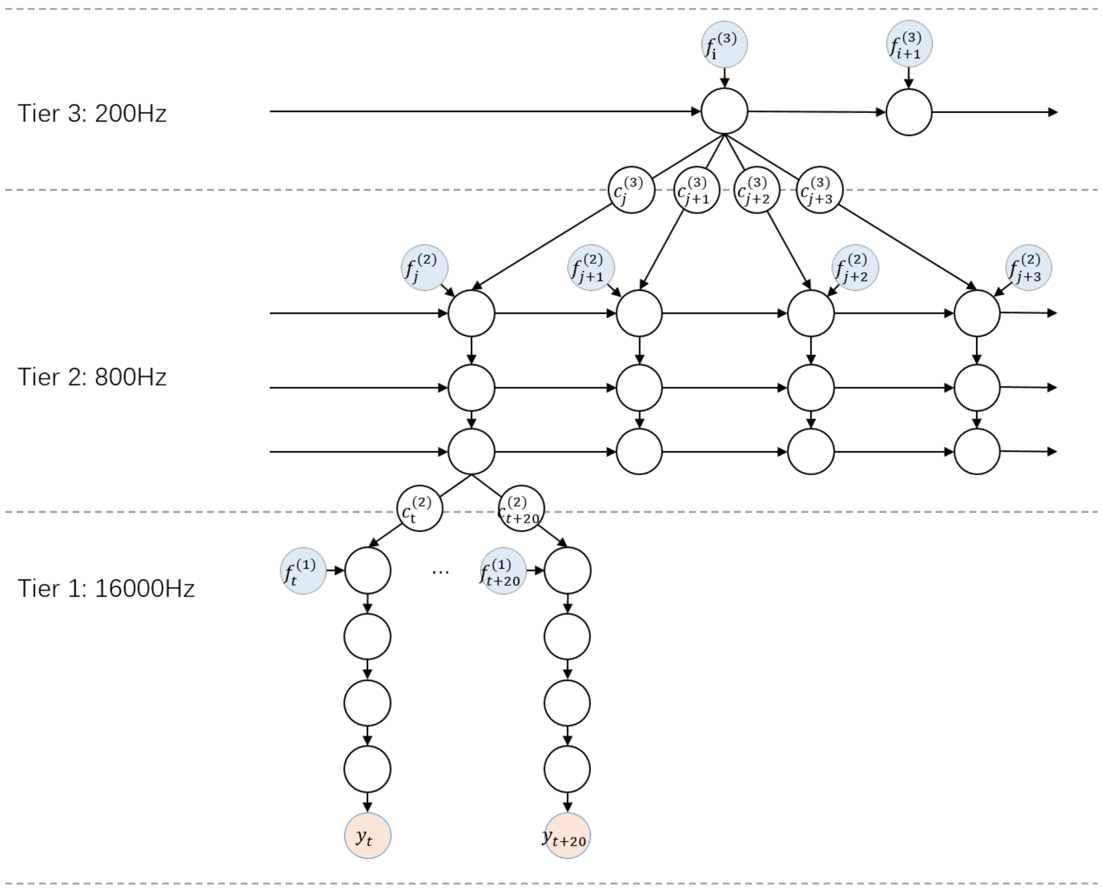


Fig. 3.4 visualization of a hierarchical RNN with 3 tiers.

Tier(s) operating below waveform-level frequency

Except for the lowest-level tier, all higher-level tiers operate on frequencies below waveform-level frequency. These tiers are (deep) RNNs operating on non-overlapping frames of several audio signal points, hence these tiers are sometimes referred to as "frame-level tiers". Each frame-level tier summarizes the history of its inputs into a conditioning vector for the next tier downward. Tiers operating below waveform-level frequency can be formulated as shown in equations (3.6, 3.7, 3.8)[2].

[2]$\mathbf{W}_f$ and $\mathbf{W}_q$ are matrices. In this thesis, by default matrices are denoted by upper case bold letters

$$\mathbf{e}_{t^{(k)}} = \begin{cases} \mathbf{W}_f \mathbf{f}^{(k)}_{t^{(k)}} + \mathbf{c}^{(k+1)}_{t^{(k)}}; & 1 < k < K \\ \mathbf{f}^{(k)}_{t^{(k)}}; & k = K \end{cases} \tag{3.6}$$

$$\mathbf{h}_{t^{(k)}} = f^{(RNN)}(\mathbf{h}_{t^{(k)}-1}, \mathbf{e}_{t^{(k)}}) \tag{3.7}$$

$$\mathbf{c}^{(k)}_{t^{(k-1)}+q} = \mathbf{W}_q \mathbf{h}_{t^{(k)}}; 1 \leq q \leq r^{(k)} \tag{3.8}$$

K is the number of tiers and k the tier index. To increase readability, unless necessary the superscript (k) is not shown. At tier k, each frame $\mathbf{f}^{(k)}_{t^{(k)}}$ includes $FS^{(k)}$ previous signal points $\{x_{t^{(k)}-FS^{(k)}}, ..., x_{t^{(k)}-1}\}$, where FS stands for frame size. The complete history, which has a growing length, is approximated by a fixed-length history vector $\mathbf{h}^{(k)}_{t^{(k)}}$. The RNN makes a history update at time step $t^{(k)}$ as a function of the previous history $\mathbf{h}^{(k)}_{t^{(k)}-1}$ and the current input $\mathbf{e}^{(k)}_{t^{(k)}}$, as shown in equation (3.7). For the top tier, $k = K$, and $\mathbf{e}^{(k)}_{t^{(k)}}$ is only the current input frame. For intermediate tiers, $1 < k < K$, and it is a linear combination of the current input frame $\mathbf{f}^{(k)}_{t^{(k)}}$ and a conditioning vector $\mathbf{c}^{(k+1)}_{t^{(k)}}$ from the next tier upward.

Note that for each tier, time step $t^{(k)}$ is related to a different frequency. Each time step $t^{(k)}$ at tier k corresponds to $r^{(k)}$ time steps in tier $k-1$. In figure 3.4, to increase readability, $t^{(1)}$ is denoted as t, $t^{(2)}$ denoted as j and $t^{(3)}$ denoted as i. Table 3.1 illustrates how time steps at different tiers are related, taking the configuration shown in figure 3.4 as example. In this configuration, $FS^{(3)} = 80, FS^{(2)} = 20, r^{(3)} = 4, r^{(2)} = 20$.

Table 3.1 relation of time steps at different tiers

	time step			
tier 3	1			
tier 2	1	2	3	4
tier 1	1 | ... | 20	21 | ... | 40	41 | ... | 60	61 | ... | 80

To condition the next tier downward, a history vector should be upsampled. Each history vector should be upsampled into a series of $r^{(k)}$ conditioning vectors, where $r^{(k)}$ is the ratio between the frame sizes of the two tiers. If $k = 2$, $r^{(k)} = FS^{(k)}$; otherwise $r^{(k)} = FS^{(k)}/FS^{(k-1)}$. By default, this upsampling is realized by a set of $r^{(k)}$ different linear mappings, although other upsampling methods can also be sensible. The approach of upsampling with $r^{(k)}$ linear projections is equivalent to to upsampling by adding zeros and then applying a linear convolution, which is sometimes called "perforated" upsampling in the context of CNNs. It has been demonstrated to work well [4] and is a fairly common upsampling technique [15].

Tier operating at waveform-level frequency

Unlike other tiers, the tier at the lowest level is a feed-forward DNN. This tier operates at the same frequency as the audio sampling frequency, and is sometimes referred to as "sample-level tier". This tier can be formulated as shown in equations (3.9, 3.10, 3.11).

$$\mathbf{e}_t^{(1)} = \mathbf{W}_f^{(1)}\mathbf{f}_t^{(1)} + \mathbf{c}_t^{(2)} \tag{3.9}$$

$$\mathbf{y}_t^{(1)} = f^{(DNN)}(\mathbf{e}_t^{(1)}) \tag{3.10}$$

$$x_t \sim Cat(\mathbf{y}_t) \tag{3.11}$$

At each time step, the input is a linear combination of the conditioning vector from the second tier and a vector including $FS^{(1)}$ previous audio signal points. With both the conditioning vector and previous signal points as input, the model considers both long-term dependency and short-term dependency when making predictions. Previous experiments have shown that if the prediction of a signal point only depends on the conditioning vector, i.e. equation (3.9) becomes $\mathbf{e}_t^{(1)} = \mathbf{c}_t^{(2)}$, the performance of the synthesis system will decrease considerably [15].

The output at each time step is a vector corresponding to a categorical distribution, each dimension showing the probability of one category. Each audio signal point is sampled from the corresponding categorical distribution. The reason to use DNN instead of RNN is that $FS^{(1)}$ is typically small, and a DNN is capable of modeling the dependency among a small number of signal points. In this thesis the DNN is a Multilayer Perceptron (MLP) with a softmax output layer.

During training, the DNN is convolved over the sequence of audio signal points, processing each window of $FS^{(1)}$ signal points and predicting the next signal point. During generation, the complete sequence of audio signal points is not available, so the model is run in an auto-regressive fashion: the sampled output at one time step will be included in the input at the next time step.

Truncated backpropagation through time

Training RNNs on long sequences can be computationally expensive. To improve the training speed, Truncated Backpropagation Through Time (TBTT) is used to train HRNNs. Each long sequence is split into short subsequences, and gradients are only back-propagated to the beginning of each subsequence. In general, if the subsequence length is increased, the performance will be improved, but the memory usage and convergence time will also increase.

It should be noted that although backpropagation is truncated, the history vector $\mathbf{h}_t$ is bootstrapped from one subsequence to the next: the last history vector of one subsequence will be the initial history vector of the following subsequence. As a result, the generated audio samples exhibit dependency much longer than the length of a subsequence. For example, Mehri et al. [15] state that their best models are trained on subsequences of length 512, which corresponds to 32 milliseconds, a small fraction of the length of a single a phoneme of human speech, but the generated samples exhibit longer word-like structures.

3.2.2 Dilated CNN

Similar to HRNN, DCNN also models the probability of a waveform as a product of conditional probabilities as shown in equation 3.1. A major difference is that there is no recurrent layer in a dilated CNN, and the long history needed for $p(x_t|x_1,...,x_{t-1})$ is modeled by a CNN, which has a large receptive field. As there is no recurrent layer, backpropagation does not need to be truncated, and the training process can be made faster by parallelizing computations. During training, the true waveform to be generated $x_{1:T} = \{x_1,...,x_T\}$ is available, so the input at each time step is available and the process of computing $p(x_t|x_1,...,x_{t-1})$ for different time steps can be parallelized. During synthesis, the model runs in an auto-regressive fashion, but the computations within one time step can still be made in parallel.

Causal convolution

When using CNN to process images, the concept of sequence is usually not important. In contrast, when using CNN to process audio, which is naturally sequential, the convolutions should be causal. Figure 3.5 shows a stack of CNN using only causal convolutions. At any time step t, the computation of $p(x_t|x_1,...,x_{t-1})$ does not depend on any future time step.

In the HRNN case, the complete history $\{x_1,...,x_{t-1}\}$ is represented by an abstract history vector $\mathbf{h}_t$. When using CNN for history representation, the complete history is truncated, and the length of the history considered, $\{x_{t-HL},...,x_{t-1}\}$, is equal to the receptive field of the CNN. A problem of standard causal convolutions is that the length of history grows linearly with the depth of the network: $HL = NL \times (FL - 1) + 1$, where HL denotes history length, NL denotes number of layers in the network, and FL denotes filter length. To model a long history, too many layers will required and there will not be enough data to train the network.

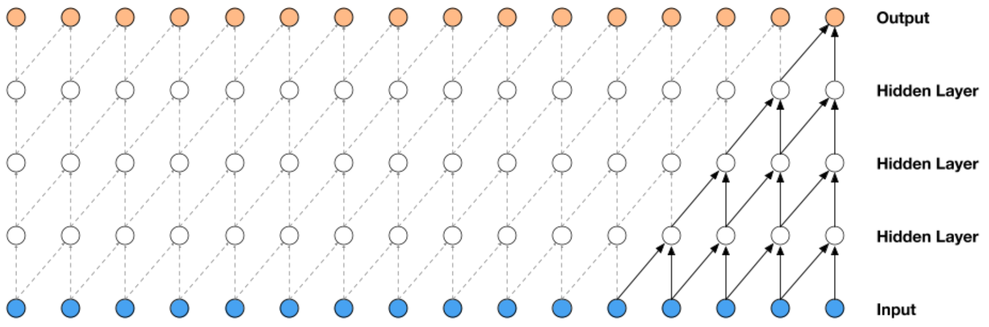


Fig. 3.5 visualization of a stack of CNNs [25].

Dilated convolution

To solve this problem, dilated convolutions can be used. A dilated convolution is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. This is similar to a convolution with a larger filter derived from the original filter by dilating it with zeros, but is significantly more efficient [25]. Figure 3.6 shows a stack of dilated CNNs. Compared to a normal convolution, a dilated convolution allows the network to model a long history with fewer layers and hence fewer parameters. For a dilated CNN, the length of history grows exponentially with the depth of the network: $HL = FL^{NL}$. For the network in figure 3.6, $FL = 2$, $NL = 4$, and $HL = 2^4 = 16$. In contrast, for the network in figure 3.5, FL and NL are the same, but HL is only 5.

In practice, the dilation is doubled for each layer up to a limit and then repeated. For example: 1,2,4,8,1,2,4,8,... Using an exponentially growing dilation allows the length of history to grow exponentially with the depth of the network, which founds the basis of dilated CNN. However, if the dilation is too large, there might be too much loss of information, and the complete history might be badly approximated. Hence when a limit is reached, the dilation is reset to 1.

