THE UNIVERSITY OF AIZU

MASTER'S THESIS

Integration of Articulatory and Spectrum Features based on Deep Learning Method

Author:
Jianguo YU

Supervisor:
Prof. Konstantin Markov

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of computer science and engineering

in the

Human Interface Laboratory
Graduate Department of Computer and Information Systems

August 25, 2016

The thesis titled

Integration of Articulatory and Spectrum Features based on Deep Learning Method

by

Jianguo Yu

is reviewed and approved by:

Date: 2016/08/25	Prof. Konstantin Markov マルコフ
Date: Aug. 25, 2016	Prof. Masahide Sugiyama [illegible]
Date: Aug. 25, 2016	Prof. Qiangfu Zhao 趙

THE UNIVERSITY OF AIZU

Abstract

Field of Study Applied Information Technologies
Graduate Department of Computer and Information Systems

Master

Integration of Articulatory and Spectrum Features based on Deep Learning Method

by Jianguo YU

Articulatory information is extremely helpful for automatic speech recognition (ASR) but not available during real-life recognition. In this paper, I propose four systems that integrate Articulatory information into ASR based on Deep Learning Method in a way that uses only acoustic feature during recognition. Two models trained with the guidance of a "teacher" model that contains both acoustic and articulatory information based on both Deep Neural Network (DNN) and Recurrent Neural Network (RNN) and Two models using articulatory inversion methods based on DNN and RNN. A new RNN architecture was also found to be more suitable for this task. For experiments, I used the Wisconsin X-ray Microbeam Database (XRMB) which consists of parallel articulatory and acoustic data. Results clearly show that all systems allow us to achieve significant reductions in the phoneme error rate (PER). The results can be even improved with a new RNN architecture proposed in chapter 4.

Acknowledgements

I still remember the first time I sent my supervisor Dr. Konstantin Markov an e-mail that I want to study speech recognition. However, I had nearly no basic knowledge needed for it. My life path has been changed since he agreed to accept me. Here, I would like to express my deepest sense of gratitude to my supervisor. Without his continuous advice and support, I couldn't have made my innocent wish come true.

I would like to express my very sincere gratitude to Prof. Masahide Sugiyama who helped me a lot with daily life in Japan.

I am also very thankful to my labmate Daria Vazhenina who has been a good senior and friend.

Finally, I take this opportunity to express the profound gratitude to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the development of Pocket-sized devices such as Google Glass, Watch Phones which tend to be too small to hold buttons or touchscreen for user input, Speech Recognition (Voice user interface) as an ideal substitute for traditional user interfaces has become the major breakthrough technology that the manufacturers are in badly need of, since natural language is the most appropriate way to manipulate machines for human and has inherent advantage in the situations where our eyes or hands are fully occupied on other things, for instance, driving a car.

1.1 Challenges in ASR

However, current state-of-the-art Automatic Speech Recognition (ASR) systems represent speech as a sequence of non-overlapping phonetic units while implicitly assuming that speech can be decomposed into disjoint acoustic segments, which limits the acoustic model's ability to properly learn the underlying variations in spontaneous or conversational speech. Although such systems perform fairly well for clearly articulated speech in "controlled" conditions, they suffer from acoustic variabilities in speech. Such variabilities can be posed by the variations in different speaking styles, talkers, contexts, background noises, reverberations, and recording device etc.

1.2 Articulatory Information

One solution to these problems is to incorporate articulatory information into ASR. Articulatory information is the movements of lips, jaw, and other speech organs when we are speaking. Fig.1.1

Articulatory information is especially useful for modeling the coarticulation. Coarticulation is a speech production effect which results in an assimilation where some sounds have influences on others (e.g. "p" sound in the word " speed" becomes "b" sound due to "s" sound.) To address this short coming, di-phone or tri-phone (several phones tied together as one unit) based acoustic models are often used, but they assume that coarticulation effects only impact neighboring phones which is not always true, as in practice, there are many instances where coarticulation effects extend beyond the immediate neighbors. While articulatory information can model coarticulation effectively since it doesn't rely on the "non-overlapping phonetic units" assumption. Here are some advantages of using articulatory information:

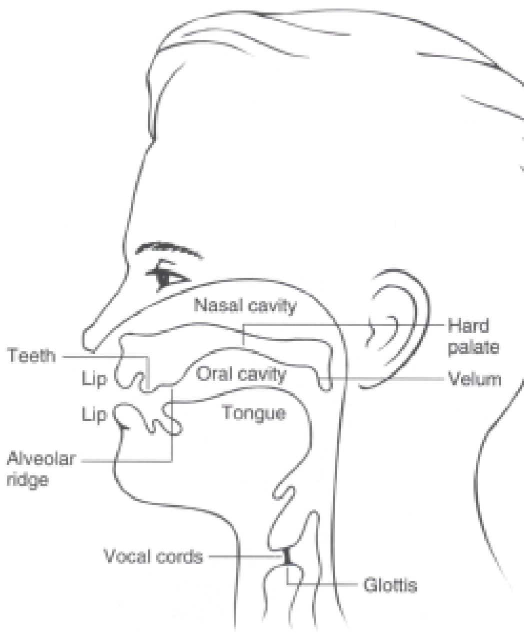


FIGURE 1.1: *Articulatory information*

1. Articulatory factors are much less affected by environmental conditions.
2. It can make systems more robust.
3. It can preserve information that tends to be lost during the extraction of speech acoustic features.
4. It can also help model coarticulation in a more systematic way rather than using tri- or quin-phone acoustic models.

An abundant of prior studies like [19] [22] have proved that articulatory information can improve the performance of ASR systems. However, because it is nearly impossible to obtain observations of articulator movements during real-life recognition, people have to come up with systems that use acoustic feature during recognition and make the effort to integrate articulatory information into ASR systems when training the systems.

1.2.1 Feature Representations

Articulatory Information is essentially images through time. The feature representations of it can be images, outlines, points. For example, the articulatory representation in the MOCHA-TIMIT database captured using Electromagnetic articulography (EMA) is the trajectories of points and the one in the USC-TIMIT database obtained using Real-time magnetic resonance imaging (rtMRI) is the video of the vocal tract during speaking. Because we want to use it along with acoustic features it's necessary to change the sampling rate of articulatory features to match the sampling rate of acoustic features.

Trajectories of Articulators
Although images naturally contain the maximum information. We can also reduce the size of data by just looking at the critical points since most of the information is carried by the critical articulators (tongue, lips, and jaw and so on) as shown in Fig.5.1. Unfortunately, such absolute measure can be inconsistent and may suffer from variability.

Tract Variables and Articulatory Gestures
Tract variables and articulatory Gesture [24] representation are relative measures and hence should be invariant. Articulatory phonology treats each word as a constellation of vocal tract constriction actions, called gestures (roughly 1 to 3 gestures for each of the phones in a phonetic transcription). Although it can model the coarticulation [11] simply and elegantly, there is no existing database of it. Therefore, it's usually synthetic data or data estimated from other databases, such as the procedure described in [25].

1.3 Articulatory and Spectrum Feature integration

As mentioned above, in practice articulatory is not available for recognition. This is the main obstacle to utilizing such useful information.

1.3.1 Feature Based

One way to address this is to generate articulatory features given the corresponding acoustic speech signal, known as *acoustic-to-articulatory mapping or articulatory inversion*[28]. Articulatory inversion suffers from non-uniqueness, which stems from the fact that many different articulatory configurations result in similar acoustic properties. Many machine-learning methods have been attempted to do the articulatory inversion, due to the difficulty of it, many have been limited to speaker-dependent inversion. In [46], a hidden Markov model (HMM) approach to articulatory movement prediction from speech was presented, which adopted a similar framework to HMM-based parametric speech synthesis. In [36] a Gaussian mixture model for the joint distribution of acoustic and articulatory features was adopted to achieve the mapping from acoustic features to articulatory features. Currently, Artificial neural networks (ANNs) have been proven to be quite effective in regression tasks. In [38], many feedforward neural networks were investigated to find the optimal architecture. However, feedforward neural networks often use pre-defined fixed-length context window of acoustic signals as input to estimate articulatory movement, which may be either insufficient or redundant to cover such correlation information. In [17], a Bidirectional Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) have achieved the state-of-the-art result in speaker-dependent articulatory inversion. The advantage of recurrent neural network is that it can learn proper context information on its own without the requirement of externally specifying a context window. [47] also shown that RNN nets can preform better than feedforward nets.

1.3.2 Model Based

Anther direction is to attempt to embed the articulatory information inside the model and leave it hidden (i.e., implicitly predict it) at test time. For example, in Markov et al. [19] described a hybrid HMM and Bayesian network (BN) model, where the BN described the dependence of acoustics on quantized EMA articulations, and of quantized EMA articulations on phones. They reported that the HMM-BN trained using articulatory information always performed better than the baseline HMM system trained only with the acoustic features. In [23], Mitra et al. proposed a Gesture-based Dynamic Bayesian Network (G-DBN) architecture that uses the articulatory gestures as hidden variables, eliminating the necessity of explicit recognition of articulatory gestures. In [2], a MULTI-VIEW method based on canonical correlation analysis (CCA) is proposed, which finds pairs of maximally correlated linear projections of data in two the views.

Chapter 2

Deep Neural Networks

Deep Neural Networks (DNNs) have recently achieved breakthrough results in almost every machine-learning task. They are a set of machine-learning algorithms inspired by how the brain works. Unlike most traditional machine-learning algorithms, Deep Neural networks perform automatic feature extraction without human intervention.[32]

2.1 Feed-Forward DNN

Two perspectives to understand the behavior of DNNs are shard here.

Linearly Separable
A toy DNN shown in Fig.2.1 It is challenging to understand the behavior of

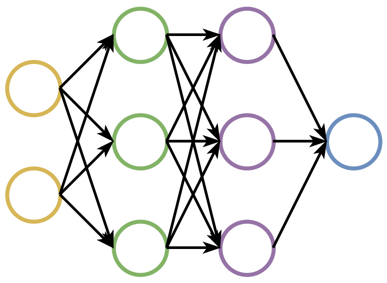


FIGURE 2.1: *simple DNN*

deep neural networks in general, but it's much easier to see what's happening when the data is passed through a single layer: A mapping from input space to output space. It is basally **linear transformation[7] followed by an activation function**, whose mathematical description is Eq.(2.1), where $\vec{x}$ is the input, W is weights matrix, $\vec{b}$ is bias, $a()$ is activation function and $\vec{y}$ is the output of this layer.

$$\vec{y} = a(W \cdot \vec{x} + \vec{b}) \tag{2.1}$$

If we break down the equation, the single layer actually transforms the data and create a new representation by the following 5 space operations.

1. Change the dimensionality of the input space.
2. Rotate the input space.
3. Scale the input space.

4. Translate the input space.
5. "bend" the input space.

The first 3 operations are done by $W \cdot \vec{x}$, the 4th one is provided by $\vec{b}$, and the 5th operation is done by $a()$ which gives nonlinearity to the layer.

Take the example of the simple classification case shown as Fig.2.2. It's impossible to separate the two curves with a line.

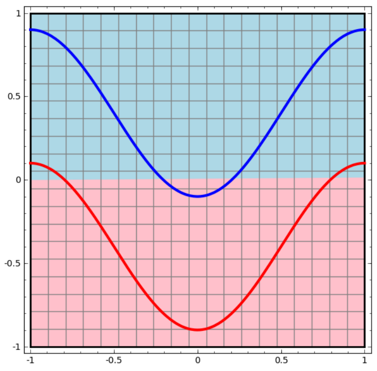


FIGURE 2.2: *simple case*

If we increase the number of layers, the 5 operations can be applied again and again so that we can finally find a hyperplane that separates the two curves in the final representation. Fig.2.2 shows the final representation after these space operations of several layers.

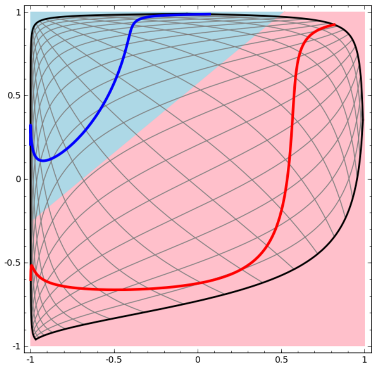


FIGURE 2.3: *After operations of several layers*

From this perspective, the parameters of DNN indicate operations applied to the input space and the learning process of DNN is to make the input space linearly separable with the interactions of 5 space-operations of several layers.

How matters form

Another perspective is about how a matter forms. We know that matters are composed of molecules which are again composed of atoms. Each layer of DNN can represent a level of matter (such as atom level or molecule level). If there are a certain amount of different atoms, they can form various molecules and eventually form a car. We can understand DNN with the similar concept, then the parameters of DNN indicate how, let's say, a car is composed of atoms. Fig.2.4 demonstrates that with the different combinations of pixels, we can get various edges, with different combinations of edges we get noses or ears and eventually we get all kinds of faces. Once we know how a certain picture forms from pixels, we can tell whether a combination of new pixels is that picture. *More about what DNNs do:* [26]

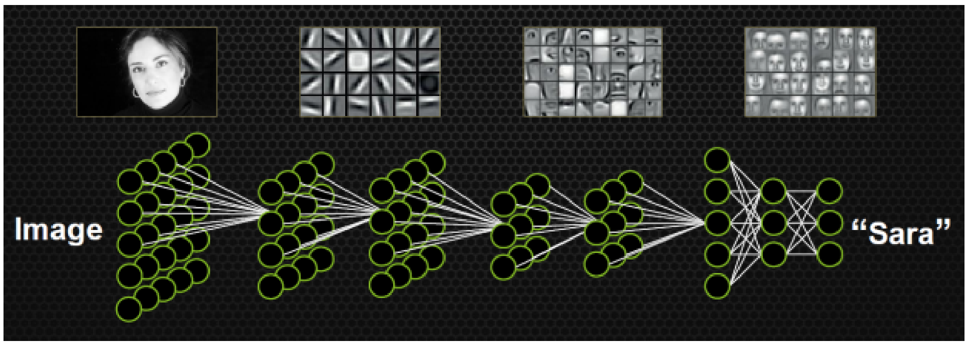


FIGURE 2.4: *How a picture of face is composed of pixels*

The learning process of DNN is to transform the input space to a linearly separable space with the interactions of 5 space-operations of several layers. But how do we train it?

2.1.1 Data Pre-Processing

Like other machine learning methods, pre-processing is needed before training.