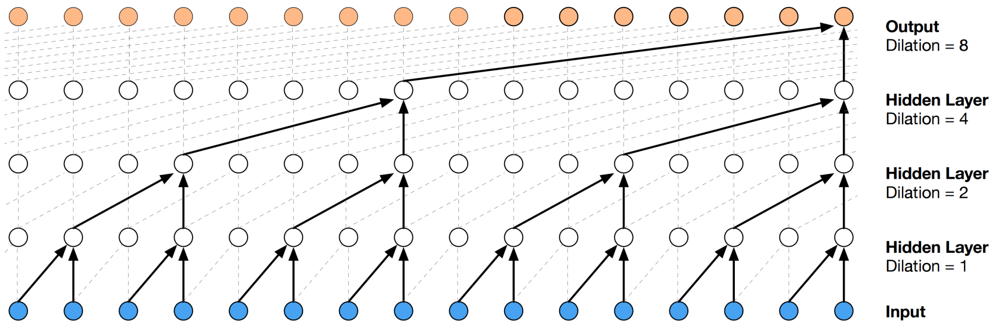


Fig. 3.6 visualization of a stack of dilated CNNs [25].

Gated activation function

When representing a long history by a sequence of previous audio signal points, ideally different signal points should be given different importance. In general the signal points representing close history should be considered more important than the ones representing distant history, although this is not always the case. In speech audio, for example, the articulation of one word generally depends more on the words close to it in the sentence, but can also depend on a related word far away in the sentence or even in another sentence.

In DCNN, the gated activation unit [26] can be used. This can be formulated as follows. $*$ denotes a convolution operator, $\odot$ denotes an element-wise multiplication operator, σ denotes a sigmoid function, $\mathbf{o}_{1:N}$ denotes the output of a convolutional layer, k denotes the layer index, f and g denote filter and gate, and $\mathbf{W}$ denotes model parameters. It is found that this gated activation function results in better performance than the rectified linear activation function [17] for modeling audio data [25].

$$\mathbf{o}_{1:N} = tanh(W_f^{(k)} * x_{1:T}) \odot \sigma(W_g^{(k)} * x_{1:T}) \tag{3.12}$$

Gating allows the network to control the importance of the different parts of the output for each layer, and hence control the importance of different parts of the history. As the sigmoid function σ outputs a value between 0 and 1, the importance of the output of $tanh$ is adjusted according to the position of the filter, i.e. which part of the history the filter is processing. The parameters for gating are all learnable, so the network can automatically learn which part of the history is more important.

Residual and skip connection

In practice many dilated CNN blocks are stacked to form a deep network, and Stochastic Gradient Descent (SGD) is used for training. Hence the model can be prone to the vanishing gradient problem [19]. To speed up convergence and allow training of very deep models, both residual and parameterized skip connections are used throughout the network, as shown in figure 3.7.

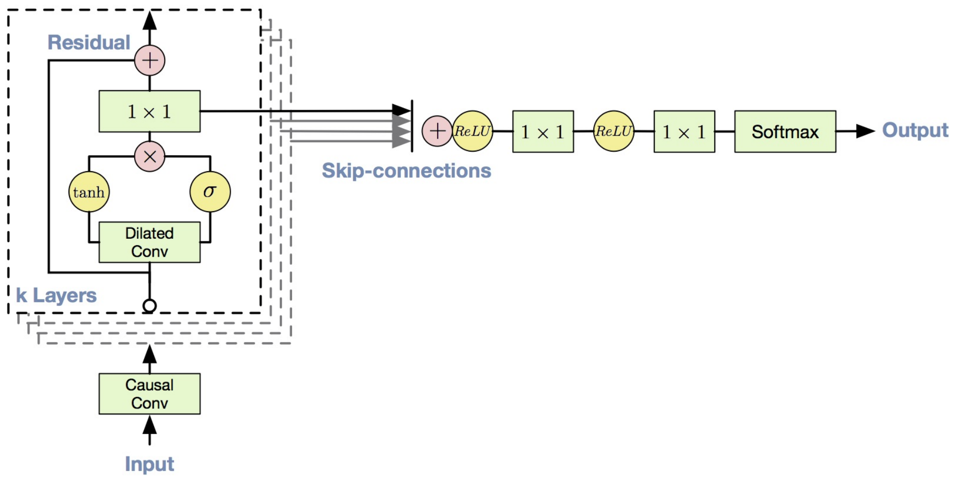


Fig. 3.7 overview of the residual block and the entire architecture [25].

3.3 Preliminary experiments

Several experiments are performed to investigate the characteristics of unconditional waveform-level synthesis. Both music and speech have been generated. By default, the synthesis system uses hierarchical RNN and models the distribution of an individual audio signal point as a categorical distribution, and linear quantization is applied to quantize raw audio data. The implementation of hierarchical RNN is based on the open source code from github[3]. The code used for this thesis is available on github[4] and the audio samples generated in various experiments are available on the speech group website[5].

3.3.1 Unconditional music synthesis

To make sure that the implementation of hierarchical RNN is correct, the first experiment is aimed at replicating the results of unconditional music synthesis in the reference paper [15].

Experimental setup

As in the reference paper, the dataset used for this experiment is the collection of all 32 Beethoven's piano sonatas, which are publicly available[6], amounting to 10 hours of non-vocal audio. 88% of the data are used for training, 6% for validation and 6% for test. The

[3]https://github.com/soroushmehr/sampleRNN_ICLR2017
[4]https://github.com/qingyundou/sampleRNN_QDOU
[5]http://mi.eng.cam.ac.uk/raven/babel/MLSALT/
[6]https://archive.org/

audio waveform data are first concatenated into one very long sequence, and then split into 8-second long sequences. Each batch contains several 8-second long sequences. Inside each batch, each 8-second long sequence is split into several mini sequences, on which truncated backpropagation through time is performed. By default the batch size is 128. The data are quantized into 256 integer values. The sampling rate is always 16kHz.

At training time, Stochastic Gradient Decent (SGD) is used to minimize the Negative Log-Likelihood (NLL). Gradients are hard-clipped to remain in the range $(-1,1)$. Update rules from the Adam optimizer [12] with an initial learning rate of 0.001 ($\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1e^{-8}$) is used to adjust the parameters. For each system, random search over hyper-parameter values [1] is conducted. The initial RNN state of all the RNN-based models is always learnable. Weight normalization [21] is used for all the linear layers to accelerate the training procedure. 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. [7].

The default synthesis system uses a 2-tier hierarchical RNN. Figure 3.8 shows the structure of the network. $r^{(2)} = 16$, $FS^{(1)} = FS^{(2)} = 16$. Tier 2 is a three-layer deep RNN. GRU is used and the dimension is 1024 for all layers. The input of tier 2 is previous audio signal points, and the output is a series of conditioning vectors, as shown in equations (3.6, 3.7, 3.8). The dimension of conditioning vector is 1024, same as the dimension of GRU. Tier 1 is a feed-forward 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 input of tier 1 is a linear combination of embedded previous audio signal points and the conditioning vector from from tier 2, as shown in equation (3.9). The output of tier 1 is a vector describing the distribution of the next signal point, as shown in equation (3.10). The batch size is 128 and the mini sequence used for TBTT has 1024 audio signal points.

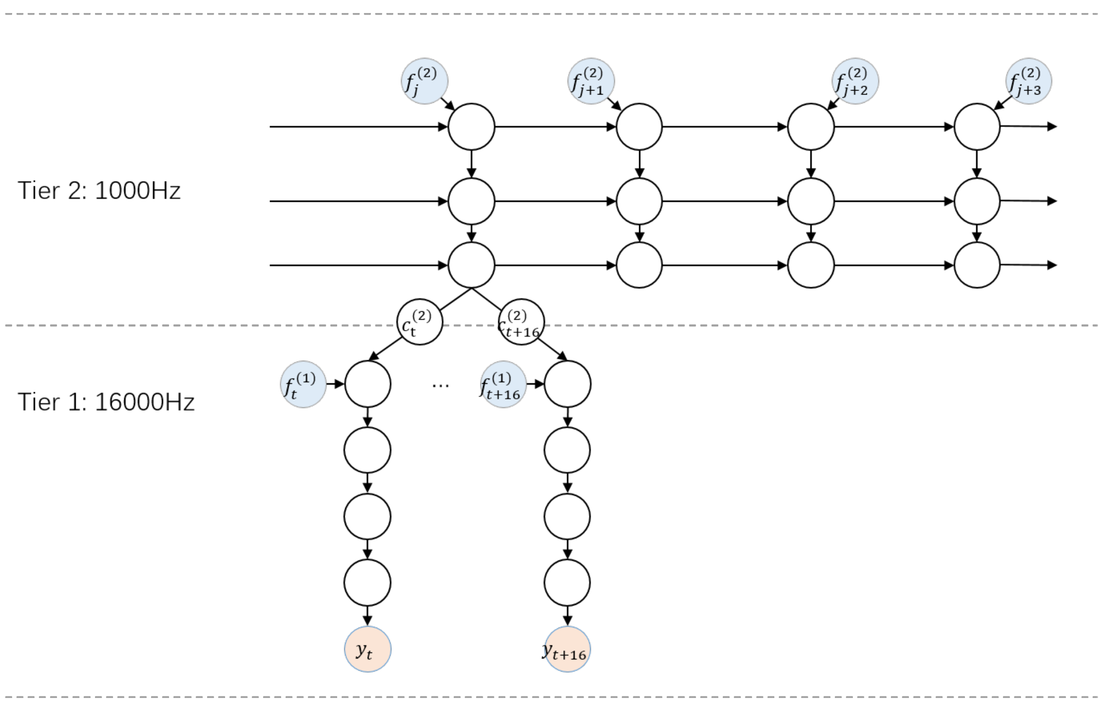


Fig. 3.8 visualization of an unrolled hierarchical RNN with 2 tiers.

In addition, a 3-tier hierarchical RNN is trained as in the reference paper[15]. Figure 3.9 shows the structure of the network. $r^{(3)} = 2$, $r^{(2)} = 4$, $FS^{(1)} = FS^{(2)} = 2$, and $FS^{(3)} = 8$. Tier 3 and tier 2 are one-layer RNNs, and the dimension is 1024 for both. Tier 1 is the same as in the above case. The batch size is 128 and the mini sequence used for TBTT has 512 audio signal points.

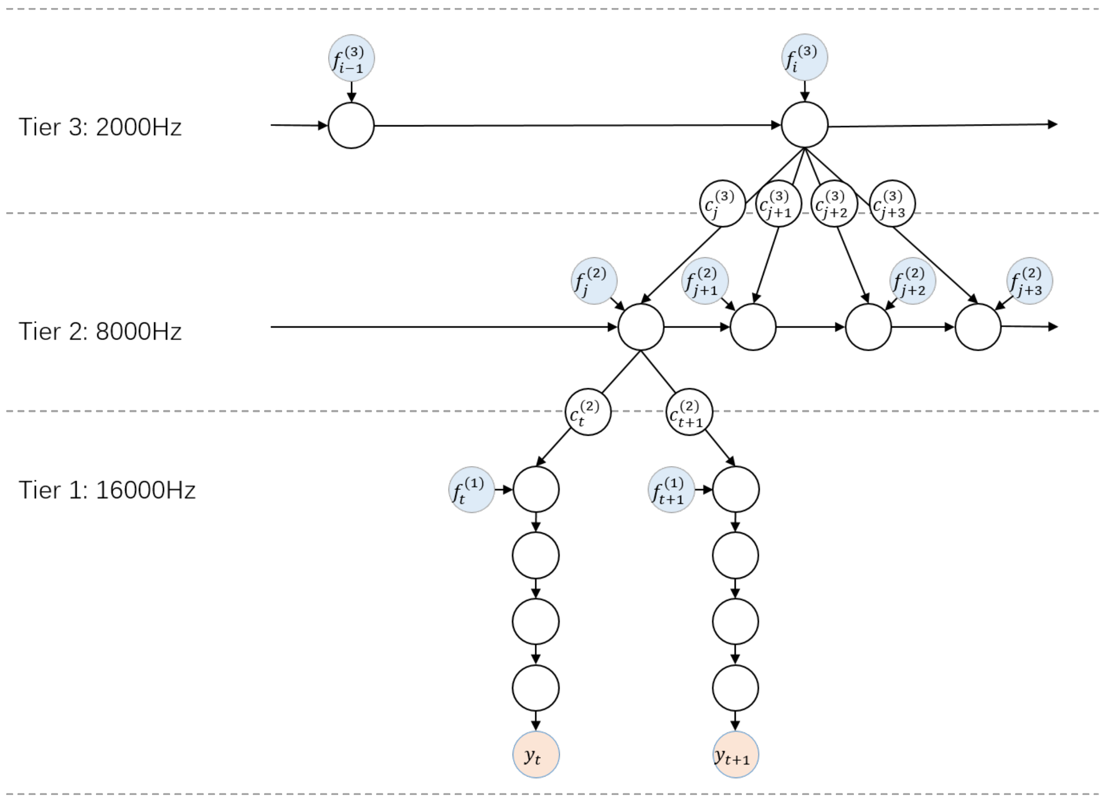


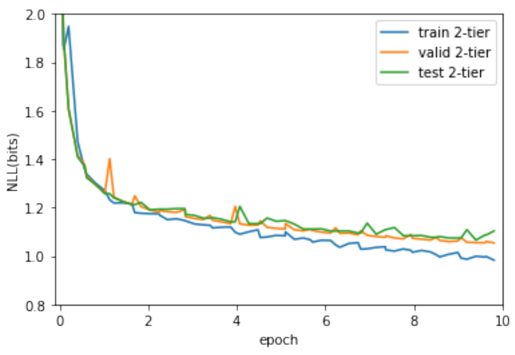
Fig. 3.9 visualization of an unrolled hierarchical RNN with 3 tiers.