Mean Subtraction
Mean subtraction is the most common way which can make it easy for the network to converge. It's just subtracting the mean across every individual feature in the data and it can center the cloud of data around the origin along every dimension.

Normalization
Normalization refers to normalizing the data dimensions so that they are of approximately the same scale. There are two common ways of achieving this:

1. Divide each dimension by its standard deviation, once it has been zero-centered.
2. Normalize each dimension so that the min and max along the dimension is -1 and 1

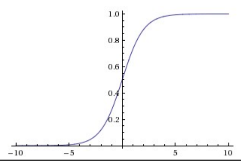
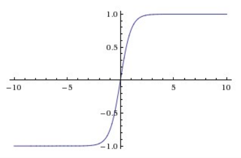
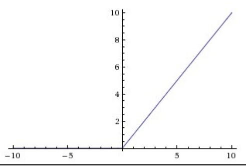
Notes that the outputs should also correspond the activation function of the output layer. For example, if we use **sigmoid** activation function, then the range of outputs should be $(0, 1)$.

One-Hot Vector
If the task is classification, then instead outputting a most likely class, we also want to know the probabilities being other classes. Then, we need to convert our labels to one-hot vectors of size **number of classes**. For example, the class with index 12 would be the vector of all 0's and a 1 at position 12.

2.1.2 Non-Linearities

The Non-linearity of the network is given by activation function. Commonly used activation functions are **Sigmoid**, **Tanh**, **ReLU**. Their details are shown in Table.1 2.1 which one to use depends on the task. For example, the gates in a Recurrent Neural Network are a Neural Network with output layer of Sigmoid, because this network controls how much information can flow. *Also see how to choose an activation function:* [21]

TABLE 2.1: *Details of activations*

Name	Plot	Equation	Range
sigmoid		$1/(1+e^{-x})$	$(0,1)$
tanh		$2/(1+e^{-2x})-1$	$(-1,1)$
ReLU		$max(0,x)$	$[0,\infty)$

Generally, ReLU is much faster than the other two. It is argued that this is due to its linear, non-saturating form. Compared to tanh/sigmoid that involves expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero. Unfortunately, ReLU has the "dying ReLU" problem. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. By setting learning rate smaller and proper Initializing of weights can somewhat help to avoid it.

Another very important activation function is the **softmax** function, which is usually used as the activation function of the output layer for classification tasks. It is a generalization of the logistic(sigmoid) function that gives the probabilities being classes and makes the sum of them equal to 1. The equation of softmax is given by Eq.(2.2) where K is the total number of neurons in the layer. This activation function gets applied row-wise (the sum of the row is 1 and each single value is in $[0, 1]$).

$$\phi(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{2.2}$$

2.1.3 Loss Functions

We train a network by minimizing the error the current network makes. Therefore, first, we need to define the error. The function that measures the error is called the **loss function**.

Classification

For the multi-class classification tasks, crossentropy and softmax output activation function come with the pair. It's given by Eq.(2.3)

$$L_i = -\sum_j t_{i,j} \log(p_{i,j}) \tag{2.3}$$

Regression

For the regression tasks, we hope the predictions and targets are as close as possible, therefore, the loss function can be any type of distance between them, like **squared error(SE)**, which is given by given by Eq.(2.4)

$$L = |t - p|^2 \tag{2.4}$$

2.1.4 Updates

Once we have the loss function, we can update the parameters of DNN by minimizing the error got from loss function.

Backpropagation

Backpropagation is the game changer that makes training deep models computationally trainable. The earliest deep-learning-like algorithms that had multiple layers of non-linear features can be traced back to Ivakhnenko and Lapa in 1965 and the earliest convolutional networks were used by Fukushima in 1979. However, backpropagation was lacking at this point. Although it was derived already in the early 1960s but in an inefficient and incomplete form. Backpropagation can make training with gradient descent as much as ten million times faster. Backpropagation is essentially the chain rule that starts backward from the error. It's just a technique for calculating derivatives quickly. The real difference it made is that it enables us to train a model within a week than many years.

Gradient Descent

The method used in conjunction with Backpropagation for finding the minimum of loss function is Gradient descent. Generates update expressions of Eq.(2.5). It takes steps proportional to the negative of the gradient as , because we want to minimize the loss function.

$$param = param - learningrate * gradient \tag{2.5}$$

Depending on the **Size of examples** in each iteration, the name will also change:

1. **Stochastic Gradient Descent (SGD)**: one example from training set in each iteration.
2. **Mini-batch gradient descent**: m examples from training set in each iteration and the gradient will be averaged over m examples.

Unlike vanilla Gradient descent that runs through **all** samples in the training set to do a single update for a parameter in a particular iteration, SGD often converges much faster. Note that sometimes people use the term SGD even when referring to mini-batch gradient descent. The size of the mini-batch is a hyperparameter but it is not very common to cross-validate it. It is usually based on memory constraints. We use powers of 2 in practice because many vectorized operation implementations work faster when their inputs are sized in powers of 2. The smaller size tends to give more generalization because it will not fit the training set too well in each iteration.

Gradient Descent also has a problem. Because it is a first-order optimization algorithm that finds a local minimum of a function, it can get stuck in local minima and fail to reach the global minima as shown in Fig.2.5.

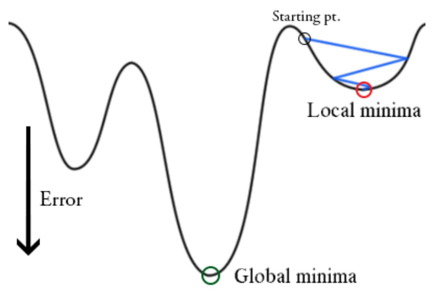


FIGURE 2.5: *get stuck in local minima*

Learning Rate

The learning rate determines to what extent the parameters will update (also called step size). A learning rate of 0 will not learn anything. The learning rate is also a hyperparameter. Fig.2.6 shows how the learning rate affects the decay of loss function. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape.

Therefore, it's useful to decay the learning rate during training. However, this can be tricky because decay it slowly and you'll be wasting computation bouncing around chaotically with little improvement for a long time. But decay it too aggressively and the system will cool too quickly, unable to reach the best position it can.

Momentum

Another technique that can help the network out of local minima is the use

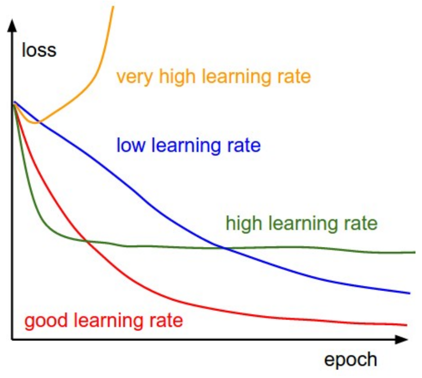


FIGURE 2.6: *How the learning rate affects the decay of loss function*

of a momentum term. Momentum simply adds a fraction m of the previous weight update to the current one. When the gradient keeps pointing in the same direction, this will increase the size of the steps taken towards the minimum. It is therefore often necessary to reduce the global learning rate when using a lot of momentum ($momentum$ close to 1). If you combine a high learning rate with a lot of $momentum$, you will rush past the minimum with huge steps. This is different from the GD update shown above, where the gradient directly integrates the position. Instead, the physics view suggests an update in which the gradient only directly influences the velocity,Eq.(2.6, 2.7) which in turn has an effect on the position. The parameter momentum is usually set to values such as $[0.5, 0.9, 0.95, 0.99]$ and velocity is initialized as 0.

$$velocity = momentum * velocity - learningrate * gradient \tag{2.6}$$

$$param = param + velocity \tag{2.7}$$

Nesterov Momentum

Nesterov Momentum is a slightly different version of the momentum update has recently been gaining popularity. Fig.2.6 shows the difference from the momentum update. It generates update in a way described as Eq.(2.8, 2.9). Also see [3] for further reading.

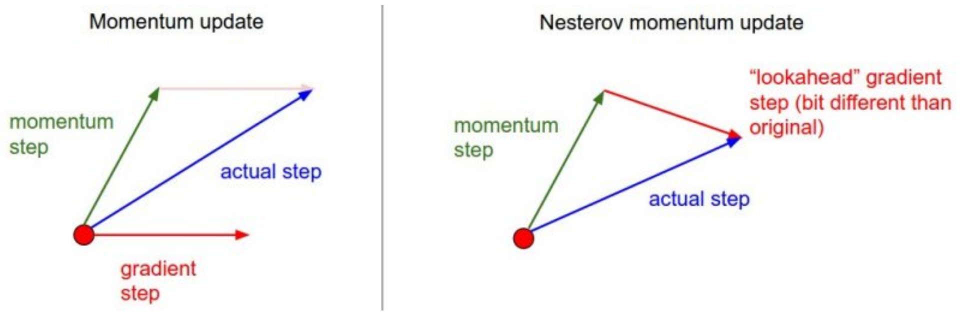


FIGURE 2.7: *With Nesterov momentum we evaluate the gradient at this "looked-ahead" position.*

$$velocity = momentum * velocity - learningrate * gradient \tag{2.8}$$

$$param = param + momentum * velocity - learningrate * gradient \tag{2.9}$$

Adaptive Updates
A bunch of update methods that adapt the learning rate and use momentum exist now. New update methods are also coming to us. Some of them are *Adagrad, Adadelta, RMSprop, Adam*[15], *Hessian-free*[20]. Which optimizer to use? In practice **Adam** is currently recommended as the default algorithm to use, and often works slightly better than **RMSProp**. However, it is often also worth trying **SGD+Nesterov Momentum** as an alternative. A good blog that is worth reading: [31]

Initialization
Before we can begin to train the network we have to initialize its parameters. One thing we should not do is to set all the initial weights to zero. Because if every neuron in the network computes the same output, then they will also all compute the same gradients during backpropagation and undergo the exact same parameter updates. In other words, there is no source of asymmetry between neurons if their weights are initialized to be the same. The right way to do it instead is to initialize weights randomly with small values. There are many ways to do this as well. The recommended heuristic is to initialize each neuron's weight vector as Eq.(2.10, where n is the dimension of the input to current layer.

$$w = \frac{N(0, \mu)}{\sqrt{n}} \tag{2.10}$$

Overfitting
Deep neural networks with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks, which means the system not only learns the pattern of training set, but also the noise of it and such case, the system can only give good predictions for the training set, but worse for test set. Fig 2.8 demonstrates underfitting and overfitting, where degree 1 is underfitting and degree 15 is overfitting.

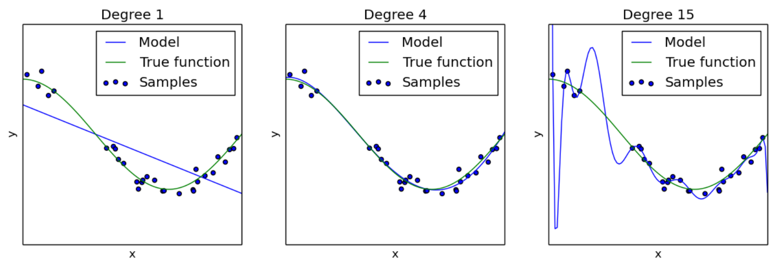


FIGURE 2.8: *underfitting and overfitting*

Train/Validation Losses A common way to see whether the model actually overfits is to split an additional set called validation set and calculate the accuracy for validation set as well. The gap between the training and validation accuracy indicates the amount of overfitting as shown in Fig 2.9

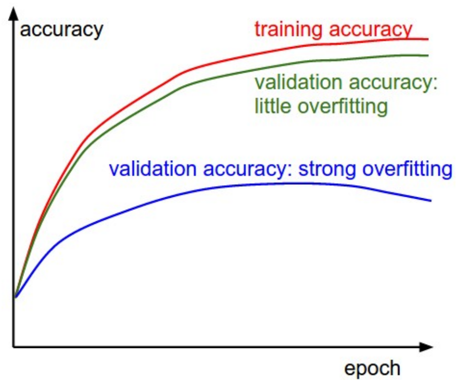


FIGURE 2.9: *The amount of overfitting can be noticed by analysis train/loss accuracies*

Add Noise To Training Set Since overfitting means the model fits the training set too well and give worse results for test data, one way to reduce this bad effect is to add noise to the training set.

L2 regularization Another common method is L2 regularization. If a deep neural network has a large number of parameters it's hard to guarantee that all parameters(nodes) have contribution to predictions. L2 regularization has the appealing property of encouraging the network to use all of its inputs a little rather than using only some of its inputs a lot. We do it by adding a penalty $1/2\lambda w^2$ to the loss function, where λ is the strength and adding $1/2$ is just for easy derivative calculation. Using the L2 regularization ultimately means that every weight is decayed linearly to zero. Because of this phenomenon, L2 regularization is also commonly referred to as *weight decay*.

Dropout Dropout is an extremely effective, simple and recently introduced regularization technique by Srivastava et al. in[33]. The key idea is to randomly drop units (along with their connections) from the neural network during training as shown in Fig 2.10. This prevents units from co-adapting too much. While training, dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise.

Early Stopping
Another straightforward way is Early Stopping, which just simply stops training once some bad conditions happens. Fig 2.11. For example, stop the training if the validation loss function increases for 3 epochs.

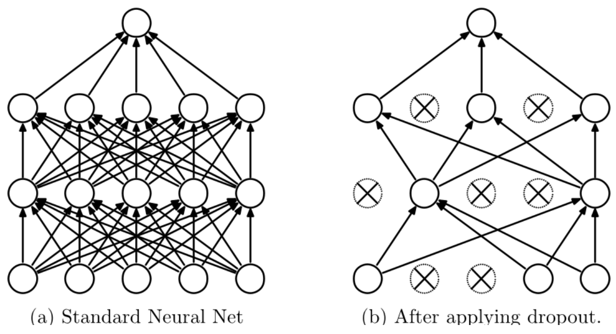


FIGURE 2.10: *Only apply dropout during training*

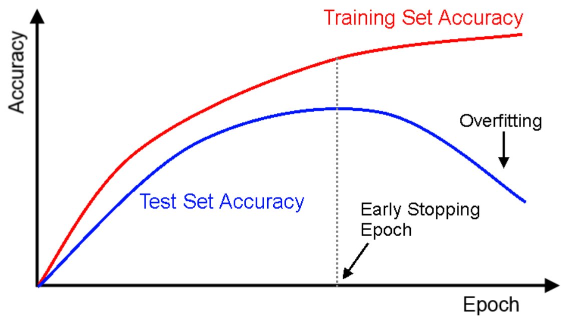


FIGURE 2.11: *Early Stopping*

2.2 Recurrent DNN

Although there are many kinds of Neural Networks, Recurrent Neural Networks (RNNs) are quite special, because in a Vanilla (Feed-Forward) Neural Networks (even Convolutional Networks) we assume that all inputs (and outputs) are independent of each other. If we want to understand a speech, we need to hear all the words to get the point the speaker wants to make. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop: Fig.2.12.

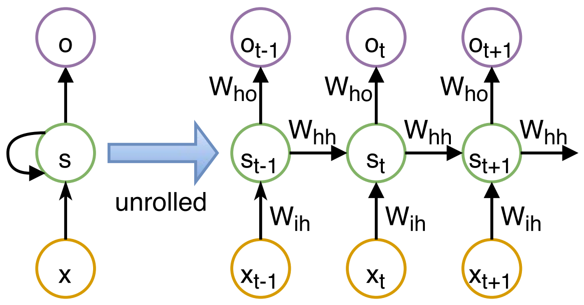


FIGURE 2.12: *An unrolled recurrent neural network*

The hidden state and output mathematical descriptions are Eq.(2.11) and Eq.(2.12). Compared to Feed-Forward Network, Recurrent Neural Network has "memory" ($\vec{s_t}$) which contains information about what happened in the previous time steps. The output at step $\vec{o_t}$ is calculated solely based on the memory at time t.

$$\vec{s_t} = tanh(W_{ih} \cdot \vec{x} + W_{hh} \cdot \vec{s_{t-1}} + \vec{b}) \tag{2.11}$$

$$\vec{o_t} = a(W_{ho} \cdot \vec{s_t} + \vec{b}) \tag{2.12}$$

From the equations we can also see that even the RNN has different structure, the basic linear transformation followed by an activation function remains. In other words, the basic idea behind every Neural Network is mappings between spaces.

With this structure, RNN can further allow us to map spaces between sequences of vectors, or in the most general case shown in Fig.2.13.

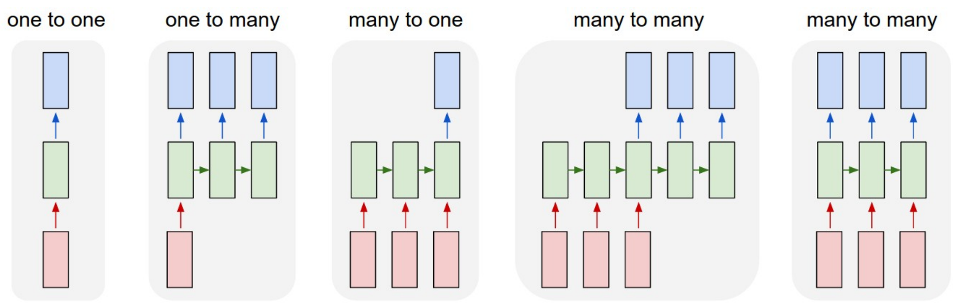


FIGURE 2.13: *Types of mapping*

Backpropagation Through Time

Recurrent networks rely on an extension of backpropagation called backpropagation through time (BPTT). As shown in Fig.2.14, to calculate gradient for x_3 we also need to apply chain rule to backpropagate gradients all the way to $t = 0$.

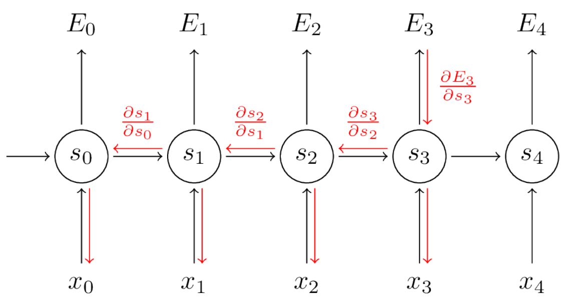

E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

FIGURE 2.14: *Backpropagation Through Time*

Vanishing and Exploding Gradients

The idea of Vanilla recurrent network is beautiful, but it has problems in practice: *Vanishing and Exploding Gradients*. The vanishing gradient problem was originally discovered by Sepp Hochreiter in 1991. The gradient for E_3 in Fig.2.14 is calculated as: Eq.2.13, from which we can see that there are two factors that affect the magnitude of gradients: the weights and the derivatives of their activation functions. If either of these factors is smaller than 1.0, the other gradients in previous layers will be driven towards 0. Thus, with small values in the matrix and multiple matrix multiplications the gradient values are shrinking exponentially fast, eventually vanishing completely after a few time steps. The information at those steps doesn't contribute to what you are learning. The fact is that you end up not learning long-range dependencies. Conversely, if the weights in this matrix are large it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as *exploding gradients*. For detailed explanation[29].

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left(\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_k} \right) \frac{\partial E_3}{\partial W} \tag{2.13}$$

On the other hand, the derivatives of activation functions like *Tanh/sigmoid* are smaller than 1.0 for all inputs except 0 as shown in Fig.2.15.

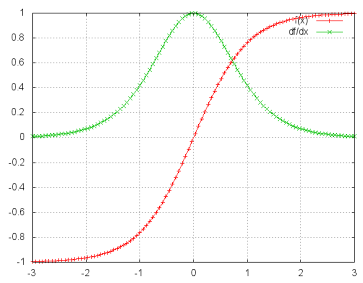


FIGURE 2.15: *Derivative of tanh*

Vanishing gradients also happen in deep Feedforward Neural Networks. It's just that RNNs tend to be very deep (as deep as the timesteps), which makes the problem a lot more common.

One way to reduce the effect of vanishing gradients is proper initialization of the Weight matrix. A more preferred solution is to use **ReLU**, since the ReLU derivative is a constant of either 0 or 1. But the most popular way is to use *Long Short-Term Memory (LSTM)* or *Gated Recurrent Unit (GRU)*.

Bidirectional RNNs

Bidirectional RNNs are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, if we predict a missing word in a sentence like "How ( ) you been?", we consider both "How" and "you been?". Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs(such as the sum of outputs of both directions at a particular timestep) Fig.2.16.

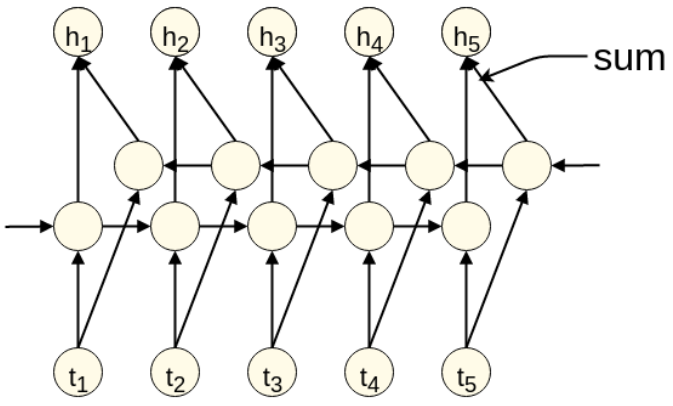


FIGURE 2.16: *Bidirectional RNN*

Deep RNNs
We can also stack multi-layers in RNN architecture, which gives us higher learning capacity, but we also need a lot of training data and the model is easy to overfit. Fig 2.17.

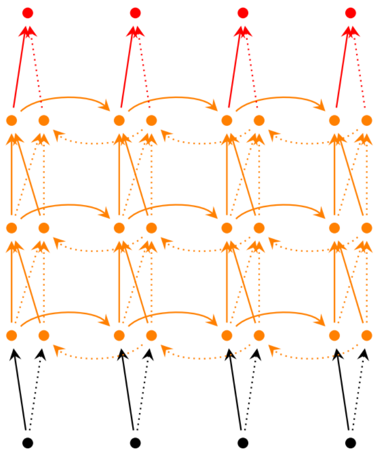


FIGURE 2.17: *Deep Bidirectional RNN*

2.2.1 Long Short-Term Memory

LSTMs are explicitly designed to avoid the long-term dependency problem. LSTM model introduces a new structure called a *memory cell*. A memory cell is composed of four main elements that serve to modulate the interactions between the memory cell itself and its environment:

1. **input gate**: One allows incoming signal to alter the state of the memory cell or block it, whose equation is Eq.2.14

$$i_t = sigmoid(W_{xi} \cdot x_t + W_{hi} \cdot h_{t-1} + b_i) \tag{2.14}$$

2. **forget gate**: One allows the state of the memory cell to have an effect on other neurons or prevent it, whose equation is Eq.2.15

$$f_t = sigmoid(W_{xf} \cdot x_t + W_{hf} \cdot h_{t-1} + b_f) \tag{2.15}$$

3. **output gate**: One modulates the memory cell's self-recurrent connection, allowing the cell to remember or forget its previous state, as needed, whose equation is Eq.2.16

$$o_t = sigmoid(W_{xo} \cdot x_t + W_{ho} \cdot h_{t-1} + b_o) \tag{2.16}$$

4. **self-recurrent**: One has a weight of 1.0 and ensures that, barring any outside interference, the state of a memory cell can remain constant from one timestep to another. The current cell is calculated by using all the gates and previous cell together as Eq.2.17. $\odot$ is elementwise multiplication and the $i_t \odot tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c)$ is the candidate state value filtered by input gate i_t.

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c) \tag{2.17}$$

i, f, o are just vanilla neural networks with sigmoid function. They have the same equation but different parameters corresponding physical meanings. LSTM cell can also be thought as a combination of filters learned by several neural networks that all take current input x_t and previous hidden stateh_{t-1} as inputs. Finally, the current hidden state is calculated as Eq.2.18, where the output activation function $tanh$ can be changed to ReLU or else. Fig.2.18 Illustrates how a LSTM cell combats vanishing gradients through a gating mechanism.

$$h_t = o_t \odot tanh(c_{t-1}) \tag{2.18}$$

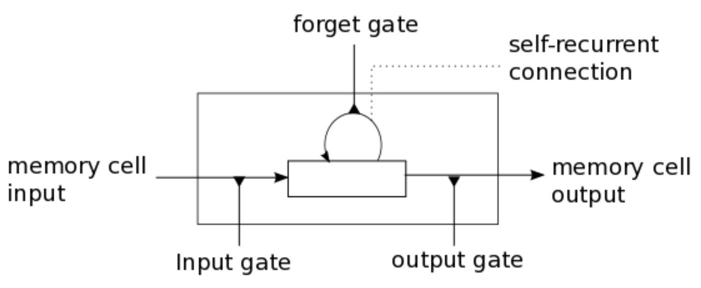


FIGURE 2.18: *Illustration of an LSTM memory cell*

Several excellent articles on LSTMs [27][13].

2.2.2 Gated Recurrent Unit

Another popular gating mechanism based method is Gated Recurrent Unit, introduced by[5]. The idea behind a GRU layer is quite similar to that of a LSTM layer. The structure is illustrated in Fig.2.19. It only has two gates,

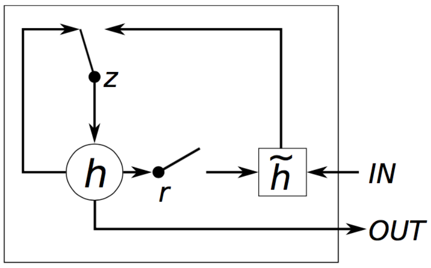


FIGURE 2.19: *GRU Gating*

the reset gate and an update gate that combines the forget and input gates. It also merges the cell state and hidden state. The resulting model is lighter than standard LSTM models.

1. **reset gate**: One determines how to combine the new input with the previous memory, whose equation is Eq.2.19

$$r_t = sigmoid(W_{xr} \cdot x_t + W_{hr} \cdot h_{t-1} + b_r) \tag{2.19}$$

2. **forget gate**: One defines how much of the previous memory to keep around, whose equation is Eq.2.20

$$u_t = sigmoid(W_{xu} \cdot x_t + W_{hu} \cdot h_{t-1} + b_u) \tag{2.20}$$

3. **self-recurrent**: The current cell is calculatedas Eq.2.21.

$$c_t = tanh(W_{xc} \cdot x_t + r_t \odot (W_{hc} \cdot h_{t-1}) + b_c) \tag{2.21}$$

4. **hidden state**: The current hidden state is calculatedas Eq.2.22.

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot c_t \tag{2.22}$$

Which to use
According to empirical evaluations in [6] and [45], in many tasks both architectures perform similarly. But GRUs have fewer parameters and may train a bit faster or need less data to generalize. On the other hand, if you have enough data, the greater expressive power of LSTMs may lead to better results.

2.2.3 Training

Because the RNNs also look at the previous information, therefore the ways we process the data for training and loss function are a little different.

Data preprocessing
We need to introduce anther dimension *timestep* for RNNs training. The input of each example is on longer a vector $\vec{x}$, but a sequence of vectors, which makes it matrix whose first dimension is timestep. If we use Mini-batch gradient descent to train the model, then the input data will become

a 3D tensor with a shape of **[batch size, timestep, vector dimension]** as illustrated in Fig.2.20.

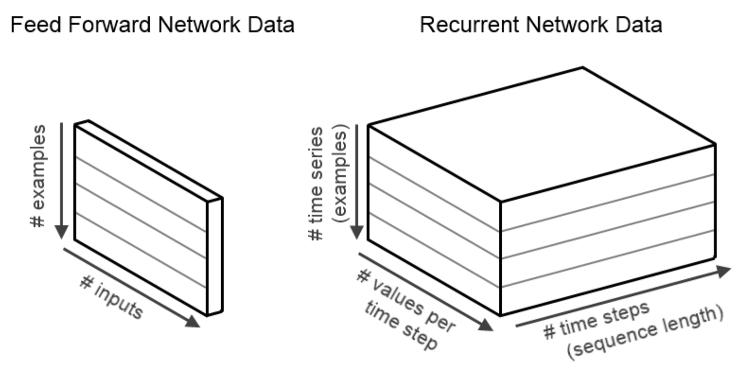


FIGURE 2.20: *RNN training inputs*

Loss function
Depending on the type of mapping shown in Fig.2.13, the loss function also varies. If it's many-to-one mapping, the network will just minimize loss for one step. If it's many-to-one mapping such as language model, then the finial loss can be the average over all timesteps as Fig.2.21.