Results and analysis

Table 3.2 shows the test NLL of the 2-tier and 3-tier hierarchical RNNs, obtained in new experiments. Similar results are reported by Mehri et al. [15], which confirms that the implementation is correct. Figures 3.10 and 3.11 show the learning curves of the two models. The costs on training data and validation data are computed at each epoch, and the cost on test data is computed when a new lowest validation cost is obtained. It can be seen that the 2-tier model has lower NLL. There are many possible reasons. First, the 2-tier model is deeper; it has three recurrent layers while the 3-tier model only has two. Second, the 2-tier model has larger frame size; for the 2-tier model $FS^{(1)} = FS^{(2)} = 16$, while for the 3-tier model $FS^{(1)} = FS^{(2)} = 2$. Third, the 2-tier model is trained on longer mini sequences; it uses 1024 audio signal points for TBTT, while the 3-tier model only uses 512 points.

Table 3.2 test NLL in bits for 2-tier and 3-tier models

	2-tier model	3-tier model
new experiment	1.066	1.117
reference paper [15]	1.076	1.159



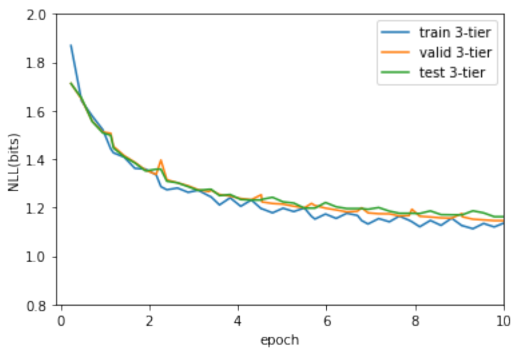


Fig. 3.10 learning curves of 2-tier model. Fig. 3.11 learning curves of 3-tier model.

Figures 3.12, 3.13 and 3.14 show examples of generated and recorded audio waveforms at two different time scales. It can be seen that in general the generated waveforms are similar to real waveforms. Both models are capable of capturing trends at different time scales. For capturing high frequency trends, 2-tier model seems better than 3-tier model. An important reason is that the 2-tier model has a much larger window size ($FS^{(1)} = 16$ instead of 2) at the lowest tier, which operates at the highest frequency.

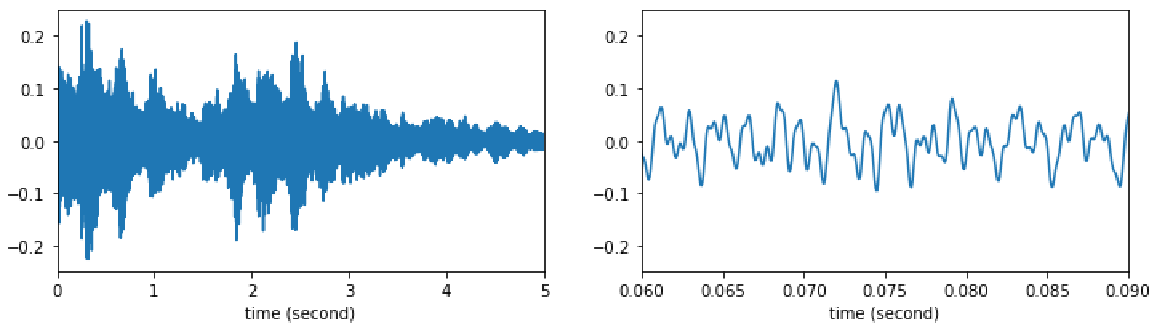


Fig. 3.12 example of recorded music waveform.

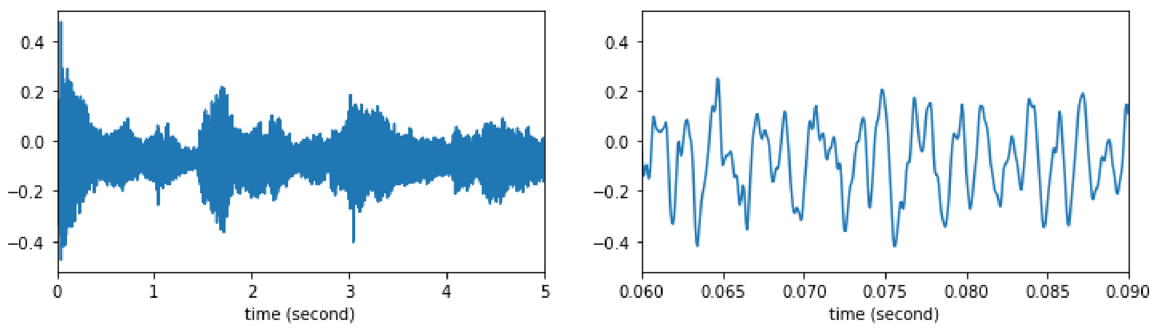


Fig. 3.13 example of music waveform generated from 2-tier model.

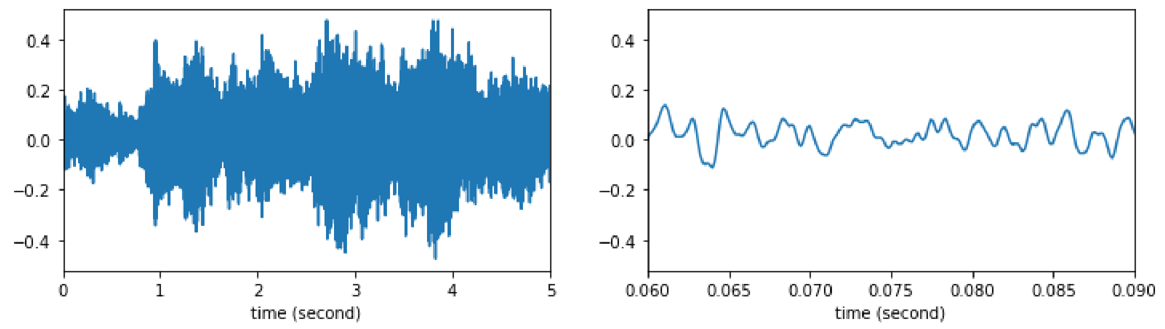


Fig. 3.14 example of music waveform generated from 3-tier model.

3.3.2 Unconditional speech synthesis

Experimental setup

Similar experiments are performed with speech data. The dataset is a collection of 2396 utterances from a single speaker. In total there is about four hours' audio. Data preprocessing and training are the same as in the experiments above. SGD is used to minimize NLL. The audio waveform data are first concatenated into one very long sequence, and then split into 8-second long sequences. Each batch contains several 8-second long sequences. Inside each batch, each 8-second long sequence is split into several mini sequences, on which TBTT is performed. The data are quantized into 256 integer values. The sampling rate is always 16kHz.

Both 2-tier and 3-tier hierarchical RNNs are used for unconditional speech synthesis. The network structures are shown in figures 3.8 and 3.9; the actual parameters will be reported in the following section, as a series of experiments are performed.

Results and analysis

First, the same 2-tier hierarchical RNN as above is used to generate speech, to investigate if the model works for speech data. The batch size is set to 64 to suit the total length of speech audio, which is less than half of music audio. All other parameters are the same as in the music synthesis case. $r = 16$, $FS^{(1)} = FS^{(2)} = 16$. The length of the mini sequence, on which TBTT is performed, is 1024. The training/validation/test split is approximately 88%-6%-6%.

Figures 3.15 and 3.16 show examples of recorded and generated speech waveforms at two different time scales. It can be seen that in general the generated waveforms are similar to real waveforms. Trends at different time scales are captured. Figure 3.17 shows the learning curves of this experiment. The final test NLL is 1.236, which is higher than that of the music synthesis experiment, 1.066. This indicates that speech is more difficult to model than piano music.

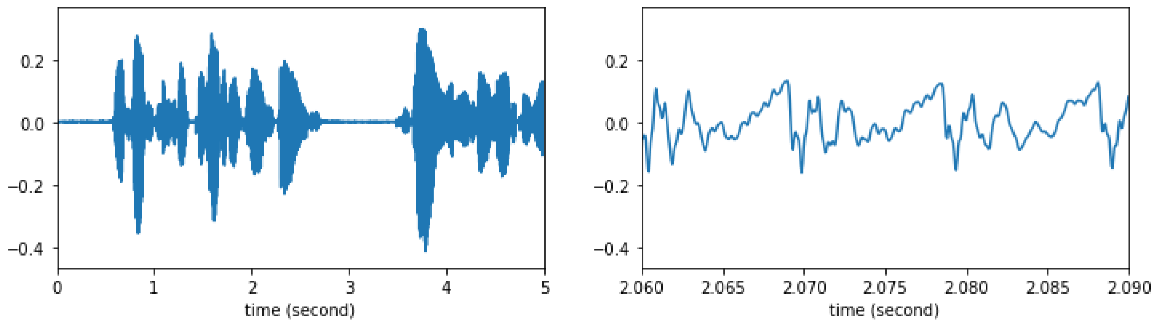


Fig. 3.15 example of recorded speech waveform.

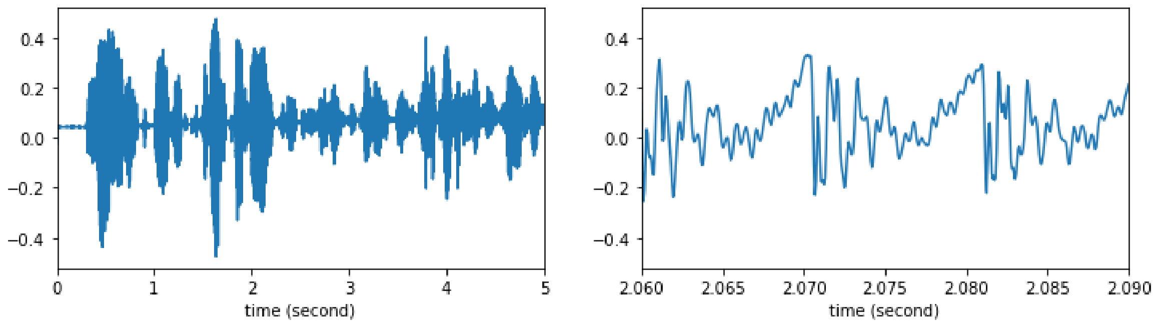


Fig. 3.16 example of speech waveform generated from 2-tier model.

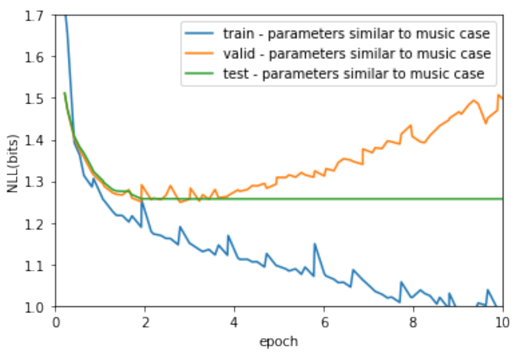


Fig. 3.17 learning curves of 2-tier model, using parameters similar to the music synthesis case.

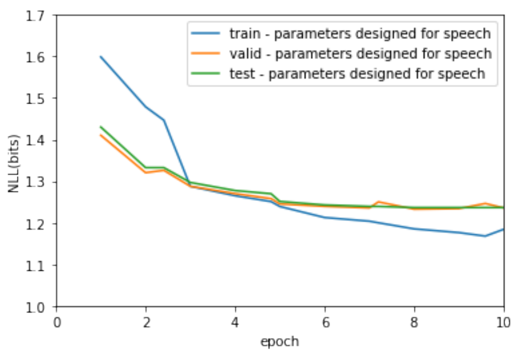


Fig. 3.18 learning curves of 2-tier model, using parameters designed for speech synthesis.

To compare the waveform-level synthesis system to the baseline synthesis system of the speech group, in the following experiments, the speech data are split in the same way as the baseline synthesis system. 2254 utterances are used for training, 70 for validation and 72 for testing. The training/validation/test split is approximately 94%-3%-3%. As there are fewer data in the validation set and test set than before, the default batch size is reduced to 20. To prepare for conditional speech synthesis, the default parameters are also adjusted. The length of the mini sequence, on which TBTT is performed, is 800. For the default 2-tier hierarchical RNN, $r = 20$, $FS^{(1)} = FS^{(2)} = 20$. For the default 3-tier hierarchical RNN, $r^{(3)} = 4$, $r^{(2)} = 20$, $FS^{(1)} = FS^{(2)} = 20$, and $FS^{(3)} = 80$.

Using the parameters designed for speech synthesis, the default 2-tier hierarchical RNN is trained, and figure 3.18 shows the learning curves. This experiment will be referred as the "default unconditional speech synthesis experiment", or the "default experiment" when "unconditional" is implicit. Comparing to figure 3.17, it can be seen that overfitting is less obvious. This is because in the new experiment, more data are used for training, and a smaller batch size is used for SGD, so each gradient update is less likely to reduce the cost on training data. The lowest validation costs are similar, and the best samples generated have similar sound quality. This indicates that the parameters used in the default experiment are well selected. In the following experiments, these parameters will be used by default, and typically only one parameter will be changed to investigate its impact on performance.

In the default experiment, linear quantization is applied. To investigate the influence of applying μ-law, the above experiment is repeated with data quantized after μ-law transformation as shown in equation (3.2). Figure 3.19 compares the two quantization schemes in terms of training cost and validation cost. It can be seen that applying μ-law results in higher cost. However, the samples generated still sound similar. Hence in the following experiments linear quantization is used by default, and μ-law is only used to fine-tune synthesis systems that have very good performances.

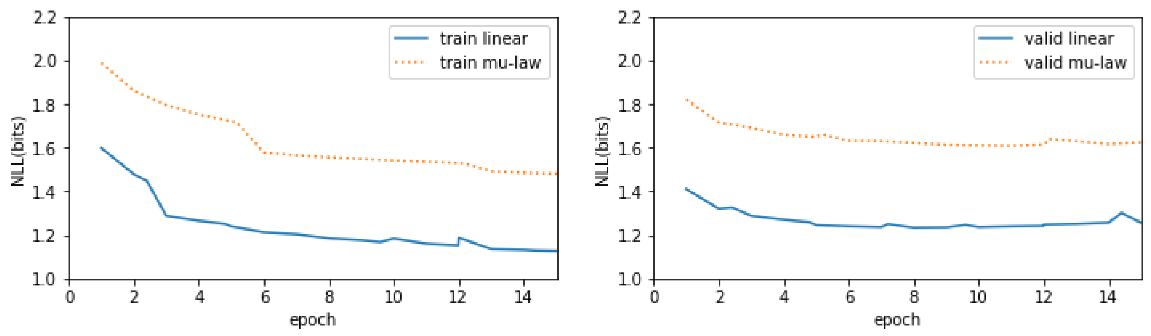


Fig. 3.19 costs for linear and mu-law quantization.