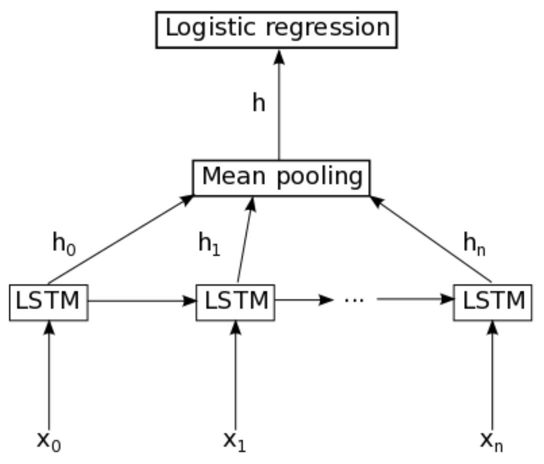


FIGURE 2.21: *The average loss over all timesteps is the finial loss*

2.2.4 RNN Architectures

As mentioned in Fig.2.13, Feed Forward Neural Network can do the one-to-one mapping and Recurrent Neural Network extend it to many-to-many or many-to-one mapping, both of which can be used in Distillation and Articulatory Inversion tasks and lead to different RNN architectures. I listed 3 architectures I used in my experiments below.

Many-To-Many
Many-To-Many mapping is commonly used in language model task and can also be used in here. The two sequences to be mapped in my experiments have the same length and the architecture is shown in Fig.2.22. The final loss is the average over all timesteps.

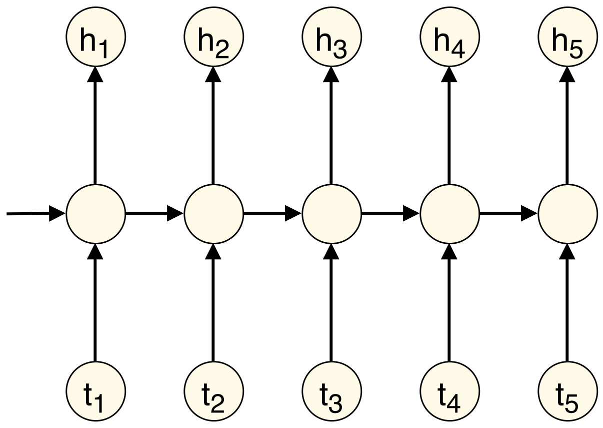


FIGURE 2.22: *Many-To-Many*

Many-To-One
Depending on which state the model predicts, the architecture also changes a bit.

Only look at history
The idea of this one is to use historical information to predict the last state. The architecture is shown in Fig.2.23 and the final loss is just the loss of last state.

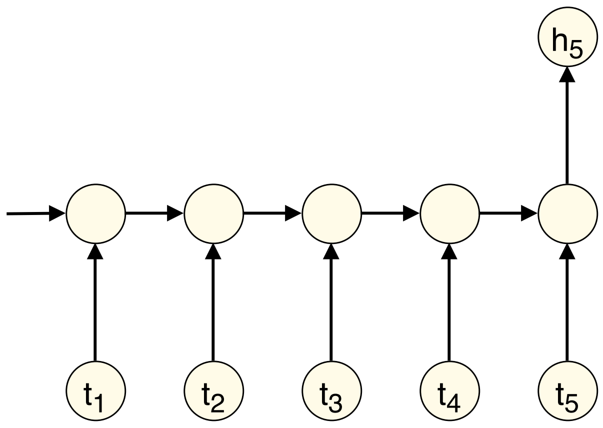


FIGURE 2.23: *A certain mount of historical states to current state*

Chapter 3

Fundamentals of ASR

A typical structure of Automatic Speech Recognition (ASR) system is illustrated in Fig.3.1.

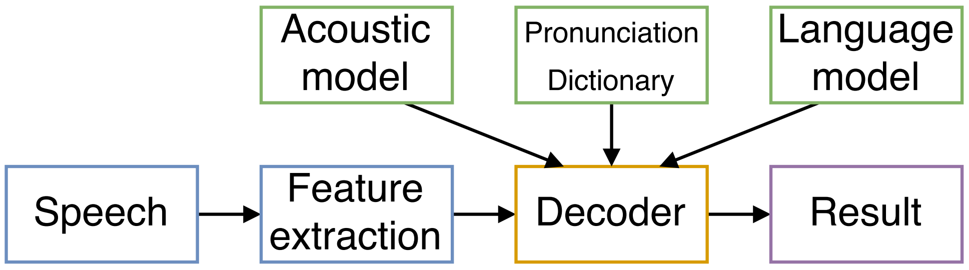


FIGURE 3.1: *Block diagram of a typical Automatic Speech Recognition System*

Speech signal first will be converted into a sequence of fixed size acoustic vectors $o = o_1, o_2, ...o_n$ and the decoder tries to find the most probable sequence of words $w = w_1, w_2, ...w_m$ given the sequence of acoustic vectors. Eq.3.1.

$$\hat{w} = \arg\max_w P(w|o) \tag{3.1}$$

However, $\arg\max_w P(w|o)$ is difficult to find directly. Here, Bayes' Rule is applied to transform Eq.3.1 into the equivalent Eq.3.2.

$$\hat{w} = \arg\max_w P(o|w)P(w) \tag{3.2}$$

Then, the likelihood $P(o|w)$ is found by *an acoustic model* and the prior probability $P(w)$ is found by *a language model*.

The basic unit of speech represented in a standard acoustic model is called *phoneme* (For example, the word "bad" is composed of three phones /b/ /ae/ /d/). For any word w, its acoustic model is the combination of several phoneme models, which is defined in *a pronunciation dictionary*.

3.1 Feature Extraction

Like any other pattern recognition, before we train a model, the speech signal will be converted into feature vectors recognizable to the system.

3.1.1 Signal Preprocessing

Before feature extraction, several steps are needed to pre-process the signal as illustrated in Fig.3.2.

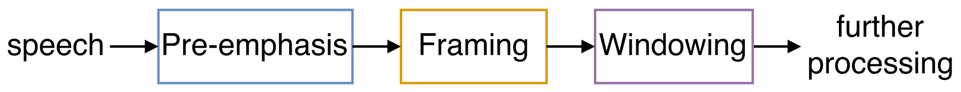


FIGURE 3.2: *Pre-processing steps*

1. **Pre-emphasis**: is a high-pass filter, that boosts the high-frequency energy.
2. **Framing**: divides speech signal into small overlapping chunks. Length is typically $20 - 25ms$ and Shift is typically $10ms$ or about 50% of the frame.
3. **Windowing**: Each frame is multiplied by a window function to minimize spectral discontinuities at the frame start/end. The most common used window is *Hamming window*.

3.1.2 Mel Frequency Cepstral Coefficient

One of the most popular and reliable features for speech recognition is *Mel Frequency Cepstral Coefficents (MFCCs)*[8]. The extraction process is shown in Fig.3.3.

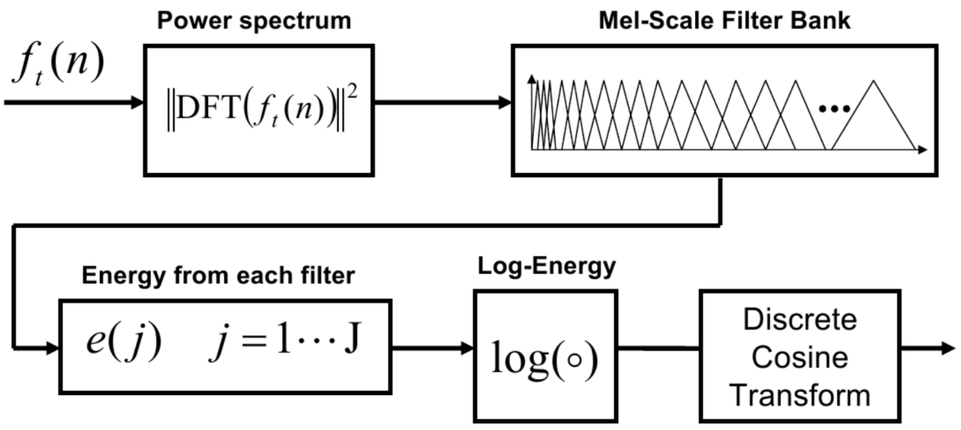


FIGURE 3.3: *MFCC block diagram*

1. **Power Spectrum**: Apply *Discrete Fourier Transform* to windowed signal $f_t(n)$ to transform the signal from time-domain to frequency-domain and take of power of it, because in frequency-domain, the signal can be analysed more consistently and easily.
2. **Mel-Scale Filter Bank**: This approximates the human auditory system's response
3. **Discrete Cosine Transform (DCT)**: After log-energy, DCT is applied to map the signal back from the frequency domain into the time domain. Instead of Inverse Discrete Fourier Transform (IDFT) use DCT because log power spectrum is real and symmetric. Usually people take the first 12 Coefficients.

4. **Frame Energy**: If we append energy computed from windowed frame, the dimensions become 13.

5. **Velocity and Acceleration**: To introduce the information of temporal changes, velocity and acceleration are often appended, resulting a 39 dimensional vector.

3.2 Language Modelling

The prior probability $P(w)$ of a sequence of words $w = w_1, w_2, ...w_K$ is given by Eq.3.3, where N is typically in the range $2 - 4$. This is called *N-gram Language Model*. It gives the probability of a word based on previous $N - 1$ words.

$$P(w) = \prod_{k=1}^{K} P(w_k | w_{k-1}, w_{k-2}, ..., w_{k-N+1}) \tag{3.3}$$

The N-gram probabilities are estimated from training texts by counting N-gram occurrences to form maximum likelihood (ML) parameter estimates. ML estimates uni-gram, bi-gram, tri-gram respectively: Eq.3.4,Eq.3.5,Eq.3.6, where $C()$ the occurrences of word or word sequence in the training texts.

$$P(w_n) \approx \frac{C(w_k)}{C(ALL)} \tag{3.4}$$

$$P(w_n | w_{n-1}) \approx \frac{C(w_{k-1}, w_k)}{C(w_k)} \tag{3.5}$$

$$P(w_n | w_{n-2}, w_{n-1}) \approx \frac{C(w_{k-2}, w_{k-1}, w_k)}{C(w_{k-1}, w_k)} \tag{3.6}$$

However, in practice There never be enough data to estimate all possible probabilities and the probability of unseen word estimated in this way will become 0. to avoid zero probability events, probability estimates are smoothed. Several smoothing methods are listed below:

1. **Back-off**: If an N-gram doesn't exist, back-off to smaller order[14].

2. **Laplace smoothing**: Assume that N-grams occur one more time than they actually do.

3. **Model interpolation**: Interpolate with lower order N-grams.

4. **Good-Turing discounting**: Redistribute probability mass from "seen" events to "unseen" events by discounting counts.

5. **Class N-gram models**: Cluster the similar words together.

3.3 Acoustic Modelling

Modern speech recognition systems are mainly based on Hidden Markov Models for finding the likelihood $P(o|w)$.

3.3.1 Hidden Markov model

The speech that a ASR system detects is generated from the internal physical changes in the human body. The internal physical changes are hidden, but the resulting sounds are observable. Hidden Markov Model (HMM)[9] is used to estimate the sequence of internal physical changes by knowing the sequence of resulting sounds.

An HMM model in ASR task is characterized by the following 5 components (2 sets of states and 3 sets of probabilities):

1. **Hidden States**: The internal hidden physical changes in human body.
2. **Observable States**: The resulting sounds generated by the internal physical changes.
3. **Initial State Probabilities**: A vector π of the initial state probabilities at time $t = 1$.
4. **Transition Probabilities**: matrix A storing every hidden state probability given the previous hidden state.
5. **Emission Probabilities**: matrix B storing every probability of observing a particular observable state given that the hidden model is in a particular hidden state.

The hidden state of the model depends only upon the previous n hidden states of the model. and each probability in A and B are time independent.

A left-to-right HMM for acoustic modelling looks like Fig.3.4.

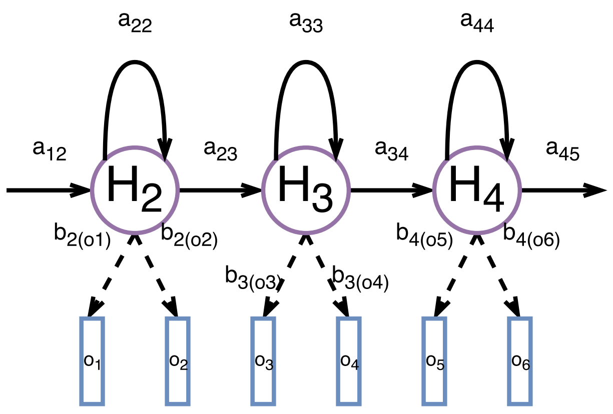


FIGURE 3.4: *HMM*

Using left-to-right topology is because speech is time-evolving, non-stationary signal. Paths from current state to previous state are not existing in ASR task neither. The joint probability that o is generated by an HMM model M moving through the hidden state sequence w is calculated as the product of transition probabilities and emission probabilities, which is Eq.3.7.

$$P(o, w|M) = a_{12}b_{2(o_1)}a_{22}b_{2(o_1)}a_{23}b_{3(o_3)}\cdots \tag{3.7}$$

Given w is hidden, the likelihood of $P(o|M)$ is computed by summing all possible state sequences $w = w(1), w(2), ..., w(T)$, that is Eq.3.8 and can be approximated by only considering the most likely state sequence, that is Eq.3.9.

$$P(o|M) = \sum_{w} a_{w(0)} a_{w(1)} \prod_{t=1}^{T} b_{w(t)(o_t)} a_{w(t)} a_{w(t+1)} \tag{3.8}$$

$$P(o|M) = \max_{w} \left\{ a_{w(0)} a_{w(1)} \prod_{t=1}^{T} b_{w(t)(o_t)} a_{w(t)} a_{w(t+1)}) \right\} \tag{3.9}$$

M_i is a Hidden Markov Model of word i. Since, $P(o|w_i) = P(o|M_i)$, the $P(o|w_i)$ can be found by computing $P(o|M_i)$.

The HMM is trained by using *Baum–Welch algorithm*[42].

3.3.2 GMM-HMM-based Modelling

In GMM-HMM-based Modelling, the $b_{w(t)(o_t)}$ is found by a Gaussian Mixture Model (GMM)[4] that computes probability density function (pdf) over a continuous space of input feature vectors.

3.3.3 DNN-HMM-based Modelling

In DNN-HMM-based Modelling, the $b_{w(t)(o_t)}$ is replaced with $P(o|h)$ found by a Deep Neural Network calculated using Eq.3.10, where h_t is a particular hidden state of a Hidden Markov Model of a word and o_t is observable vector.

$$P(o_t|h_t) = \frac{P(h_t|o_t)P(o_t)}{P(h_t)} \tag{3.10}$$

The prior probability of hidden state $P(h_t)$ is calculated from (occurrences of) the training set, and $p(o_t)$ can be assigned a constant since the observation feature vectors are regarded as independent of each other.

The DNN-HMM Training Procedure is listed below:

1. Train a standard GMM-HMM based recognizer.
2. Use force alignment to get the corresponding target label h_t of hidden state in the recognizer for every vector o_t as the DNN training data.
3. Count occurrences of hidden states in the training set to compute the prior probability of every hidden state $P(h_t)$.
4. Train a DNN that maps spaces o_t –> h_t (classification task of o_t).
5. With $P(h_t|o_t$ got from the trained DNN model and $P(h_t$ compute $P(o_t|h_t)$ using Eq.3.10.
6. Use Eq.3.9 and Eq.3.2 to get the recognition results.

Phoneme Modelling

If using phoneme modelling, then every HMM represents a phoneme instead of a word.

3.4 Decoding

In decoding, the acoustic score and language model score are combined together to determine the most likely word sequence given observation sequence. However, there is often a significant mismatch between the dynamic range of the two scores. The dynamic range of the acoustic likelihood can be excessively high, which makes the effect of language model relatively small. Therefore, the language scores are often scaled, then the fundamental formula of speech recognition can be modified as:Eq.3.11 and the solution is given by the *Viterbi algorithm*[10].

$$\hat{w} = \arg\max_{w} P(o|w)P(w)^{s} \tag{3.11}$$

where s is language model scale (lm scale) factor.

3.5 System Evaluation

The metrics for speech recognizers are word/phone error rate (WER/PER) and the accuracy, that tells how the results recognized by the system differs from the orthographic transcripts. The word/phone error rate and accuracy are defined as follows: Eq.3.12 and Eq.3.13

$$WER/PER = \frac{S+D+I}{N} \tag{3.12}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions and N is the total number of words in the reference.

$$ACC = 1 - WER/PER = \frac{N-(S+D+I)}{N} = \frac{H-I}{N} \tag{3.13}$$

where H is $N-(S+D)$, the number of correctly recognized words.

Chapter 4

System Description

In this chapter, I will describe the four systems I proposed and a new RNN architecture.

1. Distillation based on feedforward neural networks.
2. Distillation based on recurrent neural networks.
3. Articulatory inversion based on feedforward neural networks.
4. Articulatory inversion based on recurrent neural networks.

4.1 Machines-Teaching Machines

In many tasks, lots of useful information is not available during test (such as articulatory information). The most straightforward way to utilize such information is to learn a mapping(such as articulatory inversion) with accessible information and generate it during test. But inversion is difficult task, which also costs more computational resources and time.

Privileged Information
Privileged Information is a technique that utilize such information based on the idea that additional information can serve as an intelligent teacher which can guide the training process of existing models. Such approach allows building a classifier which is better than those built on the regular features alone. This framework is also known as *learning using privileged information*. The paradigm of *machines-teaching machines* has been investigated in studies of Vapnik [40] [39] and Hinton [12].

Distillation
Hinton proposed the concept of distilling the knowledge in neural networks [12], where the soft target is integrated into the training process of DNNs to transfer the knowledge from the cumbersome model to a small model that is more suitable for deployment. They use a weighted average of two different objective functions to train DNN. The first objective function is the cross entropy with the soft targets and this cross entropy is computed using the same high temperature in the softmax of the distilled model as was used for generating the soft targets from the cumbersome model. The second objective function is the cross entropy with the correct labels. This is computed using exactly the same logits in softmax of the distilled model but at a temperature of 1.

Soft Target Standard classification based on DNNs use the one-hot vector as the hard target to calculate loss minimized. However, the entropy of one-hot vector is low and can not tell us the relations between classes. For example, if we classify images, pictures of the owl might look a bit like the hawk but is quite different from pictures of the dog or the car. With the soft target, we can know the relations between classes, in other words, the distances between classes in the representation space. The idea is similar to *word embedding* in language model tasks.

For example, in Fig.4.1

$W(\text{"}woman\text{"}) - W(\text{"}man\text{"}) \approx W(\text{"}aunt\text{"}) - W(\text{"}uncle\text{"})$
$W(\text{"}woman\text{"}) - W(\text{"}man\text{"}) \approx W(\text{"}queen\text{"}) - W(\text{"}king\text{"})$

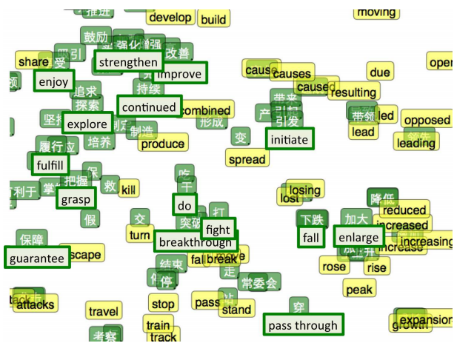


FIGURE 4.1: *The distances between words*

The soft targets have the length as the one-hot vector but also have values at other indices. They are the output of "teacher model" that trained with more information even not available during real test time.

Temperature Soft target is valuable information that defines a rich similarity structure over the data but it has very little influence on the cross-entropy cost function during the transfer stage because the probabilities are so close to zero. One way to address this is to soften the soft target with a temperature as shown in Eq.4.1. It is just to divide the inputs by T before inputs are passed through the softmax function.

$$\phi(x)_j = \frac{e^{(x_j/T)}}{\sum_{k=1}^{K} e^{(x_k/T)}} \tag{4.1}$$

Generalized Distillation

Generalized distillation has been termed in [18] to frame two techniques of Hinton's distillation[12] and Vapnik's privileged information[39] that enable machines to learn from other machines.

In the framework, an "intelligent teacher"is incorporated into machine learning and the training data is formed by a collection of triplets

$$(x_1, x_1^*, y_1), \ldots, (x_n, x_n^*, y_n) \sim P^n(x, x^*, y),$$

where x_i, y_i is a feature-label pair and x_i^* is additional information about x_i, y_i provided by an intelligent teacher. The teacher is assumed to develop a language that effectively communicates information to help the student come up with better representation and to enable to learn characteristics about the decision boundary which are not contained in the student samples.

The process is as follows:

1. Learn teacher $f_t \in \mathcal{F}_t$ in eq. (4.2) using $\{(x_i^*, y_i)\}_{i=1}^n$.

$$f_t = \arg\min_{f \in \mathcal{F}_t} \frac{1}{n} \sum_{i=1}^{n} l(y_i, \sigma(f(x_i^*))) + \Omega(||f||) \tag{4.2}$$

   Here, $x_i^* \in \mathcal{R}^d$, $y_i \in \Delta^c$, Δ^c is the set of c-dimensional probability vectors, F_t is a class of functions from $\mathcal{R}^d$ to $\mathcal{R}^c$, $\sigma : \mathcal{R}^c \to \Delta^c$ is a soft-max function, l is a loss function and Ω is an increasing function which serves as a regularizer.

2. Compute teacher soft labels $\{\sigma(f_t(x_i^*)/T\}_{i=1}^n$ using temperature parameter $T > 0$.

3. Learn student $f_s \in \mathcal{F}_s$ in eq. (4.3) using $\{(x_i, y_i)\}_{i=1}^n$, $\{(x_i, s_i)\}_{i=1}^n$ and imitation parameter $\lambda \in [0, 1]$.

$$\begin{aligned} f_s &= \arg\min_{f \in \mathcal{F}_s} \frac{1}{n} \sum_{i=1}^{n} [(1-\lambda) l(y_i, \sigma(f(x_i))) \\ &+ \quad T^2 \lambda l(s_i, \sigma(f(x_i)))] \end{aligned} \tag{4.3}$$

$$s_i = \sigma(f_t(x_i)/T) \in \Delta^c \tag{4.4}$$

   Here, $\mathcal{F}_s$ is a function class simpler than $\mathcal{F}_t$.

Take the example of articulatory task, the training process is illustrated as Fig.4.2 Outputs of the teacher DNN are used as soft targets s_i and together with the hard targets y_i act as arguments of the student DNN loss function as in Eq.(4.3). In addition, teacher DNN outputs are smoothed with the temperature parameter T according to Eq.(4.4). The input training data for the student DNN are acoustic features only, and are fed in batches. The corresponding concatenated acoustic and articulatory data, also in batches, are given to the teacher DNN input. However, only student DNN parameters are updated during this procedure.

During the test, only student DNN is used and the state probability predictions from the "hard" output, i.e. the output that was compared with the hard targets during training, are fed to the HMM decoder as shown in Fig.4.3.

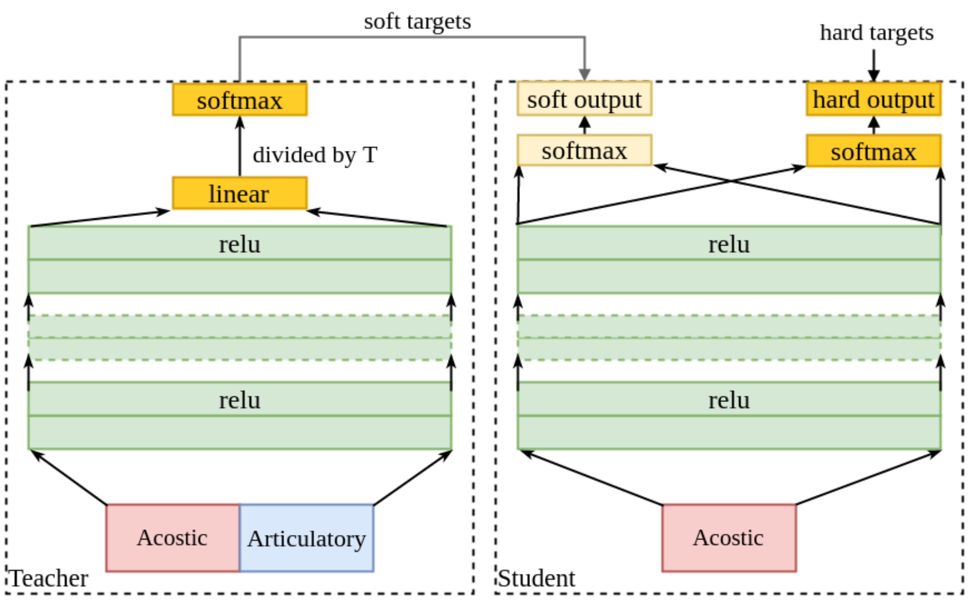


FIGURE 4.2: *The distillation training block diagram*

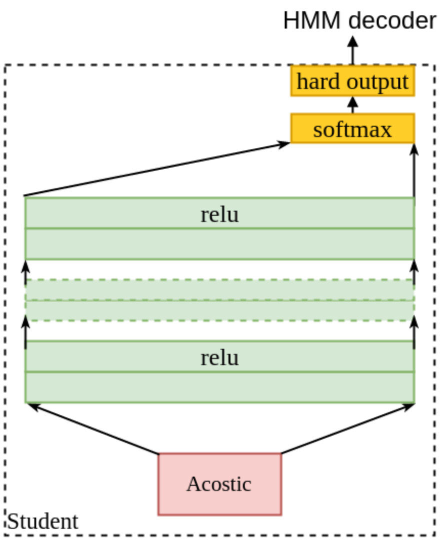


FIGURE 4.3: *Testing with student DNN*

4.2 Articulatory Inversion

Many methods can do the articulatory inversion, yet I'm focusing on the methods based on DNN. The block diagram using articulatory inversion is illustrated in Fig.4.4.

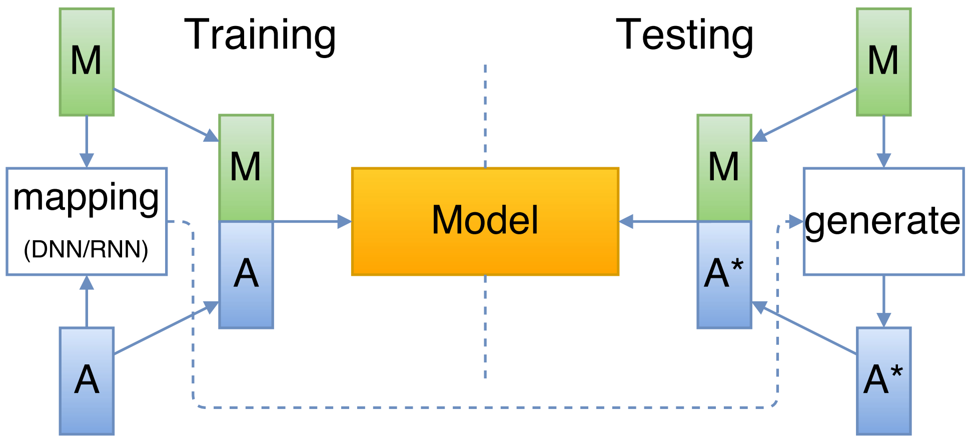


FIGURE 4.4: *articulatory inversion block diagram*

During Training

1. train model with both acoustic and articulatory data
2. map acoustic and articulatory features using DNN/RNN

During Testing

1. generate the articulatory feature given acoustic feature
2. recognize in standard way

4.3 New RNN Architecture

In chapter 3, many-to-one mapping as shown in Fig.2.23 only looks at the history information. whereas, I found that with a new RNN architecture described below, the model can even give better results than the standard architecture.

look at history and future
The idea of this one is to use historical information as well as the future information to predict one state. The architecture is shown in Fig.4.5 and the final loss is just the loss of state in the middle. Compared with the RNN architecture that only looks at historical information, this one is better at modelling coarticulations, because phonemes coming after also have influences on the current phoneme.

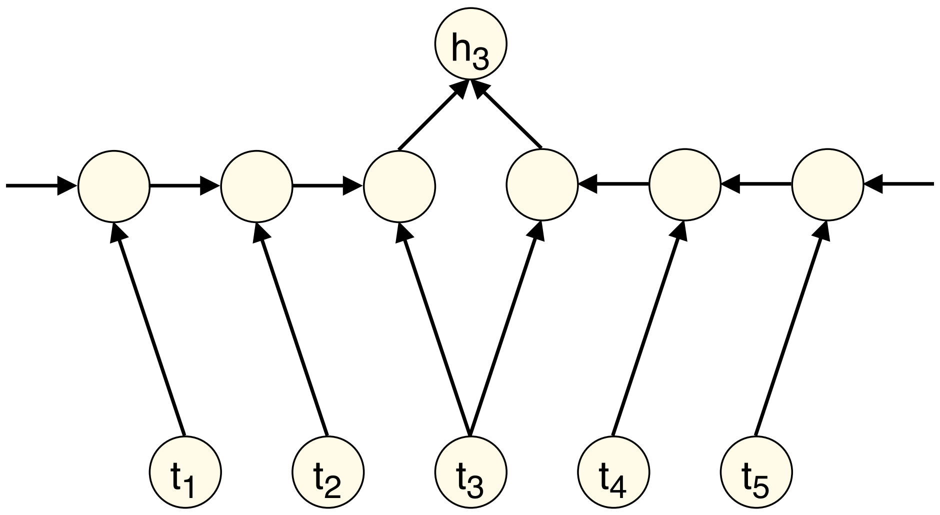


FIGURE 4.5: *A certain mount of historical states and future states to one state*

Chapter 5

Experiments

5.1 Database Description

I experimented with the University of Wisconsin X-ray microbeam database (XRMB) [43] which consists of simultaneously recorded acoustic and articulatory measurements from 47 American English speakers (22 males, 25 females). Each speaker's recordings comprise at most 118 tasks whose type can be number sequence, TIMIT sentences, isolated word sequence, paragraph as well as non-speech oral motor. The order and content of records was the same for all speakers.