To investigate the influence of the length of mini sequence, on which TBTT is performed, a 2-tier hierarchical RNN is trained, where the length of mini sequence is 2000 instead of the default value 800. Figure shows compares different mini sequence lengths in terms of the resulting training cost and validation cost. It can be seen that the two lengths yield similar costs. The generated samples are also similar. One reason is that the history vector is bootstrapped during training and synthesis. It can also be seen that when the mini sequence is shorter, the convergence is faster. This is because there are more mini sequences for SGD, and the model parameters are updated more frequently. However, shorter mini sequence length does not necessarily mean better performance, as noisier SGD is less likely to converge to a good point. In the following experiments, 800 will still be the default mini sequence length.

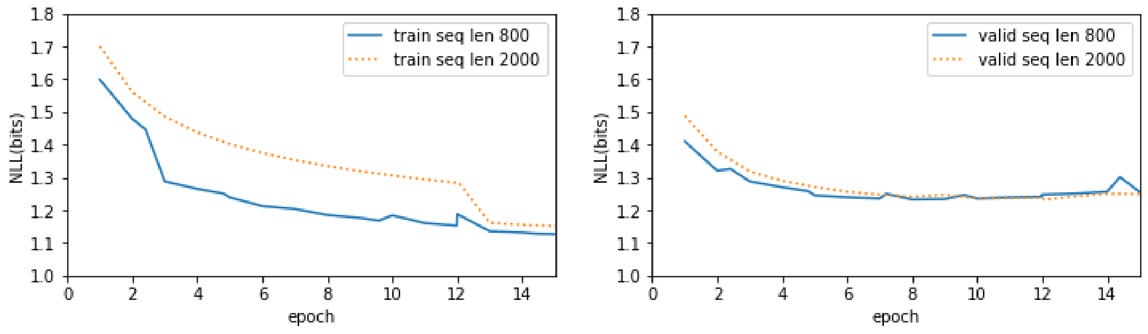


Fig. 3.20 costs for different mini sequence lengths.

To investigate the influence of the number of tiers, a 3-tier hierarchical RNN is trained. The default parameters are used: $r^{(3)} = 4$, $r^{(2)} = 20$, $FS^{(1)} = FS^{(2)} = 20$, and $FS^{(3)} = 80$. The third tier is added on top of the default 2-tier hierarchical RNN, and $NR^{(3)}$, the number of RNN layers to use for the third tier, is tuned. Figure 3.21 compares different $NR^{(3)}$s in terms of the resulting training cost and validation cost. The experiment for $NR^{(3)} = 3$ was stopped earlier to save GPU usage, as it was clearly worse than other cases. It can be seen that $NR^{(3)} = 1$ yields the lowest cost. The generated samples show the same trend: the $NR^{(3)} = 1$ model generates samples with the best sound quality. This is because $NR^{(3)} = 1$ is consistent with the upsampling rate: $r^{(3)} = 4$, $r^{(2)} = 20$. As tier 3 has a less complicated task than tier 2, fewer layers are needed. If the $r^{(k)}$ and $FS^{(k)}$ change, $NR^{(k)}$ should change accordingly.

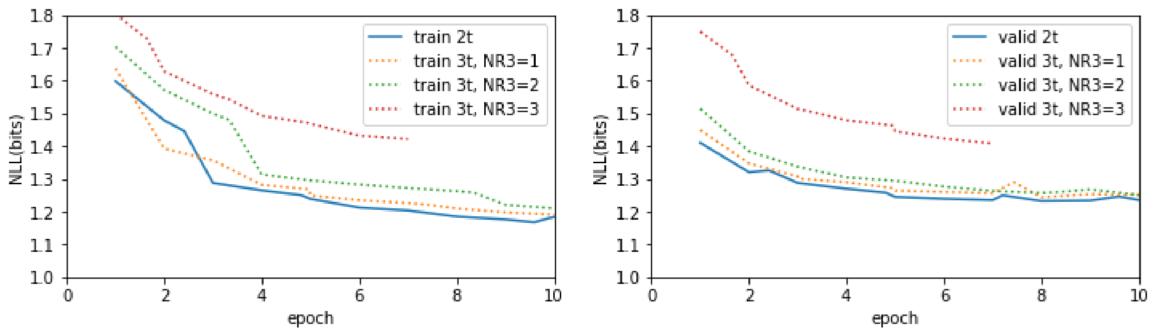


Fig. 3.21 costs for 2-tier and 3-tier models.

Chapter 4

Conditional (speech) synthesis

4.1 Incorporating additional information

4.1.1 Network structure

With additional information $\mathbf{l}_{1:T} = \{\mathbf{l}_1 ... \mathbf{l}_T\}$, the conditional distribution of a waveform $x_{1:T} = \{x_1 ... x_T\}$ can be formulated as in equation (4.1). At each time step, the distribution of an audio signal point depends on both previous signal points and additional information.

$$p(x_{1:T}|\mathbf{l}_{1:T}) = \prod_{t=1}^{T} p(x_t|x_1...x_{t-1}, \mathbf{l}_t) \tag{4.1}$$

The network structures described in the unconditional synthesis chapter will change accordingly. For hierarchical RNN, the labels are typically feed into the network at tiers operation below waveform-level frequency; the reason will be explained in the following sections. The input at those tiers will be a combination of label on additional information, previous audio signal points, and conditioning vector from the tier above except for the top tier. Equation (3.6) will be reformulated as equation (4.2). k denotes tier, $\mathbf{W}_l$ denotes the parameters for incorporating additional information.

$$\mathbf{e}_{t^{(k)}} = \begin{cases} \mathbf{W}_l \mathbf{l}^{(k)}_{t^{(k)}} + \mathbf{W}_f \mathbf{f}^{(k)}_{t^{(k)}} + \mathbf{c}^{(k+1)}_{t^{(k)}}; & 1 < k < K \\ \mathbf{W}_l \mathbf{l}^{(k)}_{t^{(k)}} + \mathbf{f}^{(k)}_{t^{(k)}}; & k = K \end{cases} \tag{4.2}$$

For dilated CNN, the input to the network will be a combination of text label and previous audio signal points. Equation (3.12) will be reformulated as equation (4.3). k denotes layer index, $W_{fl}^{(k)}$ and $W_{gl}^{(k)}$ denotes additional filter and gating parameters for incorporating additional information.

$$\mathbf{o}_{1:N} = tanh(W_{fx}^{(k)} * x_{1:T} + W_{fl}^{(k)} * \mathbf{l}_{1:T}) \odot \sigma(W_{gx}^{(k)} * x_{1:T} + W_{gl}^{(k)} * \mathbf{l}_{1:T}) \tag{4.3}$$

4.1.2 Text labels

The additional information can be music style, speaker identification, text labels or other types of information. This chapter mainly investigates how to incorporate text labels for conditional speech synthesis, which is generally more complicated than incorporating other types of information. A major characteristic of text label is that it changes over time. Unlike speaker identification, which is normally the same throughout an utterance, text label typically changes at a frequency of 200Hz. Hence incorporating static information such as speaker ID can be considered a special case of incorporating text information, where the information is the same at each time step.

Given raw text, the label sequence $\mathbf{l}_{1:T}$ can be obtain by various approaches. Two general approaches are investigated in this thesis. The first approaches is to derive standard text labels from raw text using traditional methods. This approach yields text labels that are easy to interpret, but requires linguistic knowledge. The second approach is to train a neural network to derive text labels for conditioning purpose. In theory, this approach can yield text labels close to ideal, and no linguistic knowledge is required. However, in practice the uninterpretable labels make the network difficult to tune.

Standard labels

A standard Text-to-Speech (TTS) system consists of two parts: text analysis and speech synthesis. The text analysis part takes a word sequence as input and outputs a sequence of standard labels describing the text. The speech synthesis part takes the label sequence as input and outputs a speech signal sequence. The standard labels are 601-dimensional vectors extracted from text and aligned with waveform. Each label corresponds to 80 signal points, i.e. the frequency of the label is 200Hz. This frequency is much lower than the frequency of the waveform, which is 16000Hz. Therefore, to be used for waveform level synthesis, standard labels need to be upsampled. There are various forms of upsampling, such as repetition and linear interpolation. In this thesis linear interpolation is applied and will be analyzed in detail in section 4.2.

The high dimensionality of the standard labels can be problematic in training the conditional model, especially when the input of the conditional model is a linear combination of text label and the input of the unconditional model. For example, for a 2-tier hierarchical RNN, if the frame size of the second tier is 20, then the dimensionality of text label is 30 times higher. Although in theory the network can adjust the weights for text label and the weights for previous audio signal points, in practice the network is very likely to be stuck in a local minimum where information from the text label overwhelms that from the previous audio signal points. This problem can be solved by various methods, such as using bottleneck features, using non-linear combination, pretraining the network and careful initialization. In this thesis pretraining is applied and will be analyzed in detail in section 4.3 about implementation issues.

Labels generated by neural networks

Instead of using standard text labels, neural networks can be trained to map raw text to labels that can be fed into the unconditional synthesis models. An encoder-decoder neural networks with attention mechanism is found to work well [15]. The encoder is a bidirectional RNN that maps text to some intermediate representation $\mathbf{w}_{1:N}$. The decoder is an RNN with attention mechanism that maps the intermediate representation $\mathbf{w}_{1:N} = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_N\}$ to labels $\mathbf{l}_{1:T} = \{\mathbf{l}_1, \mathbf{l}_2, ..., \mathbf{l}_T\}$ that can be used for conditioning. This network is shown in figure 4.1, and can be formulated as follows. t denotes time step; $\mathbf{s}$ denotes history generated by the RNN; $\mathbf{g}$ denotes an intermediate vector; $\mathbf{a}$ denotes attention weights; $f^{(ATT)}$, $f^{(DNN)}$ and $f^{(RNN)}$ represent the RNN with attention mechanism.

$$\mathbf{a}_t = f^{(ATT)}(\mathbf{s}_{t-1}, \mathbf{a}_{t-1}, \mathbf{w}_{1:N}) \tag{4.4}$$

$$\mathbf{g}_t = \sum_{n=1}^{N} a_{t,n}\mathbf{w}_n \tag{4.5}$$

$$\mathbf{l}_t = f^{(DNN)}(\mathbf{s}_{t-1}, \mathbf{g}_t) \tag{4.6}$$

$$\mathbf{s}_t = f^{(RNN)}(\mathbf{s}_{t-1}, \mathbf{g}_t, \mathbf{l}_t) \tag{4.7}$$

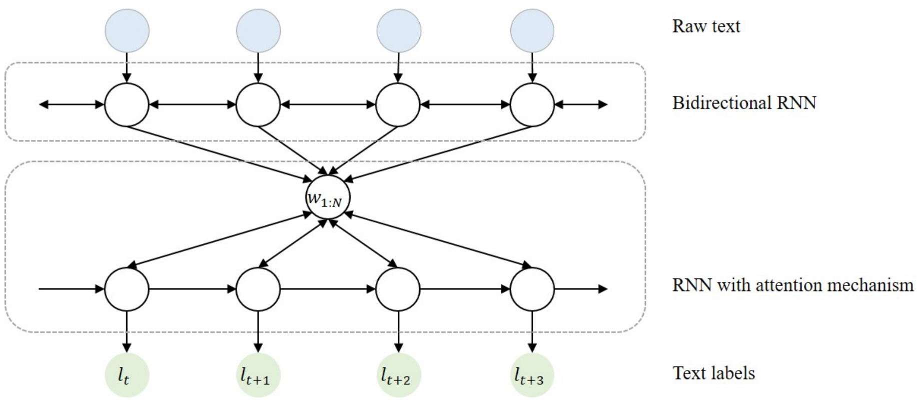


Fig. 4.1 attention-based recurrent sequence generator.

4.2 Combining hierarchical RNN with standard labels

This thesis develops a new conditional synthesis model, combining hierarchical RNN and standard labels. In particular it investigates how to design the structure of hierarchical RNN and how to upsample standard labels so that they are suitable for conditional synthesis.

4.2.1 Designing network structure

An important characteristic of hierarchical RNN is that it has several tiers, which have different inputs and are meant to model audio structures at different time scales. Higher tiers operate on a larger time scale / lower frequency, and lower tiers operate on a smaller time scale / higher frequency. Feeding the labels into different tiers will lead to very different results.

The first tier of hierarchical RNN is a feed-forward network, and there is no recurrence. If labels are only added at this tier, the history of previous labels will not at all be modeled. The other tiers are all deep RNNs, and can model history up to a certain length. Although in theory an RNN models the complete history, in practice the history is only modeled up to a few hundred time steps. If a tier operates on a lower frequency, it will have a larger time step, and will be more suitable to model audio structure at a larger time scale. Hence adding labels on a higher tier allows the network to model a longer history of previous labels. On the other hand, if labels are added to a higher tier, at each time step the current label will have a smaller impact on the the lowest tier, which makes predictions at waveform-level. As the labels go through the network, some information might be lost before the lower tiers are reached. For example two different labels might be mapped to the same conditioning vector after some linear and non-linear mappings. Therefore labels should be added to both high-level and low-level tiers.