Task list
Task list is shown in Table.5.1. Only normal speed sentences and number sequence tasks were used in the experiments for now.

1. Word type: seven words selected randomly from the full list of citation words.
2. Sentence type: three different examples selected from the sentence list.
3. Long passages of connected speech: were partitioned(the last and first sentences of each contiguous subdivision were repeated).

Data Forms
There are three forms of data in XRMB database:

1. Orthographic transcripts of the spoken utterance.
2. Digitized waveforms of the recorded speech(sample rate: 21.74 kHz).
3. Simultaneous 2D articulators' trajectories ($[n, 16]$), where row is time, sampled every 6.866 milliseconds and column is x and y-coordinate history for 8 pellets on the tongue, lips, and jaw as shown in Fig.5.1.

Pronouncing Dictionary
In my experiments, the pronouncing Dictionary is the Carnegie Mellon University (CMU) Pronouncing Dictionary without stress[37]. The CMU dictionary is an open-source machine-readable pronunciation dictionary for North American English that contains over 134,000 words and their pronunciations. The current phoneme set has 39 phonemes, not counting varia due to lexical stress. Along with silence, there are 40 phonemes in my systems, which are listed in A.

TABLE 5.1: *Task List.*

Record type	**Number**
word: standard	40
word-like: vcv, cvc, vseq, v	4
sentence: normal	36
sentence: fast and slow	6
sentence: clear	1
sentence: emphasis	2
paragraph	6
swallow	10
diadochokinetic	3
counting (1-20)	1
number sequences	5
oral gym: wag and protrude	4

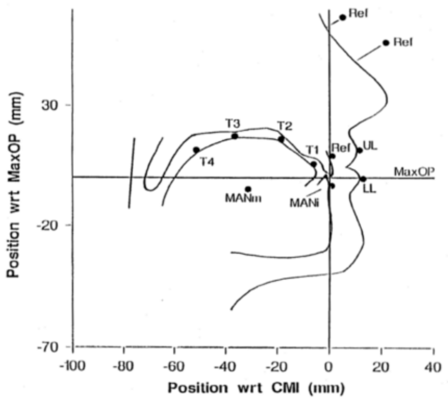


FIGURE 5.1: *Placement of the 8 pellets on T1,T2,T3,T4,MANm, MANi,UL,LL points.*

5.2 Data Processing

Acoustic Data Processing
I downsampled the acoustic signal from 21.74 kHz to 16 kHz, and my acoustic features are 13-dimensional mel-frequency cepstral coefficients (MFCCs) computed every 10ms over a 25ms window, along with their first and second derivatives, resulting in 39 dimensional frames.

Articulatory Data Processing
I also down-sampled articulatory data from the original rate of $145.7Hz$ to $100Hz$ to match the frame rate ($10ms$) of acoustic features and use the x,y coordinates of the 8 articulators along with their first and second derivatives as articulatory feature vectors of 48 dimensions. Including the first and second derivatives of the articulatory data is helpful since the movement itself can't tell apart speech pause from other phones.

Data Cleaning
Due to limitations in the recording technologies, articulatory measurements

contain missing data when individual pellets are mis-tracked. Though there are methods to reconstruct missing data [41], I decided to use only complete data. The missing entries as well as data that are not consistent with orthographic transcripts were cut off. Utterances are also split into files, each containing only one sentence with silence parts of at the beginning and end reduced to 150ms.

To do this, first, I need to know the exact time period of every phoneme. This was done using the Penn Phonetics Lab Forced Aligner[44]. Then each utterance was then processed following the several steps listed below:

1. Cut off the periods whose articulatory data is missing.
2. From the remaining part, find out the silent parts whose period are longer than $150ms$.
3. Take the first and last silent parts as well as the words between them.
4. If such silent parts are less than two, discard the utterance.
5. All acoustic and articulatory vectors were then mean and variance normalized on per utterance basis.

After these steps, I split the remaining data into several sub-sets as shown in Table.5.2.

TABLE 5.2: *Details of the datasets.*

	Train	**Test**	**Validation**	**Total**
Speakers	36	5	4	45
Female	19	3	2	24
Male	17	2	2	21
Utterances	3040	379	256	3675
Words	33883	4167	2758	40808
Phonemes	144863	17815	11580	174258
Hours	2:33:14	19:46	12:16	3:05:16

5.3 GMM-HMM baseline

The global information about systems used in the experiments is listed below:

1. HMM model: standard 3-state left-to-right monophone model
2. Phoneme language mode: a simple bi-gram trained on data transcriptions including the paragraph task
3. HMM states: 120 states

Speaker Dependent
To see how recognizers using the data in XRMB preforms, four conventional GMM-HMM recognizers were built using HTK Toolkits:

1. only use acoustic features (39D)

2. use MFCCs and features of T1,T2,T3,T4 articulators (39+24D)
3. use MFCCs and features of T1, T2, T3, T4, UL, LL articulators (39+36D)
4. use MFCCs and features of all articulators (39+48D)

The phoneme error rate (PER) of speaker JW45 (102 utterances) when using different number of Gaussian components are collected in Fig.5.2. The re-

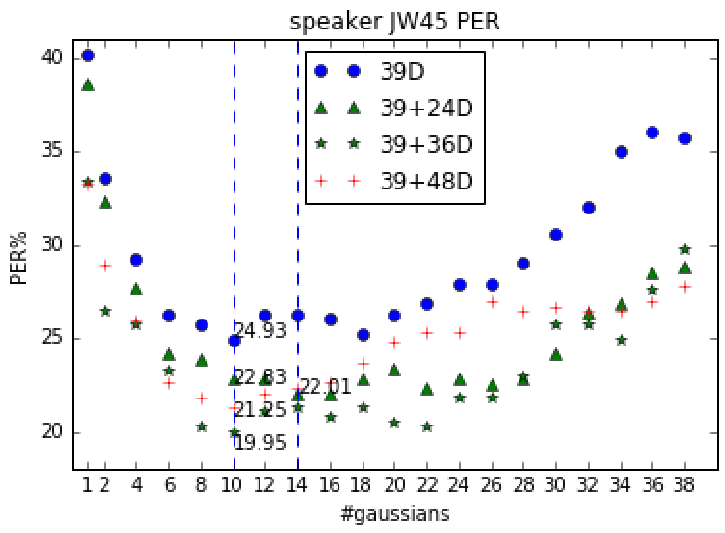


FIGURE 5.2: *Results of speaker JW45*

sults verified as well that with articulatory information, the ASR can achieve much better preference. However, the recognizer with all articulators is not as good as the one with T1, T2, T3, T4, UL, LL articulators. This is probability due to the lack of data and also indicate the pattern of articulators MANm, MANi is not as clear as others.

Speaker independent
The recognizers above also were also trained in speaker independent (Table.5.2) task and the results are collected in Fig.5.3. This time, with more data, the articulatory information gives big improvement. The systems with more than 38 Gaussian components don't give clear improvement, therefore the system with 38 Gaussian components was chosen to be the baseline.

After optimized the language model's weight/penalty, the final results decoded using Julius are summarized in Table.5.3. Frame level DNN training targets were generated from this GMM-HMM system.

TABLE 5.3: *Phone error rates for conventional GMM-HMM system.*

LM weight/penalty	**MFCC**	**MFCC+ART**
0/0	29.95	12.01
7.0/2.0	18.85	9.67
7.0/1.0	18.73	9.72

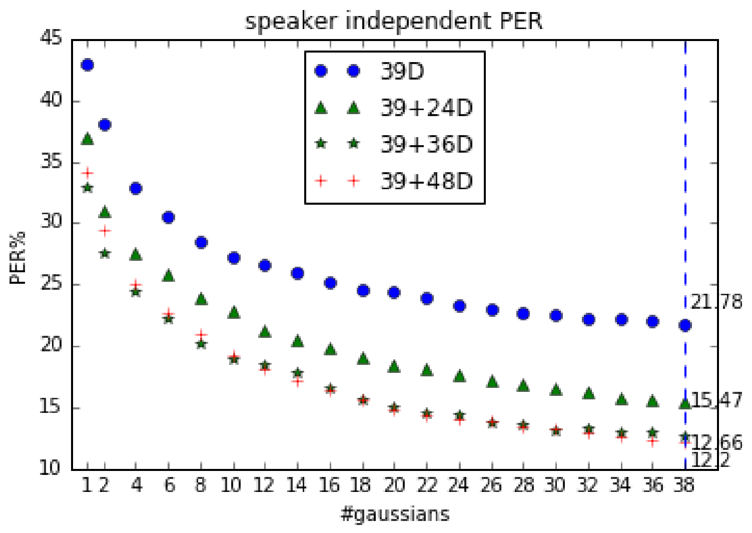


FIGURE 5.3: *Results in speaker independent task*

5.4 Generalized Distillation

5.4.1 Feed-Forward DNN

The next step is to replace GMM with Feed-Forward DNN. Following some other studies [16][30] and [1], I set the input window of 17 feature vectors, resulting in 663 or 1479 input nodes when using MFCC or MFCC+ART features. global DNN settings used in the experiments are listed in Table.5.4.

TABLE 5.4: *Global DNN settings*

Regularization	dropout and early stopping
Loss function	Categorical Cross-entropy
Hidden activation functions	ReLU's
Output activation function	softmax
Hidden nodes	all layers have same number of hidden nodes
Update	adam

I also compared several optimization methods such as SGD+Nesterov Momentum(0.9) [34], rmsprop[35] as well as Adam [15], but found no significant differences in results. Adam and Rmsprop are faster, but the initial learning rate has to be smaller than SGD's. For the following experiments, Adam optimization was used and the learning rate starts from 1e-4, and is multiplied with 0.1 if validation data loss doesn't go down for 3 epochs. The entire training procedure is stopped when the learning rate is smaller than 1e-6 or the maximum of 100 iterations is reached.

To select optimal DNN structure, four parameters were varied, the number of hidden layers [3,**5**,6], the number of hidden nodes [1024,2048,3072], batch size [128,**256**,512] as well as drop out probability [0.1,0.2,0.3,**0.4**]. In total 144 DNN models were trained and found that more layers and nodes increase the model's learning power, whereas smaller batch size and higher dropout probability tend to prevent the model from over-fitting. The best results are from the models that are well-balanced between this two trends.

Teacher Model
First, the MFC+ART Models (87D inputs) were tested in frame level state classification mode. The number of hidden layers did not have big effect on the accuracy which was around 82%. However, that the dropout influences the performance. Several training configurations and the corresponding frame level state classification accuracies as well as the phoneme error rates for a DNN trained with 256 batch size are shown in Table.5.5. The best PER result of **4.56**% was achieved with 5 hidden layers, 3072 nodes and 40% dropout. Compared with the GMM-HMM system, this performance is 2 to 3 times better. Thus, we chose this DNN as our teacher model.

TABLE 5.5: *Training conditions and performance of the MFC+ART model with 256 batch size*

Layers	Nodes	Drop	PER%	Acc%
3	2048	0.3	5.13	82.2
3	3072	0.3	5.19	82.3
3	2048	0.4	4.96	82.4
3	3072	0.4	4.98	82.6
4	2048	0.3	4.86	82.3
4	3072	0.3	5.01	82.4
4	2048	0.4	4.69	82.0
4	3072	0.4	4.71	82.5
5	2048	0.3	4.82	82.2
5	3072	0.3	5.03	82.1
5	2048	0.4	4.67	81.8
5	**3072**	**0.4**	**4.56**	**82.4**
6	2048	0.3	4.78	81.9
6	3072	0.3	4.61	82.3
6	2048	0.4	4.75	81.8
6	3072	0.4	4.90	81.5

Student Model
The same strategy was applied when using only acoustic feature. Based to the results of MFC+ART model, the batch size was set to 256. Table 5.6 shows the MFC Model training conditions and performance for different number of hidden layers and nodes.

Distillation Training
Based on the results from MFC Model, for the distillation training experiments, the student model has 4 hidden layers, 2048 hidden layer nodes, 40% dropout. The temperature parameter in Eq.(4.4) was varied from 1 to 5 and the imitation value was changed from 0 to 1 in steps of 0.2. Learning of the distilled student was performed as illustrated in Fig.4.2. The results in terms of PER are summarized in Fig.5.4. The blue dashed line shows the result of the student when trained alone, so it doesn't depend on the

TABLE 5.6: *Training performance of the MFC Model.*

Layers	**Nodes**	**Drop**	**PER%**	**Acc%**
3	2048	0.3	9.18	79.5
3	3072	0.3	9.11	79.4
3	2048	0.4	8.50	80.0
3	3072	0.4	8.68	79.9
4	2048	0.3	9.02	79.5
4	3072	0.3	8.98	79.5
4	**2048**	**0.4**	**8.18**	**79.7**
4	3072	0.4	8.46	79.7
5	2048	0.3	8.43	79.7
5	3072	0.3	8.92	79.2
5	2048	0.4	8.23	79.6
5	3072	0.4	8.34	79.4

temperature or imitation parameters. The teacher result serves as the lower bound distilled student can achieve. As can be seen from the figure, for $T = 1$ and $\lambda = 0.6$, distillation result is 6.74% PER, which is 17.6% better than the result of the student alone. For the temperature $T = 2$, the performance is still better than the student alone, but not as good as temperature $T = 1$. Temperatures of 10 and higher, however, performed worse which can be explained with the smoothing effect they have on the teacher output.