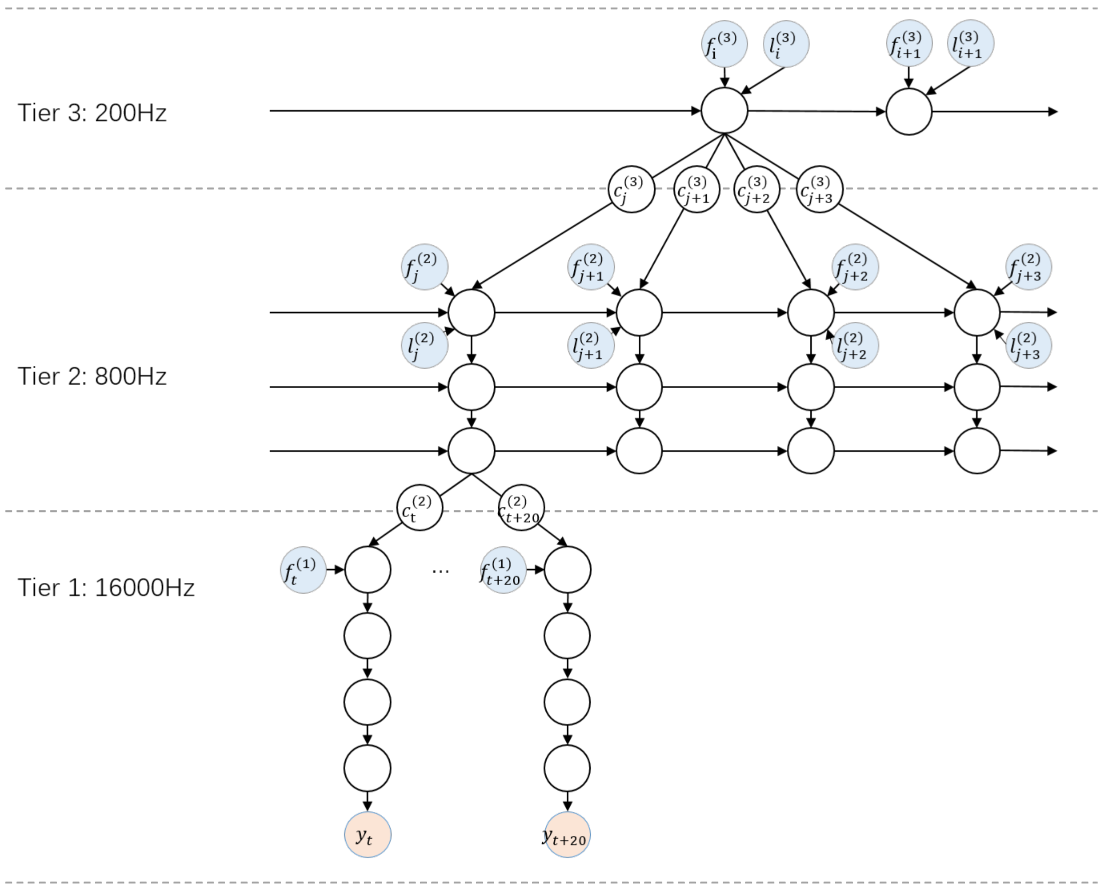


Fig. 4.2 visualization of an unrolled conditional hierarchical RNN with 3 tiers.

Figure 4.2 shows the structure of a conditional hierarchical RNN that is found to work well. $r^{(3)} = 4$, $r^{(2)} = 20$, $FS^{(1)} = FS^{(2)} = 20$, and $FS^{(3)} = 80$. Tier 1 is a DNN, and operates at waveform-level frequency, 16000Hz. Tier 2 is a three-layer RNN; it operates at 800Hz, and outputs 20 conditioning vectors at each time step. Tier 3 is a one-layer RNN; it operates at 200Hz, and outputs 4 conditioning vectors at each time step. Labels are added at both tier 2 and tier 3, to model audio structure at different time scales. The recurrent unit is GRU. From experience, history as long as about 100 time steps will be traced. For tier 2, each time step corresponds to 1.25ms, so the effective history length is 125ms, which is at phoneme-level. For tier 3, each time step corresponds to 5ms, so the effective history length is 500ms, which is at word-level. It takes about 5 days (120 hours) to train this network, so the parameters are not extensively tuned. In general, using more tiers will lead to better performance, and the tiers should operate at sufficiently diverse frequencies. For a particular RNN tier, the number of layers to use should be consistent with the upsampling rate $r^{(k)}$ of that tier. A tier with higher upsampling rate outputs more conditioning vectors, and should have more layers. This analysis has been confirmed by the preliminary experimental results in section 3.3.2. In the network shown in figure 4.2, $r^{(3)} = 4$ and $r^{(2)} = 20$, accordingly tier 3 has only one layer and tier 2 has three layers.

4.2.2 Upsampling labels

Standard labels have a relatively low frequency of 200Hz, and are typically upsampled to be used for waveform level synthesis. As each label corresponds to 80 audio signal points, to be added to tier k, the upsampling rate is $80/FS^{(k)}$. For example, for the network shown in figure 4.2, at tier 2 the labels are upsampled and the upsampling rate is $80/20 = 4$.

There are various forms of upsampling, such as repetition and linear interpolation. In this thesis linear interpolation is applied. If two neighboring labels are the same, linear interpolation is the same as repetition. If two neighboring labels are different, applying linear interpolation assumes that the transition happens evenly. Figure 4.3 illustrates how upsamling is performed. The upsampling rate is 4; 10 original standard labels are plotted as orange crosses, and 40 upsampled labels are plotted as blue dots.

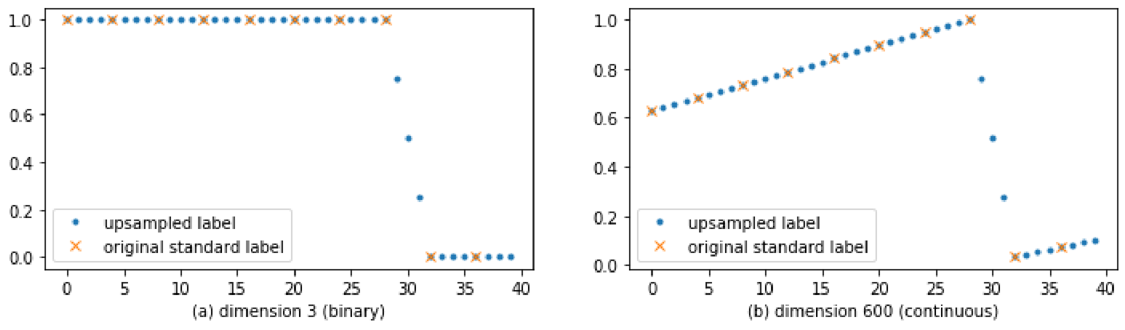


Fig. 4.3 visualization of upsampling by linear interpolation.

Each standard label has 601 dimensions; the first 592 dimensions are binary, and the rest are continuous. Figure 4.3 (a) shows a binary dimension, and figure 4.3 (b) a continuous dimension. When applying linear interpolation, all dimension are considered continuous. Although the resulting values for binary dimensions are difficult to interpret, they carry useful information for the neural network.

4.3 Implementation issues

While neural networks are powerful for approximating complicated functions, their performance is sensitive to factors such normalization [10] and optimization methods used during training. In this section, several implementation issues are analyzed. The analysis is in the context of conditional synthesis, but is also useful for other cases where neural networks are used.

4.3.1 Data preprocessing

In general, data preprocessing includes normalization and dimensionality reduction. Normalization linearly maps the raw data, which probably have a wide dynamic range, to a certain range that is suitable for the activation function. As gradient-based methods are typically used to train neural networks, data should be mapped to such a range that the resulting gradient is unlikely to be too small. For a 3-tier hierarchical RNN, the first tier uses RELU activation, so the input data is mapped to the range $(0,1)$, which is above zero; the other tiers use tanh and sigmoid activation, so the input data is mapped to the range $(-2,2)$, which is zero-centered. For data with more than one dimension, normalization also adjusts the importance of different dimensions. As label data are 601 dimensional vectors, normalization is performed separately on each dimension.

Dimensionality reduction can be performed on raw data to both reduce the number of parameters and improve performance. For high dimensional data, some dimensions might be more important than others. If all the weights are initialized in the same way, it might take a long time for SGD to converge to the point where weights for important dimensions are higher. For a hierarchical RNN, if labels are added to tier 1 and the frame size $FS^{(1)}$ is low, for example 20, the 601 dimensional labels will probably overwhelm the 20 previous audio signal points, and result in waveforms that are too smooth. For conditional synthesis, PCA can be performed on standard labels, and bottleneck features can be used, which are expected to improve performance.

4.3.2 Pretraining

One problem of using gradient-based method is that convergence to the global minimum of the cost function is not guaranteed. The more parameters a neural network has, the more likely it is to be trapped in a local minimum. In some experiments performed for this thesis, it is found that the conditional synthesis model generates worse samples than the corresponding unconditional synthesis model. One reason is that the conditional synthesis model has more parameters and is trapped in a local minimum.

To solve this problem, in this thesis all condition synthesis models are pretrained as uncondition synthesis models. During pretraining, at first the weights for labels are locked to zero. At each epoch the model parameters are saved. After several epochs, typically 15, the parameters resulting in the lowest cost on validation data will be selected as the starting point for full training, during which the weights for labels are also updated. This technique is found to be very useful for the conditional speech synthesis experiments.

4.3.3 Training speed

SGD is used to train the models because computing the gradient over all training data can be too expensive. Each gradient update only considers a random batch of training data, and there is a trade-off to make. If the batch is too large, the parameters will not be updated frequently enough; if the batch is too small, the gradient will be too noisy to update the parameters towards the right direction. For the experiments in this thesis, it is found that if one gradient update is performed for each utterance in the training data, which is similar to setting batch size to 1, the training will take more than 10 days. Therefore, the batch size used for the experiments is typically larger than 20. As part of the SGD method, learning rate is also important for training speed. In general higher learning rate leads to faster training. However, this is not always the case, especially when gradient is only computed over a small set of training data. In this thesis Adam optimization [12] is used to automatically adjust the learning rate.

Chapter 5

Experiments

5.1 Performance metric

Unlike speech recognition, speech synthesis does not have a simple objective performance metric such as word error rate. The objective performance metrics are only indicative, and subjective listening tests are the gold-standard for speech synthesis assessment. For speech, there are many attributes to assess, such as naturalness, intelligibility and expressiveness. Hence subjective listening tests are typically limited to a small number of attributes.

For the experiments in this section, several different performance metrics will be used. First, during training the cost function is evaluated on training, validation and test data at each epoch. The results are presented as three learning curves. Second, the waveforms generated are plotted and compared to the waveforms of the corresponding recorded utterances. These two metrics can be used to tell if one system is considerably better than another. The problem is that if two waveforms are both similar to the reference waveform, it will be difficult to tell which one is better. The root-mean-square error (RMSE) between generated and reference waveforms can be computed, but is sensitive to phase shift. To solve this problem and to perform deeper analysis, for some experiments vocoder features are extracted from the waveforms, and RMSE is computed between the feature trajectories. Denote two trajectories as $x_{1:T} = \{x_1, ..., x_T\}$ and $y_{1:T} = \{y_1, ..., y_T\}$, RMSE can be formulated as in equation (5.1).

$$RMSE = \sqrt{\frac{1}{T}\sum_{i=1}^{T} |y_t - x_t|^2} \tag{5.1}$$

The vocoder features are 246-dimensional vectors with a 5 millisecond frame, and each feature vector can be split into 5 streams:

1. mel-cepstrum: 180-dimensional comprising 60-dimensional STRAIGHT derived mel-cepstrum parameters, delta and delta-delta parameters;
2. log-F0: 1-dimensional REAPER derived log-transformed F0;
3. delta log-F0: 1-dimensional delta parameters for log-F0;
4. delta-delta log-F0: 1-dimensional delta parameters for delta log-F0;
5. aperiodicity: 63-dimensional comprising 21-dimensional STRAIGHT derived aperiodicity parameter, delta and delta-delta parameters.

5.2 Experimental setup

The waveform dataset is a collection of 2396 utterances from a single speaker, same as the dataset for the unconditional speech synthesis. Data preprocessing is also the same. The label dataset is a collection of corresponding labels. The labels are aligned with the waveforms, and each label corresponds to 80 signal points, i.e. the frequency of the label is 200Hz. Each label is a 601-dimensional vector; the first 592 dimensions are binary and the other dimensions are continuous. Each dimension is normalized separately.

5.2.1 Baseline: feature-level synthesis

The baseline synthesis model operates at feature-level. At each time step, it takes a 601-dimensional label vector $\mathbf{l}$ as input and outputs a 247-dimensional vocoder feature vector $\mathbf{v}$, which are then used by a vocoder to generate q sequence of signal points $x_{1:80}$ forming a waveform. The mapping from label to vocoder feature is realized by a neural network. As shown in figure 5.1, it has three GRU-RNN layers, and the dimension of each layer is 1024; the output layer is a linear layer mapping a 1024-dimensional vector to a 247-dimensional vector. During training, L2 regularization is applied. The optimizer is Adam [12] and the learning rate is tuned. Although called a baseline, this system is well tuned and can generate samples very similar to recorded speech.

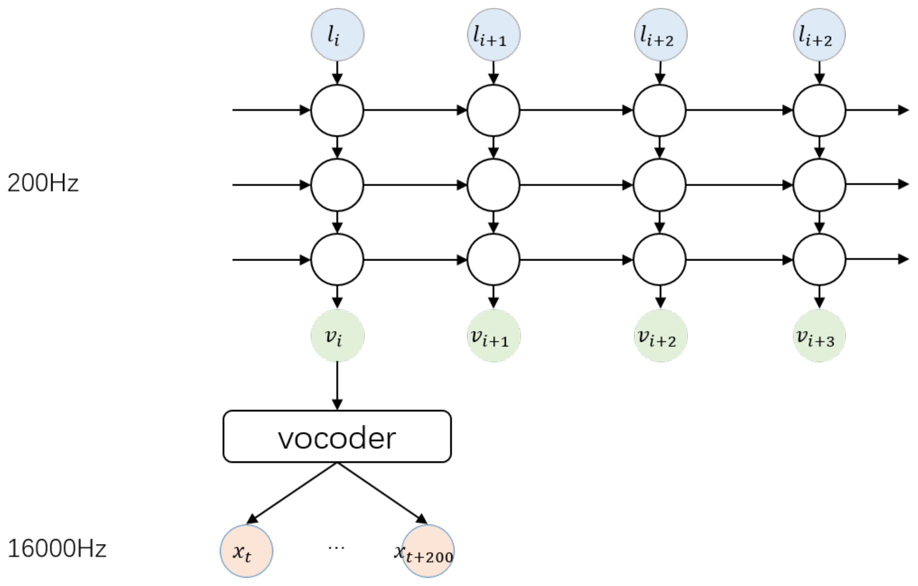


Fig. 5.1 visualization of the baseline model.

5.2.2 Waveform-level synthesis

For waveform-level synthesis models, the training method is the same as in the unconditional speech synthesis experiments. SGD is used, and gradients are hard-clipped to remain in the range $(-1,1)$. Update rules from the Adam optimizer [12] with an initial learning rate of 0.001 ($\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1e^{-8}$) is used.

The default model is a 2-tier conditional hierarchical RNN, as shown in figure 5.2. There are two tiers, $r = 20$, $FS^{(1)} = FS^{(2)} = 20$, and standard labels are added to tier 2. Tier 2 is a three-layer RNN; GRU is used and the dimension is 1024 for all layers. Tier 1 is four-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 size of the embedding layer was also 256. The batch size is 20 and the mini sequence used for TBTT has 800 audio signal points.

In the following experiments, typically only one parameter will be different from the default model and will be mentioned. Other unmentioned parameters are the same as the default model. This allows investigation on the impact of that parameter.

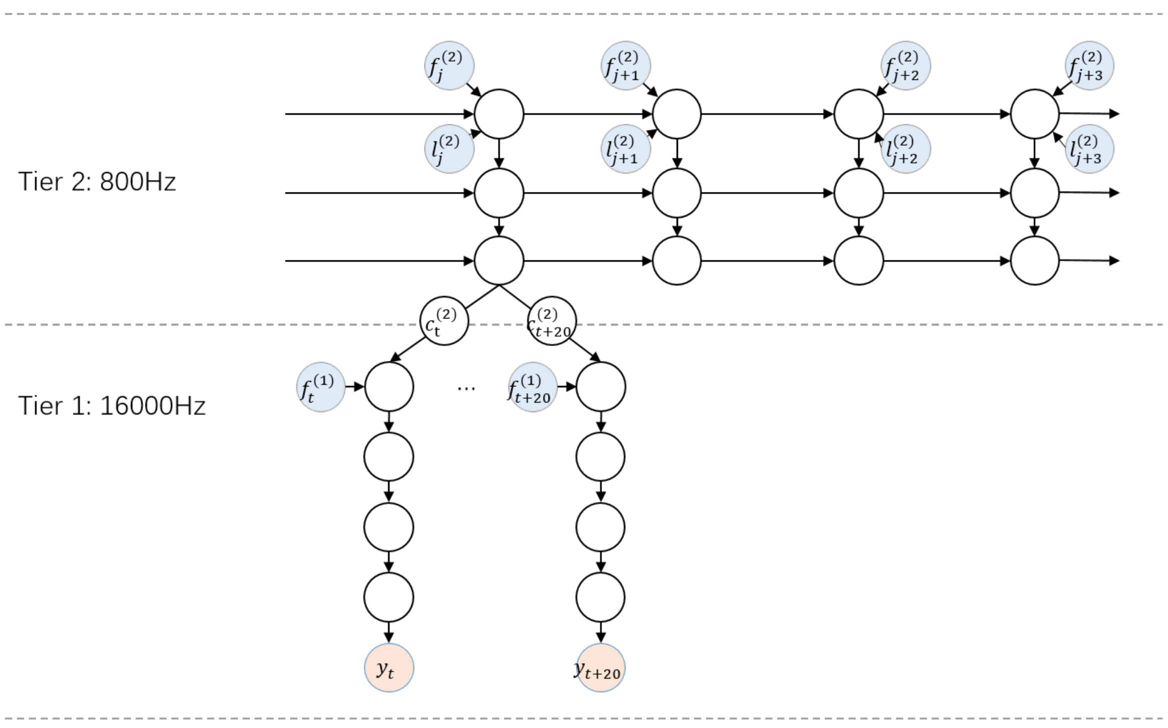


Fig. 5.2 visualization of an unrolled conditional hierarchical RNN with 2 tiers.

5.3 Results and analysis

5.3.1 2-tier conditional hierarchical RNN

First the default 2-tier conditional hierarchical RNN is trained. The model is pretrained for 8 epoches as if it were an unconditional synthesis model. Figure 5.3 shows the learning curves of this model. Training and validation costs are computed at each epoch; test cost is only computed when a new lowest validation cost is obtained. It can be seen that the testing curve converges after about 20 epochs, i.e. 100 hours. Figures 5.4 and 5.5 show the waveforms of the generated utterance and the corresponding recorded utterance at two different time scales. It can be seen that the generated waveform is similar to the reference waveform, but lacks variation.

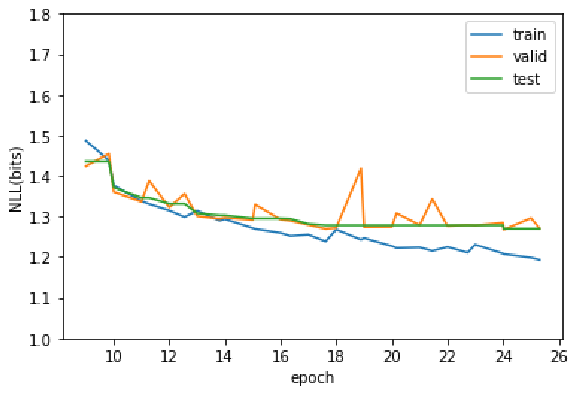


Fig. 5.3 learning curves of the default 2-tier conditional synthesis model

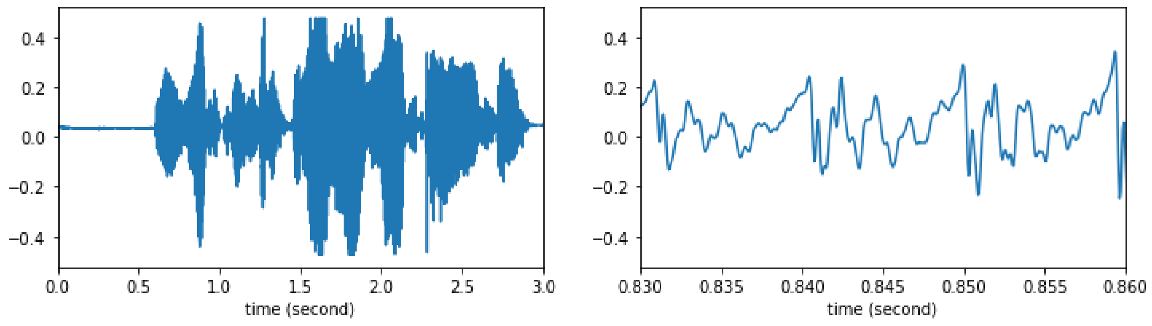


Fig. 5.4 waveform generated by the default 2-tier conditional synthesis model

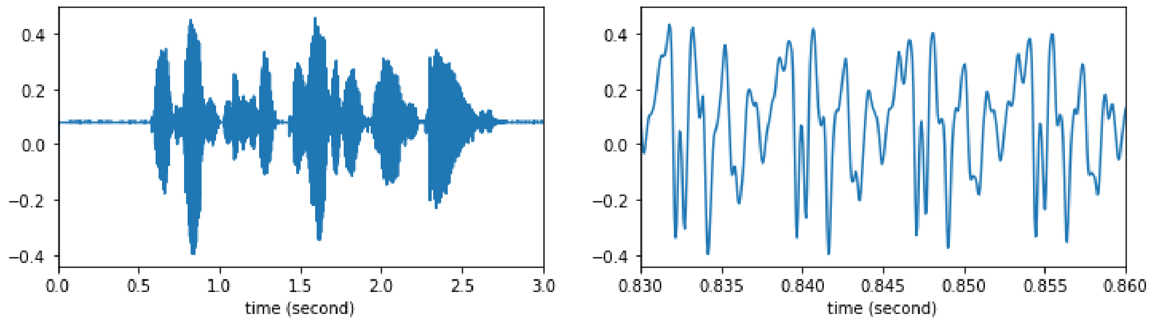


Fig. 5.5 waveform of the recorded utterance

Investigation on the effect of adding labels

The default 2-tier conditional synthesis model can be considered the conditional version of the 2-tier unconditional synthesis model shown in figure 3.8. Figure 5.6 compares these two models in terms of costs on training and validation data. The unconditional synthesis model has lower training cost and similar validation cost as the conditional synthesis model; however, this does not mean that it generates waveforms closer to the reference waveforms. Figure 5.7 shows the waveform generated by the unconditional synthesis model, which corresponds to the waveform in figures 5.4 and 5.5. It can be seen that the waveform generated by the conditional synthesis model is much closer to the reference waveform. This is because the costs shown by the learning curves are indicative of, but not equivalent to, the quality of generated waveforms. When computing the costs, at each time step a piece of reference waveform is fed into the network, and the network predicts the signal point for the next time step. At the next time step the input will again be from the reference waveform; even if the prediction is severely wrong, no error will be accumulated. In contrast, when generating, the output at one time step will be the input at the next time step, so error will be accumulated. For unconditional synthesis, error is accumulated and no correction is available; this is why at the beginning the waveform in figure 5.7 is relatively similar to the reference waveform in figure 5.5, but then it is less and less similar. For conditional synthesis, the label helps correcting previous errors made by the network, hence the waveform in figure 5.4 stays similar to the reference waveform.

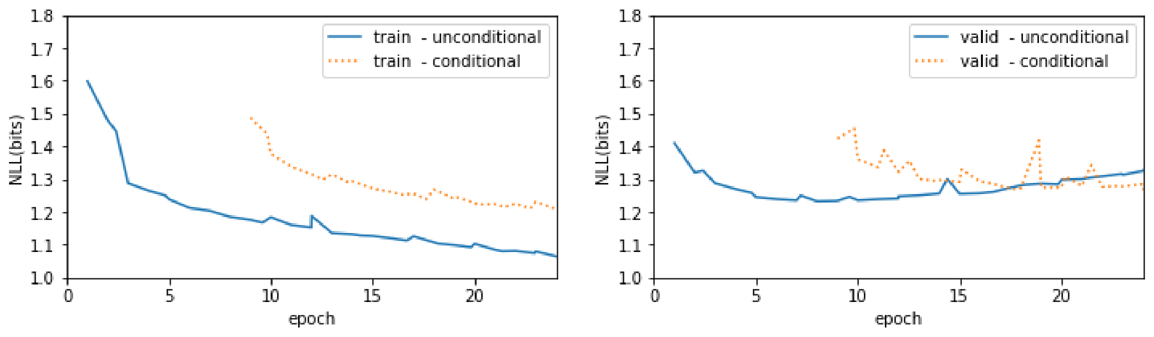


Fig. 5.6 costs for unconditional and conditional synthesis models

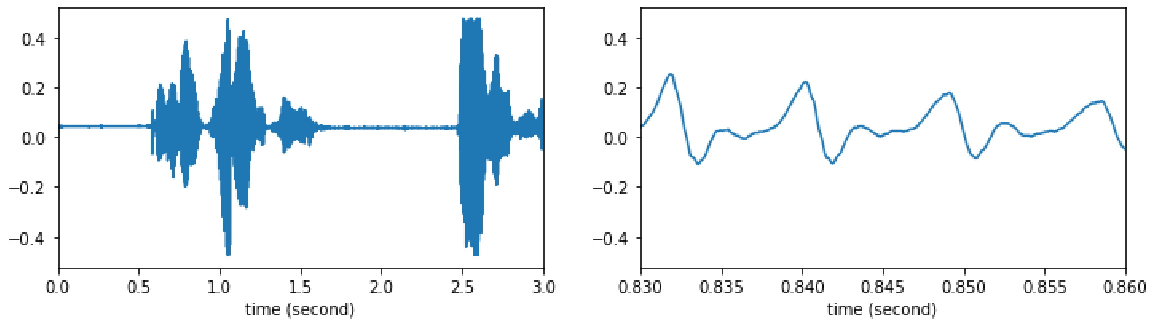


Fig. 5.7 waveform generated by the 2-tier unconditional synthesis model

Investigation on optimization

Another interesting observation from figure 5.6 is there is a big "jump" of training cost between conditional and unconditional synthesis models at epoch 9. For the conditional model, the first 8 epochs are pretraining epochs, during which the weights for labels are locked to 0, and cost of the conditional model is exactly the same as the unconditional model. At epoch 9, the weights are unlocked, but the cost becomes much higher. There are two major reasons. First, when pretraining ends, the weights for the 601-dimensional labels are no longer locked, so the cost function changes. Second, the Adam optimizer used for SGD is reset when pretraining ends. Intuitively, the Adam optimizer adjusts the learning rate of SGD by considering the first and second order momenta, which represent the history of optimization. When pretraining ends, the cost function changes and optimization starts again from a different point in a higher-dimensional space. To verify the above analysis, different learning rates are used to train the same conditional synthesis model. Figure 5.8 shows the training curves of the unconditional model (solid line) and conditional models (dashed lines) with different learning rates. As expected, when the learning rate is smaller, the "jump" of training cost between conditional and unconditional models at epoch 9 will also be smaller.