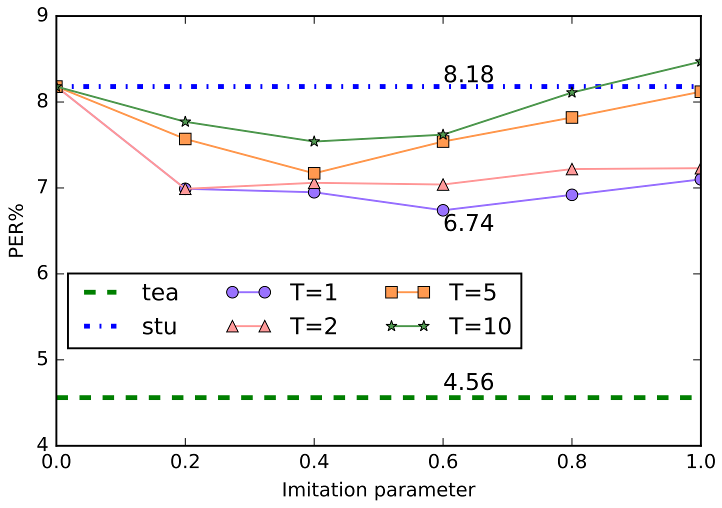


FIGURE 5.4: *Results of distillation training*

Using DNN in the acoustic model provides big performance boost compared to the conventional GMM-HMM systems and the experiments have confirmed this observation as well. The student DNN trained without distillation achieves 8.18% average PER, while the GMM-HMM system's result is 18.73%. When the distillation framework was applied, additional 17.6% performance improvement was achieved leading to PER as low as 6.74%.

5.4.2 Recurrent DNN

The FeedForward nets then were replaced by Recurrent nets to see how Recurrent DNNs perform using Distillation training.

Teacher Model
The Recurrent nets are BLSTM-DNN hybrid models and the global settings are shown in Table.5.7

TABLE 5.7: *Global RNN settings*

Timesteps	30
Batchsize	128
RNN cells	BLSTM
Cell nodes	2048
Hidden nodes	2048

The rest are the same as DNNs', But have an additional architecture as illustrated in 5.5. Several frames (timesteps) are used to predict a single HMM state.

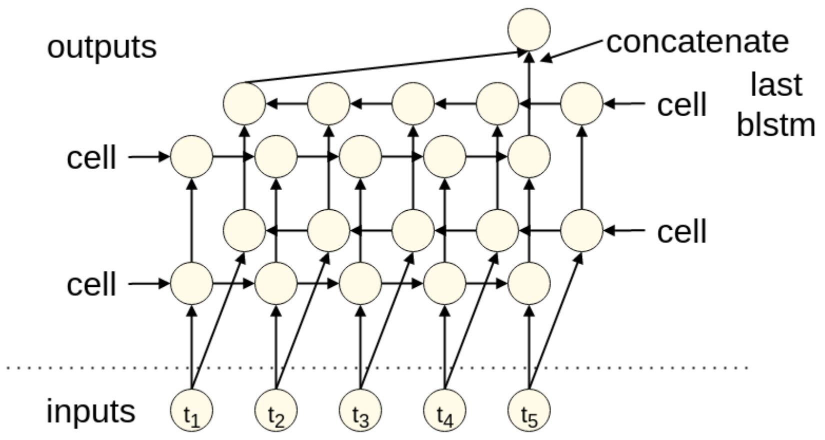


FIGURE 5.5: *Deep BLSTM architecture*

The length of timesteps for my task can vary due to the fact that there is no history information for the first n frames of every sentence. For example, the first frame of a sentence has only one timestep (ifself), which makes the RNN become DNN and the second frame of a sentence has a timesteps of 2.

Training RNNs is very time-consuming even with the help of GPU. To select optimal RNN structure, first two factors were varied: dropout probability [0, 0.1, 0.2,**0.3**, 0.4] and the architecture [IBO, IBBO, IFBO, IFBFO, IFBBO, **IFFBFFO**], where I, F, B ,O denote input, feedforward, BLSTM, output layers respectively. In total 30 models were trained. Some results are shown in Table 5.8. The best PER of **4.26%** was achieved with a structure of IFFBFFO, 2048 hidden nodes, 2048 cell nodes and 30% dropout.

Further more, the PER of **3.71%** can be achieved when change timestep to 41, batch size to 32. However with the same dimension of cell, BLSTM and LSTM preform similar for my tasks.

TABLE 5.8: *Training performance of different RNN architectures with 30% dropout.*

Archi	PER%	Acc%
IBO	5.26	77.4
IBBO	4.89	78.9
IFBO	4.9	79.3
IFBBO	4.39	80.7
IFBFO	4.75	79.3
IFFBFFO	**4.26**	**80.2**

Student Model
The PER of **4.61%** can be achieved with a student RNN model trained with the same architecture and parameter size as the teacher's. But the HMM states for training were got from GMM+HMM model of the teacher, because I found that targets from teacher model (MFC+ART) always give better results than targets from student model (MFC).

Distillation Training
When $T = 1$ and $\lambda = 0.6$, PER of distillation based on RNN is **4.44%**.

5.5 Articulatory Inversion

Speaker Dependent
This mapping corresponds to the GMM+HMM baseline with (39+24D) feature and baseline with (39+36D). I did the articulatory inversion using Feedforward and RNN nets.

FeedForward Configuration: 3 layers with 400 hidden nodes per layer.
Samples of 8 articulators predictions using feedforward net are shown as Fig.B.1 and B.2.

BRNN Configuration: one feedforward layer + two bidirectional LSTMs with 300 hidden nodes and 60 timesteps.
Samples of 8 articulators predictions using this architecture are shown as Fig.B.3 and B.4.
The left columns are coordinate x, and right columns are y. We can see that the predictions of MANm, MANi are not good in both architectures, which indicate again the pattern of articulators MANm, MANi is not as clear as others. The RNN predictions are better and smoother. All results are summarized in Table.5.9. Values in ( ) are root-mean-square er-

TABLE 5.9: *Summarized PER of GMM+HMM model for JW45 articulatory inversion*

type	real data	FF	BLSTM
24D	21.74	23.1(0.4431)	22.28(0.392)
36D	19.67	–	25.14(0.45)

ror (RMSE) and type 24D means the model uses MFCCs and features of T1,T2,T3,T4 articulators (39+24D). The baseline that only uses MFCCs is 24.93% of PER. When using 39+24D feature, both feedforward and bidirectional LSTM give improvements, whereas bidirectional LSTM is better than feedforward. However, when using 39+24D feature, the results of BLSTM is a little worse than only using MFCCs. There is not enough data for it to learn the mapping when articulators MANm, MANi are used.

Speaker independent
For speaker independent, all features(39+48D) are used to see the general performance compared with Generalized Distillation method.

FeedForward When the acoustic model and the articulatory inversion model are both FeedForward nets, the PER became **6.49%**.

BRNN When the acoustic model and the articulatory inversion model are both BRNN nets, the best PER of **4.18%** was achieved with the same structure of IFFBFFO, 2048 hidden nodes, 2048 cell nodes and 30% dropout, but the new bidirectional LSTMs as shown in Fig.4.5 are used here.

5.6 Cross Validation

Cross Validation was also experimented to make the conclusion more convincing. I also excluded 3 speakers who have very few utterances. Then split remaining 42 speakers into 7-fold cross validation, where test set contains 6 speakers. The number of utterances in train and test sets and GMM+HMM baseline results for all folds are shown in Table.5.10.

TABLE 5.10: *utterances of train/test sets and GMM+HMM baseline for all folds*

Fold ID	Train	Test	MFC	MFC+ART
0	2917	521	17.36	7.36
1	3005	433	20.82	10.99
2	2957	481	18.9	9.92
3	2940	498	18.6	9.02
4	2990	448	23.04	11.22
5	2941	497	20.84	10.45
6	2878	560	20.8	11.46
average	2947	491	20.05±3.05	10.06±1.82

The PER results got from different systems are collected in Table.5.11.

TABLE 5.11: *utterances of train/test sets and GMM+HMM baseline for all folds*

Fold ID	stuFF-stuTAR	stuFF-teaTAR	teaFF	FFinversion	FFdistill
0	6.23	5.93	3.56	5.08	5.56
1	8.06	7.3	4.73	6.51	7.02
2	8.03	7.49	4.59	6.68	6.91
3	7.07	6.1	3.86	5.27	5.47
4	9.9	9.16	5.65	7.72	8.65
5	9.64	8.61	5.29	7.77	8.16
6	8.69	8.15	5.47	7	7.47
average	8.23±1.49	7.53±1.27	4.74±0.55	6.58±0.98	7.03±1.24

The five systems shown above are:

1. **stuFF-stuTAR**: a student model based on feedforward network that has 4 layers with 2048 nodes per layer, 40% dropout, 256 batchsize and the inputs to network are only acoustic features (39D), the targets of HMM state are got from the alignment of the student GMM+HMM (39D) baseline system.

2. **stuFF-teaTAR**: a student model as same as the first one, but the the targets of HMM state are got from the alignment of the teacher GMM+HMM (39+48D) baseline system.

3. **teaFF**: a teacher model that has the same structure as the student model, but the inputs to this model are acoustic and articulatory features (39+48D), and the the targets of HMM state are got from the alignment of the teacher GMM+HMM (39+48D) baseline system.

4. **FFinversion**: a feedforward network for articulatory inversion which has 5 layers with 3072 nodes per layer, 50% dropout, 128 batchsize. after predicting the articulatory feature from the acoustic feature, the appended feature will be treated as the inputs to the teaFF system.

5. **FFdistill**: a student model as same as the first one, but has soft-targets generated from teaFF model during train. The plots showing the results using different T and λ for all folds are collected in C

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this work, four ASR systems with integrated articulatory and acoustic features were proposed and compared.

1. Distillation based on feedforward neural networks.
2. Distillation based on recurrent neural networks.
3. Articulatory inversion based on feedforward neural networks.
4. Articulatory inversion based on recurrent neural networks.

From this work we can observed that:

1. The conventional GMM-HMM system using only acoustic feature gets the PER of 20.05±3.05%. When real articulatory measurements are also used, the system got the PER of 10.06±1.82%. This is a relative PER reduction of **49.8%**, which also confirmed that articulatory information is very helpful for ASR system.

2. When replacing the GMM with feedforward neural network for acoustic modelling, MFC-feature system got a PER of 8.23±1.49%, MFC+ART feature system got a PER of 4.74±0.55%. Both systems got a big performance boost compared to the conventional GMM-HMM systems.

3. The ASR system based feedforward neural network without any help of articulatory information have the PER of 8.23±1.49. Surprisingly, when the hard targets during training of student DNN model are provided from the GMM-HMM baseline containing both acoustic and articulatory feature, about **8.5%** PER reduction can be achieved. This is a very simple but effective way to utilize articulatory information.

4. Furthermore, about **14.6%** PER reduction can be observed when distillation training are also applied. Distillation training using soft-targets provided by a "rich" informative teacher model provides the relationships between classes in output space and can be treated as a very good regularization method or pre-training method.

5. The articulatory inversion method based on feedforward neural network is the best system that achieves **20%** PER reduction. Dislike distillation method, inversion method provides more more informative input representation that helps system to distinguish different classes. However inversion method requires doubled computational cost and

time than distillation method, since it has another DNN model that learns the mapping between acoustic and articulatory feature spaces. Although it is the most accurate ASR system, but it is also the most expensive ASR system.

6. The same conclusions can also be observed when replacing the feedforward neural network with the Recurrent neural network. Recurrent neural network acoustic model using only acoustic feature can achieve the performance that MFC+ART-feature feedforward neural network gets and don't need to pre-defined the window-size, since it can learn which input is important or irrelevant.

6.2 Future Work

In this work, all systems preform as good as expected, but many parts can also be improved as below:

1. The articulatory feature used in this work is only the movements of articulators, which is very simple. However potential articulatory information can be lost in such representation. The next attempt is to use images of vocal tract through time. The dimension can be reduced using convolutional autoencoder.

2. There also are other issues to be investigated within this framework including the effect of the teacher performance on training set, more sophisticated ways of "teaching", not just linear combination of loss functions.

3. Recurrent neural network preforms extremely good, but its computational cost prevents it from widely using in practice. My next experiment could be transferring information from RNN to DNN model.

4. articulatory inversion based RNN is the best ASR system among the proposed ones, but it has to learn the mapping between acoustic and articulatory feature spaces, then learns the posterior probability $P(o|w)$, which seems to be redundant. Because neural network is a good technique for regression, I could learn the mapping between acoustic feature and the representation transformed from MFC+ART feature directly.