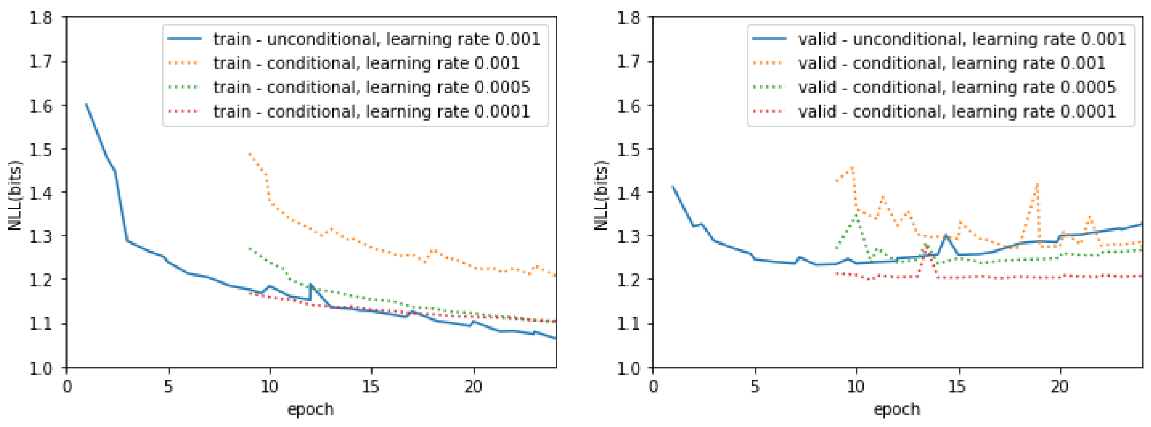


Fig. 5.8 costs for different learning rates

Investigation on pretraining

To investigate the effect of pretraining, a 2-tier conditional hierarchical RNN is built without pretraining. Figure 5.9 compares the costs on training and validation data for the experiment with pretraining and the experiment without pretraining. It can be seen that pretraining significantly reduces the costs on both training and validation data. When played, the waveforms generated from the experiment with pretraining sound much better than the experiment without pretraining. As analyzed previously, gradient-based method is prone to converging to a bad local minimum, especially when there many parameters. With pretraining, the conditional model starts from a good point in the parameter space, and is less likely to converge to a bad point.

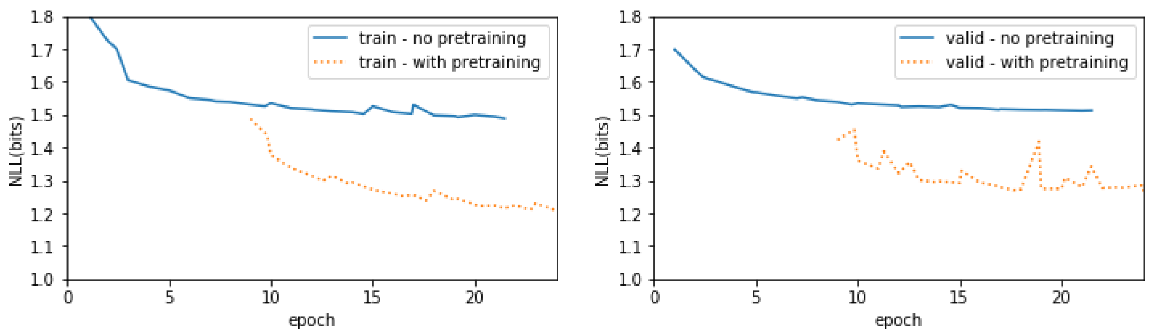


Fig. 5.9 costs for experiment with pretraining and the experiment without pretraining

Investigation on the length of TBTT

Truncated backpropagation through time (TBTT) is applied to train hierarchical RNNs more efficiently. Each long sequence is split into short subsequences, and gradients are only back-propagated to the beginning of each subsequence. To investigate the influence of the length of TBTT, a 2-tier conditional hierarchical RNN is trained, and the length of subsequence is 2000, instead of the default value 800. Figure 5.10 shows the training and validation costs for different TBTT lengths. It can be seen that when TBTT length increases, training cost increases slightly, and validation cost stays at the same level. As analyzed previously, when costs are close, the model with lower cost is not necessarily better. Figure 5.11 shows the waveform of the utterance generated with longer TBTT. Compared to the waveform in figure 5.4, this waveform is slightly more similar to the reference waveform, shown in figure 5.5. When played, the waveform generated with longer TBTT is more intelligible. In general, if the subsequence length is increased, the synthesis system performance will be improved, but the memory usage and convergence time will also increase.

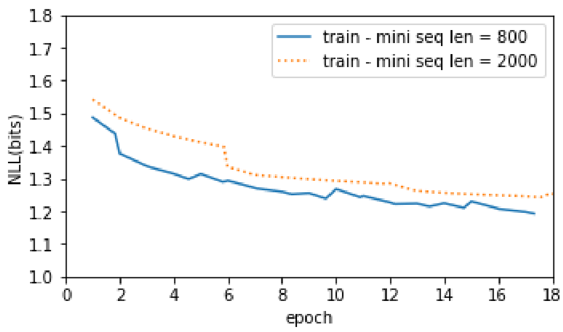


Fig. 5.10 costs for 2-tier conditional synthesis models trained with different TBTT lengths

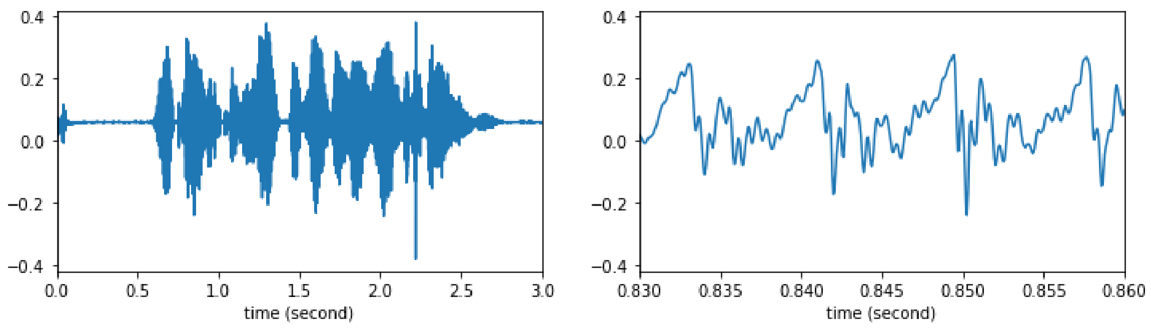


Fig. 5.11 waveform generated by the 2-tier conditional synthesis model trained with longer TBTT

5.3.2 3-tier conditional hierarchical RNN

Investigation on the number of tiers

To investigate the influence of the depth of the network, and to improve the performance of conditional synthesis, a 3-tier conditional hierarchical RNN is trained. Its structure is shown in figure 4.2. $r^{(3)} = 4$, $r^{(2)} = 20$, $FS^{(1)} = FS^{(2)} = 20$, and $FS^{(3)} = 80$. Figure 5.12 shows the learning curves of this model. Again the testing curve converges after about 20 epochs, i.e. 100 hours. Figure 5.9 compares the 3-tier and 2-tier models in terms of costs on training and validation data. It can be seen that the 3-tier model has slightly higher costs. When played, the waveforms generated from the 3-tier model sound slightly noisier but are considerably more intelligible. Figure 5.14 shows the waveform of the utterance generated from the 3-tier model. Compare to the waveform generated by the 2-tier model, shown in figure 5.4, this waveform is much more similar to the reference waveform, shown in figure 5.5. As analyzed previously, when there are more tiers, audio structures are modeled at more time scales. Tier 2 operates at 800Hz, which is at phoneme-level. Tier 3 operates at 200Hz, which is at word-level. Adding tier 3 allows the model to capture word-level audio structures.

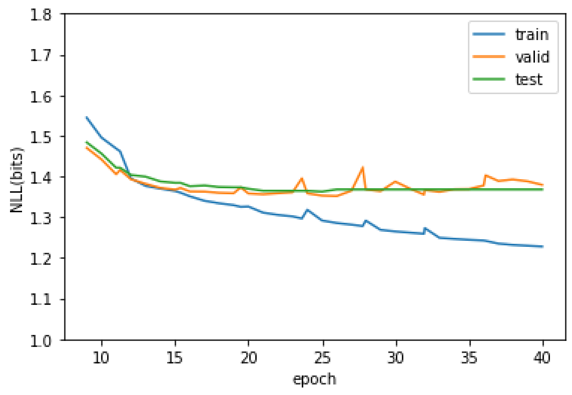


Fig. 5.12 learning curves of the 3-tier conditional synthesis model

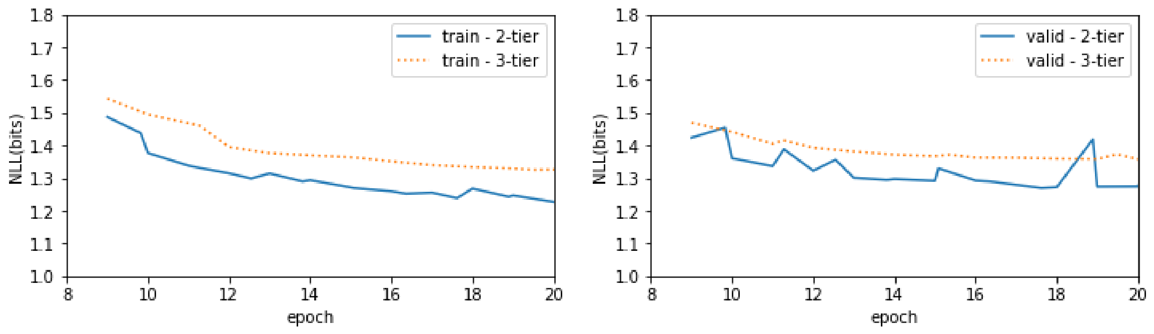


Fig. 5.13 costs for 2-tier and 3-tier conditional synthesis models

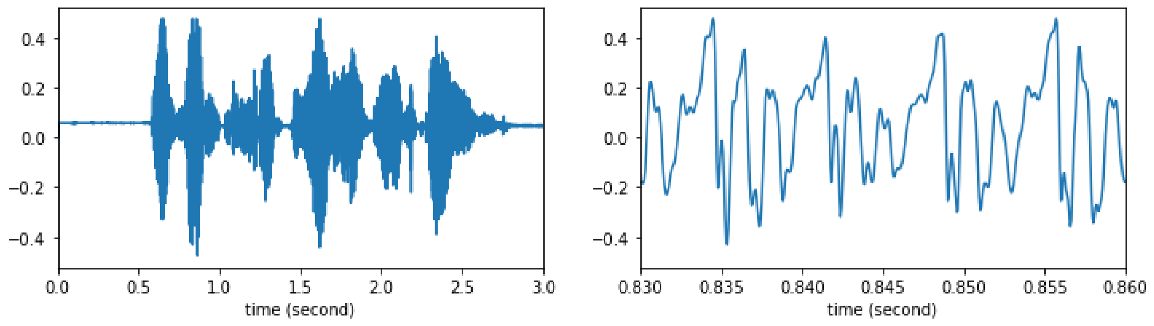


Fig. 5.14 waveform generated by the 3-tier conditional synthesis model

5.3.3 Baseline feature-level synthesis model

To compare waveform-level synthesis and feature-level synthesis, the baseline model shown in figure 5.1 is also trained. Figures 5.15 shows the waveform of the generated utterance at two different time scales. As the baseline system automatically cuts the silence at the beginning and the end, the waveform is padded with zero in figure 5.15 to facilitate comparison. It can be seen that the generated waveform is similar to the reference waveform in figure 5.5. Comparing the waveforms generated by the 3-tier waveform-level synthesis model and the feature-level synthesis model, it can be seen that the two models have comparable performance. However, it should be noted that compared to the feature-level synthesis model, the waveform-level synthesis model is almost not tuned at all.

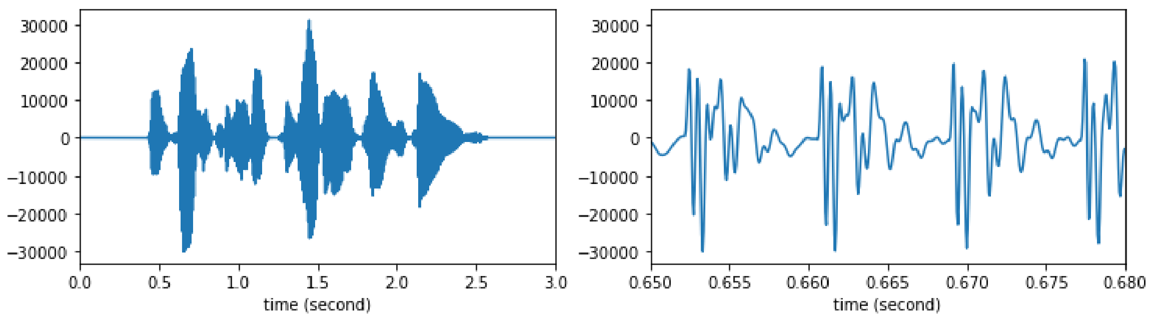


Fig. 5.15 waveform generated by the feature-level synthesis model

5.3.4 Feature-domain analysis

Comparing waveforms directly in time-domain can be difficult. If two waveforms are similar but there is a phase shift, the RMSE can be so high that no similarity is indicated. To solve this problem, vocoder features are extracted from the reference and generated waveforms, and RMSE is computed between the feature trajectories. This allows a better comparison between waveforms, and analysis on synthesis models from more aspects.

In addition to the baseline feature-level synthesis model, three waveform-level synthesis models are analyzed in feature-domain: the default 3-tier model, the default 2-tier model and the 2-tier model trained with longer TBTT. Figure 5.16 visualizes feature-domain analysis. It shows the trajectories of mel-cepstrum (dimension 1), aperiodicity (dimension 1) and log(F0) extracted from the waveform generated by the 3-tier model, and compares them to the corresponding trajectories extracted from the reference waveform. Other models are analyzed in the same fashion. It can be seen that in general all three trajectories follow the same trends as the references, and most errors are in the silent region at the beginning of the utterance.

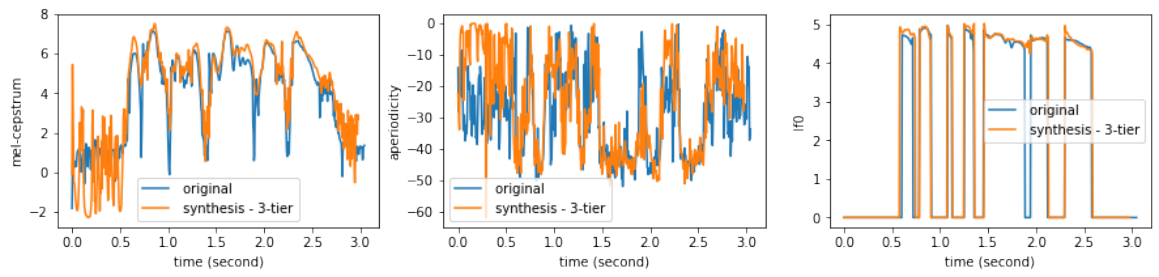


Fig. 5.16 trajectories of vocoder features

To perform more quantitative analysis, table 5.1 shows the added RMSE for three types of vocoder features: mel-cepstrum, aperiodicity and log(F0). For the first two types there are multiple dimensions, and the RMSE for each dimension is added to show a general RMSE level. Although the scale / 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. When computing RMSE, if two trajectories are not equally long, they will be synchronized by phase shifting and zero-padding in a way that the resulting RMSE is minimal. It can be seen that the 3-tier model is generally he best among waveform-level synthesis models: it has the lowest added RMSE for mel-cepstrum and log(F0), and its added RMSE for aperiodicity is very close the best value achieved. Comparing the two 2-tier models, it can be seen that training with a long TBTT improves log(F0) but degrades aperiodicity. Comparing the 3-tier model and the baseline model, it can be seen that the waveform-level synthesis achieves performance as good as the feature-level synthesis in terms of mel-cepstrum and log(F0), and it achieves better performance in terms of aperiodicity.

Table 5.1 added RMSE for three types of vocoder features

	mel-cepstrum	aperiodicity	log(F0)
3-tier	6.93	141.66	1.05
2-tier	7.77	137.08	1.86
2-tier, longer TBTT	7.78	149.03	1.34
baseline	7.06	176.89	0.96

A major reason why the 3-tier model performs better than other HRNN-based models is that it captures audio structures at different frequencies. On top of the 2-tier model with tiers operating at 16000Hz and 800Hz, the 3-tier model has an extra tier operating at 200Hz. Figure 5.17 shows the RMSE for each of the 60 dimensions of mel-cepstrum. Lower dimensions correspond to lower frequency ranges. It can be seen that the 3-tier model performs much better at low frequency range, and at high frequency range the 2-tier model is equally good.

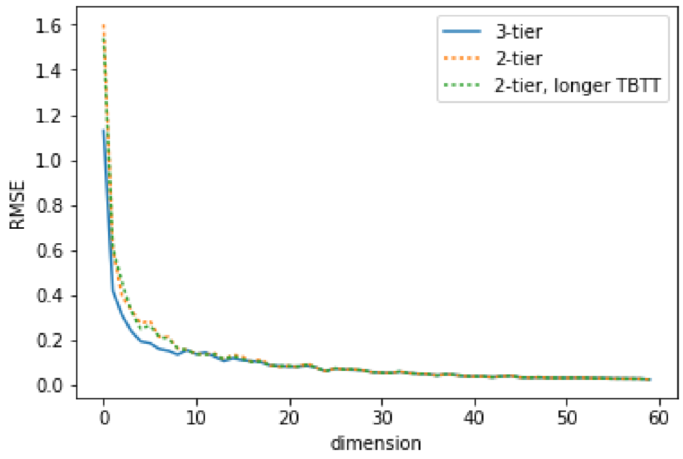


Fig. 5.17 RMSE for individual dimensions

A common problem of parametric speech synthesis is that the feature trajectories are overly smooth. Table 5.1 shows the added variance for three types of vocoder features; the variance of the reference trajectories are is also shown for comparison. For log(F0), all models have similar variance as the reference. For mel-cepstrum, the 2-tier model trained with longer TBTT performs slightly better than other models, and has a variance very close the reference variance. For aperiodicity, all the waveform-level synthesis models have much better performance than the feature-level synthesis model, which is expected as the feature-level model has much higher added RMSE for aperiodicity. Comparing the 3-tier model and the baseline model, it can be seen that the waveform-level synthesis model tends to generate waveforms that are less smooth in feature-domain, except for aperiodicity where the feature-level synthesis model seems to have a problem.

Table 5.2 added variance for three types of vocoder features

	mel-cepstrum	aperiodicity	log(F0)
reference	7.67	903.60	5.24
3-tier	8.34	899.47	5.45
2-tier	6.46	879.40	5.24
2-tier, longer TBTT	7.71	919.45	5.48
baseline	6.75	1535.94	5.14

Chapter 6

Conclusion and future work

6.1 Conclusion

In this thesis, two unconditional waveform-level synthesis models are investigated, namely hierarchical RNN and dilated CNN. Detailed analysis is made about how they represent history and target waveform. Experiments on hierarchical RNN are performed, with both piano music data used by Mehri et al. [15] and Nick speech data. The experimental results in section 3.3 indicate that the model works well in both cases, and that the speech data is more difficult to model than the music data. It is also found that the number of layers in a tier should be consistent with the upsampling rate of that tier; a deeper network does not always perform better.

For conditional waveform-level synthesis, two approaches of incorporating text information are investigated: using standard text labels and using labels generated by a neural network with attention mechanism. A new conditional waveform-level synthesis model is developed, combining hierarchical RNN and standard text labels. The experimental results in section 5.3 indicate that the new model can achieve performance comparable to traditional feature-level synthesis models. In terms of RMSE and variance computed in feature-domain, the 3-tier waveform-level synthesis model performs better than the feature-level synthesis model. It is found that having multiple tiers in the hierarchy, operating at different time scales, significantly improves performance. The results also indicate that the RNN-based model can be well trained with TBTT, which improves training speed; increasing the length of TBTT does not necessarily improve performance.

6.2 Future work

One limitation of the current implementations of waveform-level synthesis models is the training speed. As the models are all based on giant neural networks, training can be very slow. For the hierarchical RNN, each epoch takes about 5 hours, when training is performed on GPU with about 3 hour's speech data. On average each model requires 20 epochs to be well trained, so within 3 months at most about 20 models can be trained. An important step for future work is to parallelize computation in the training process, and potentially also modify the network structure to improve training speed.

In the current structure of hierarchical RNN, the initial state of RNN h_0 is a learnable constant parameter. While this is much better than random initialization, h_0 can be made a learnable function of the input at the first time step. Moreover, van den Oord et al. [25] reported that feeding F0 into dilated CNN improves the performance. F0 information is currently not fed into the conditional hierarchical RNN, and can be incorporated in future experiments.

In terms of data proprocessing, the raw speech data are separate utterances, and silence can be relatively long at the beginning and the end. So silence is probably more common than any other non-silent phone. As generating non-silent phones is usually more important, in future experiments the silence in the data can be reduced to further improve the performance.

References

[1] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

[2] Bishop, C. (1994). Mixture density networks: Neural computing research group technical report ncrg/94/004. *Aston University*.

[3] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[4] Dosovitskiy, A., Springenberg, J. T., Tatarchenko, M., and Brox, T. (2017). Learning to generate chairs, tables and cars with convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(4):692–705.

[5] 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*.

[6] Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

[7] 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.

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

[9] Hunt, A. J. and Black, A. W. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 1, pages 373–376. IEEE.

[10] 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.

[11] Kang, S., Qian, X., and Meng, H. (2013). Multi-distribution deep belief network for speech synthesis. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8012–8016. IEEE.

[12] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[13] 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.

[14] Maia, R., Zen, H., and Gales, M. J. (2010). Statistical parametric speech synthesis with joint estimation of acoustic and excitation model parameters. In *SSW*, pages 88–93.

[15] Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. (2016). Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*.

[16] Moulines, E. and Charpentier, F. (1990). Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech communication*, 9(5-6):453–467.

[17] 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.

[18] Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

[19] 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.

[20] Sagisaka, Y. (1992). Atr v-talk speech synthesis system. *Proc. ICSLP, 1992*.

[21] 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.

[22] Sotelo, J., Mehri, S., Kumar, K., Santos, J. F., Kastner, K., Courville, A., and Bengio, Y. (2017). Char2wav: End-to-end speech synthesis.

[23] Theis, L. and Bethge, M. (2015). Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935.

[24] Tuerk, C. and Robinson, T. (1993). Speech synthesis using artificial neural networks trained on cepstral coefficients. In *EUROSPEECH*.

[25] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*.

[26] van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016b). Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798.

[27] Yoshimura, T. (2002). Simultaneous modeling of phonetic and prosodic parameters, and characteristic conversion for hmm-based text-to-speech systems. *PhD diss, Nagoya Institute of Technology*.

[28] Ze, H., Senior, A., and Schuster, M. (2013). Statistical parametric speech synthesis using deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7962–7966. IEEE.

[29] Zen, H., Braunschweiler, N., Buchholz, S., Gales, M. J., Knill, K., Krstulovic, S., and Latorre, J. (2012a). Statistical parametric speech synthesis based on speaker and language factorization. *IEEE transactions on audio, speech, and language processing*, 20(6):1713–1724.

[30] Zen, H., Gales, M. J., Nankaku, Y., and Tokuda, K. (2012b). Product of experts for statistical parametric speech synthesis. *IEEE Transactions on Audio, Speech, and Language Processing*, 20(3):794–805.

[31] Zen, H., Tokuda, K., and Black, A. W. (2009). Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064.

Appendix A

Recurrent neural network

A.1 Basic units

A.1.1 vanilla recurrent unit

$$h_t = f(W^f x_t + W^r h_{t-1} + b) \tag{A.1}$$

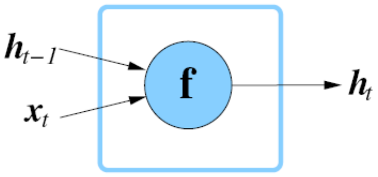


Fig. A.1 vanilla recurrent unit

A.1.2 GRU

$$i_f = \sigma(W_f^f x_t + W_f^r h_{t-1} + b_f) \tag{A.2}$$

$$i_o = \sigma(W_o^f x_t + W_o^r h_{t-1} + b_o) \tag{A.3}$$

$$y_t = f(W_y^f x_t + W_y^r (i_f \odot h_{t-1}) + b_y) \tag{A.4}$$

$$h_t = i_o \odot h_{t-1} + (1 - i_o \odot y_t) \tag{A.5}$$

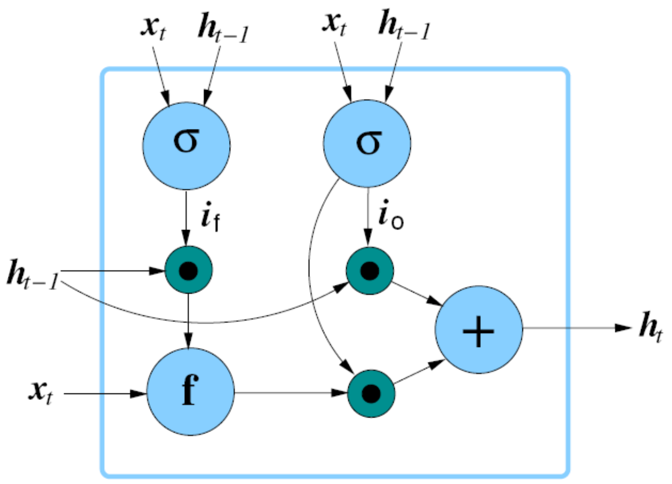


Fig. A.2 GRU

A.1.3 LSTM

$$i_f = \sigma(W_f^f x_t + W_f^r h_{t-1} + b_f + W_f^m c_{t-1}) \tag{A.6}$$

$$i_i = \sigma(W_o^f x_t + W_o^r h_{t-1} + b_o + W_i^m c_{t-1}) \tag{A.7}$$

$$i_o = \sigma(W_o^f x_t + W_o^r h_{t-1} + b_o + W_o^m c_t) \tag{A.8}$$

$$c_t = i_f \odot c_{t-1} + i_i \odot f^m(W_c^f x_t + W_c^r h_{t-1} + b_c) \tag{A.9}$$

$$h_t = i_o \odot f^h(c_t) \tag{A.10}$$

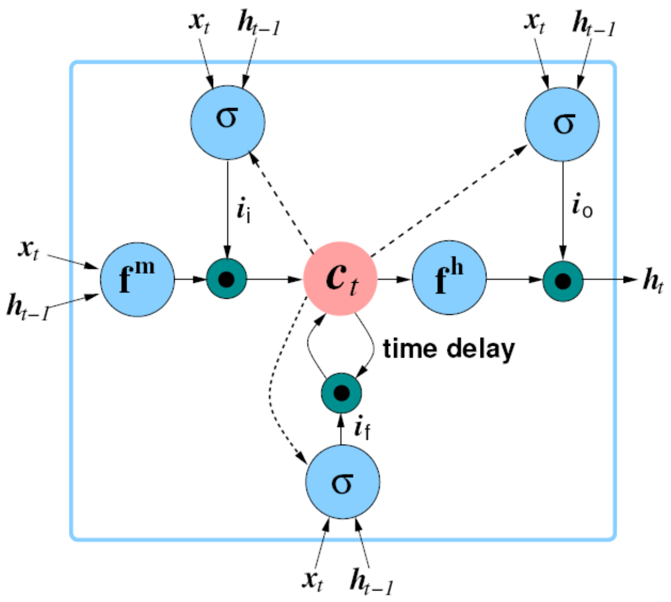

x_t
h_{t-1}
σ
i_i
i_o
$\mathbf{f}^{\mathrm{m}}$
c_t
$\mathbf{f}^{\mathrm{h}}$
h_t
time delay
i_f

Fig. A.3 LSTM