Appendix A

Phoneme list

TABLE A.1: *Phoneme list*

Phoneme	Example	Translation	Phoneme	Example	Translation
AA	odd	AA D	L	lee	L IY
AE	at	AE T	M	me	M IY
AH	hut	HH AH T	N	knee	N IY
AO	ought	AO T	NG	ping	P IH NG
AW	cow	K AW	OW	oat	OW T
AY	hide	HH AY D	OY	toy	T OY
B	be	B IY	P	pee	P IY
CH	cheese	CH IY Z	R	read	R IY D
D	dee	D IY	S	sea	S IY
DH	thee	DH IY	SH	she	SH IY
EH	Ed	EH D	T	tea	T IY
ER	hurt	HH ER T	TH	theta	TH EY T AH
EY	ate	EY T	UH	hood	HH UH D
F	fee	F IY	UW	two	T UW
G	green	G R IY N	V	vee	V IY
HH	he	HH IY	W	we	W IY
IH	it	IH T	Y	yield	Y IY L D
IY	eat	IY T	Z	zee	Z IY
JH	gee	JH IY	ZH	seizure	S IY ZH ER
K	key	K IY			

Appendix B

Articulatory Inversion Plots

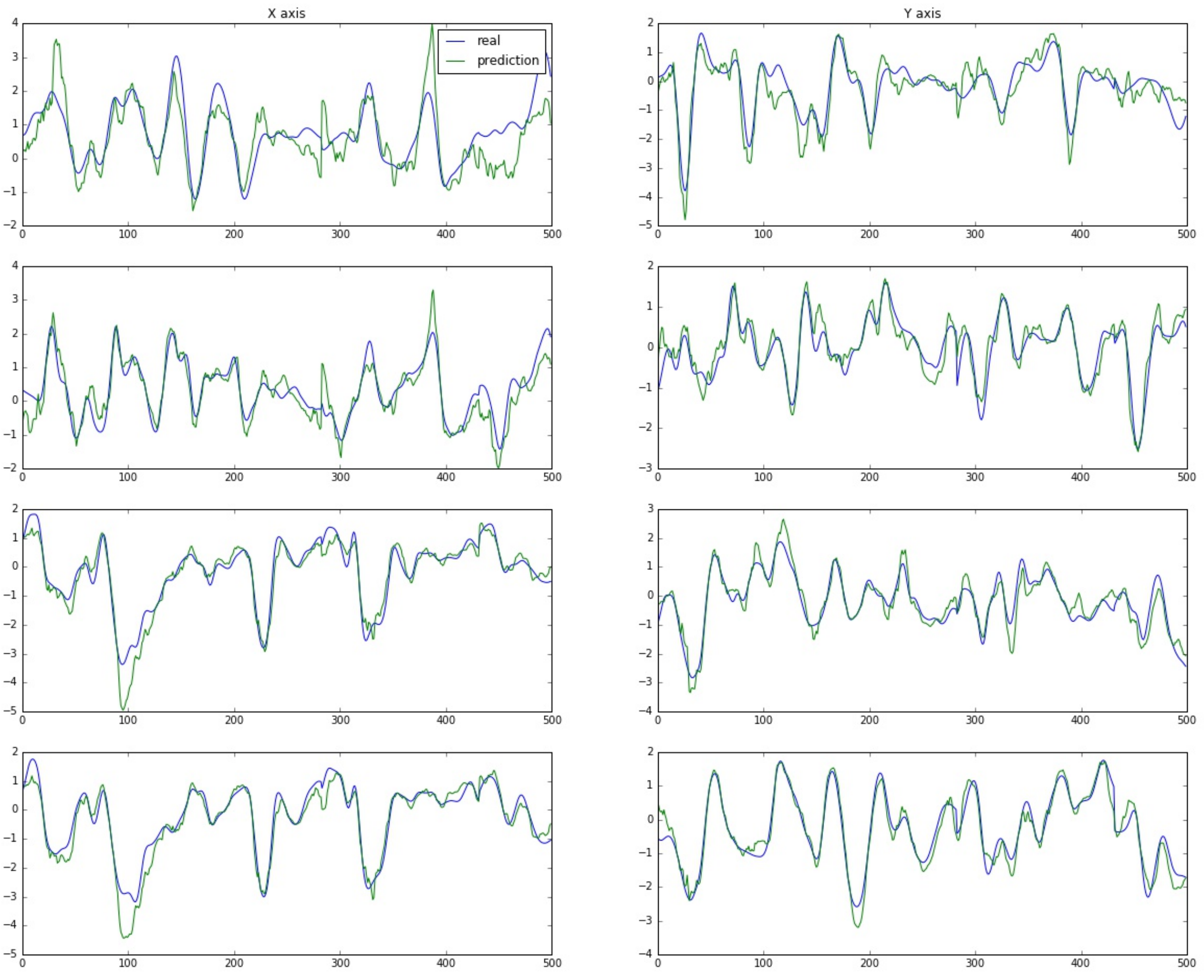


FIGURE B.1: *feedforward net predictions of articulators UL, LL, T1, T2*

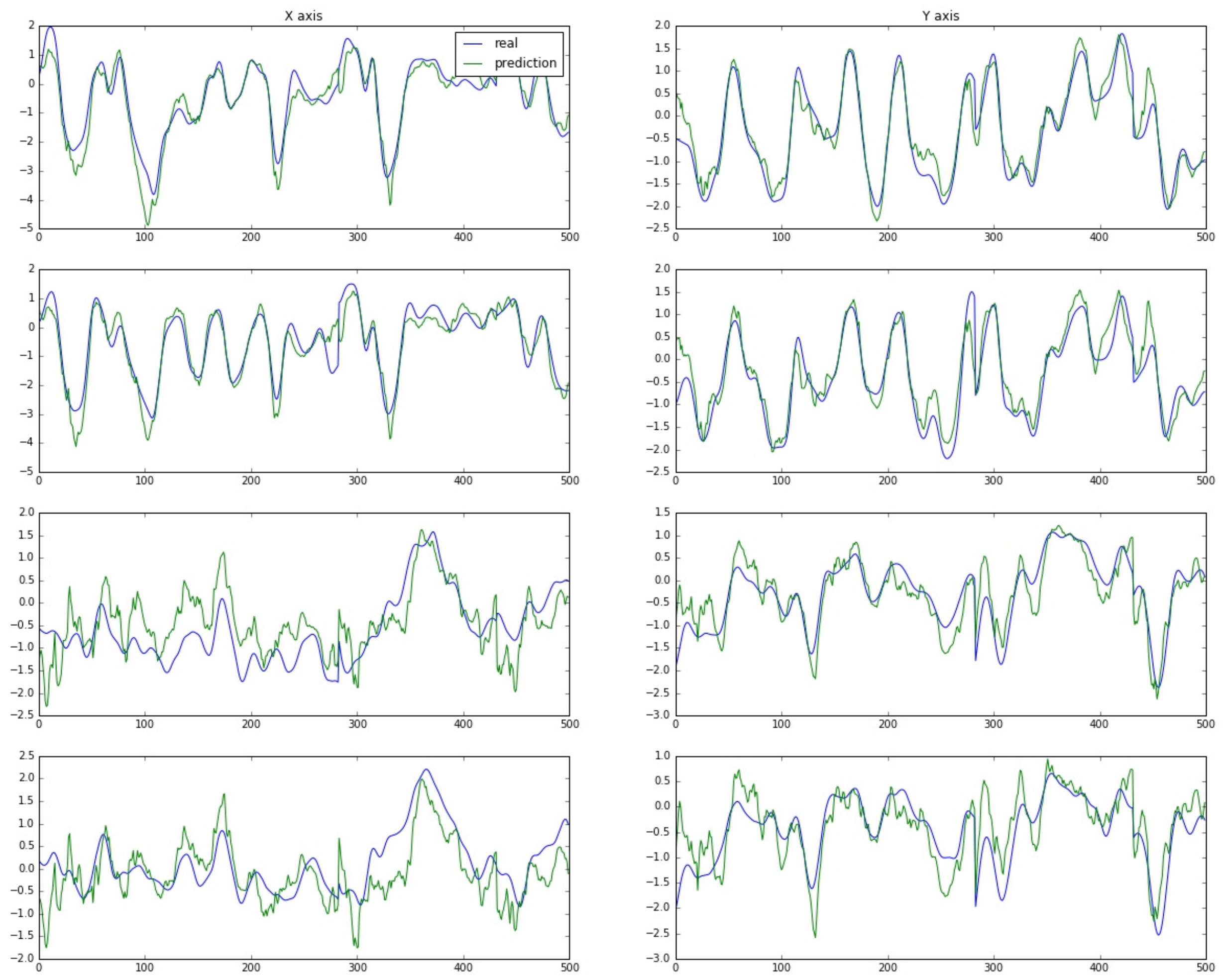


FIGURE B.2: *feedforward net predictions of articulators T3, T4, MANm, MANi*

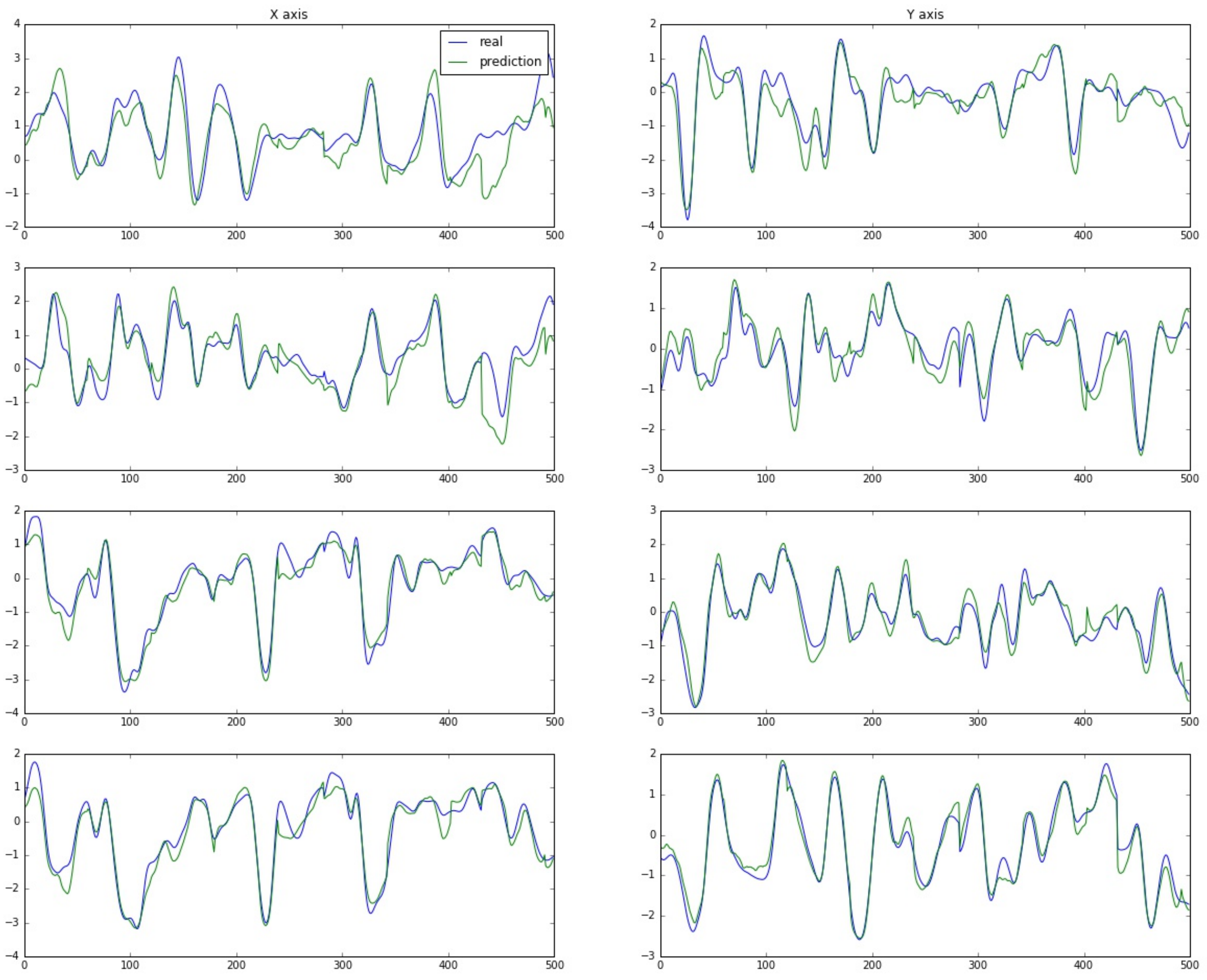


FIGURE B.3: *rnn net predictions of articulators UL, LL, T1, T2*

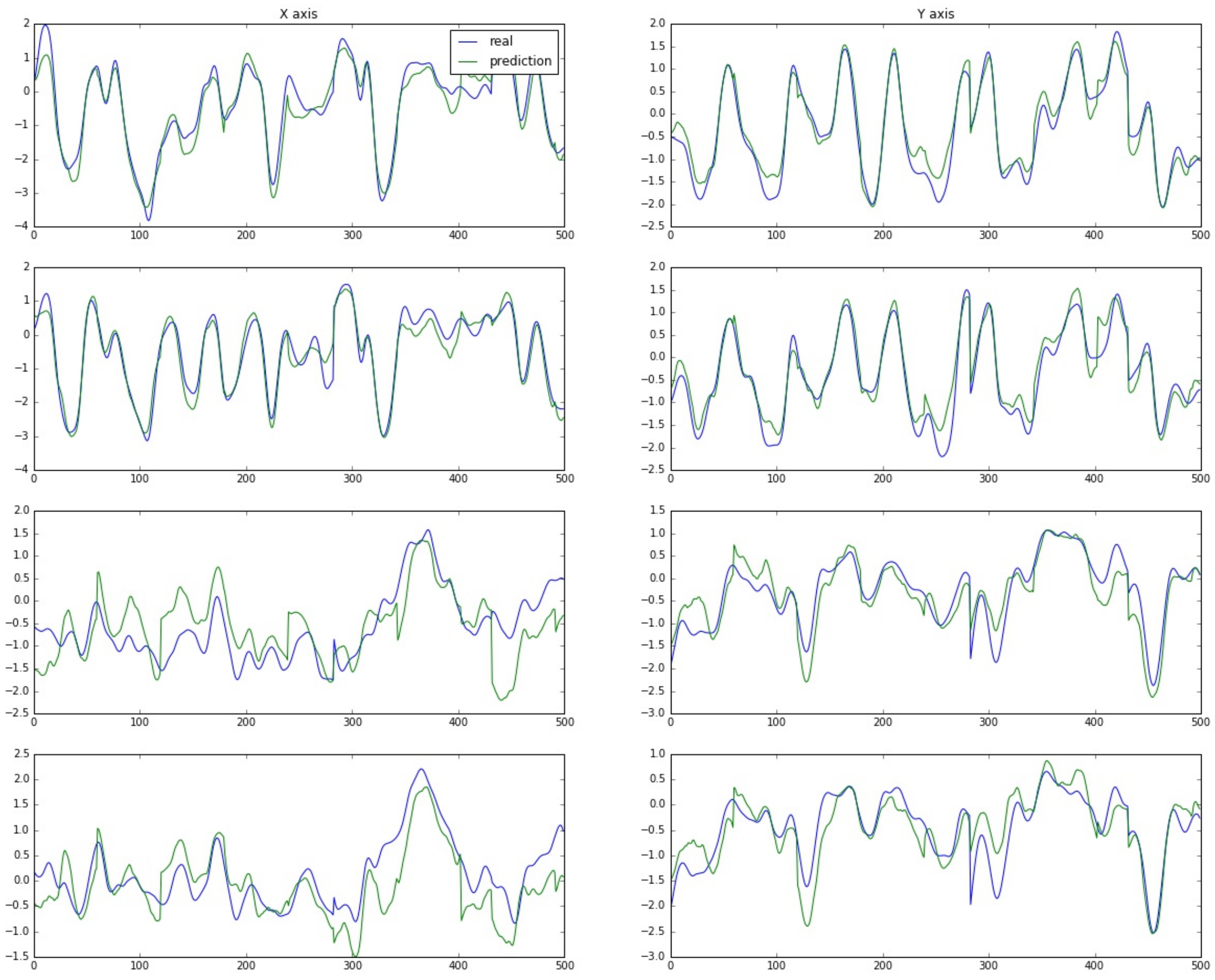


FIGURE B.4: *rnn net predictions of articulators T3, T4, MANm, MANi*

Appendix C

Distillation Plots

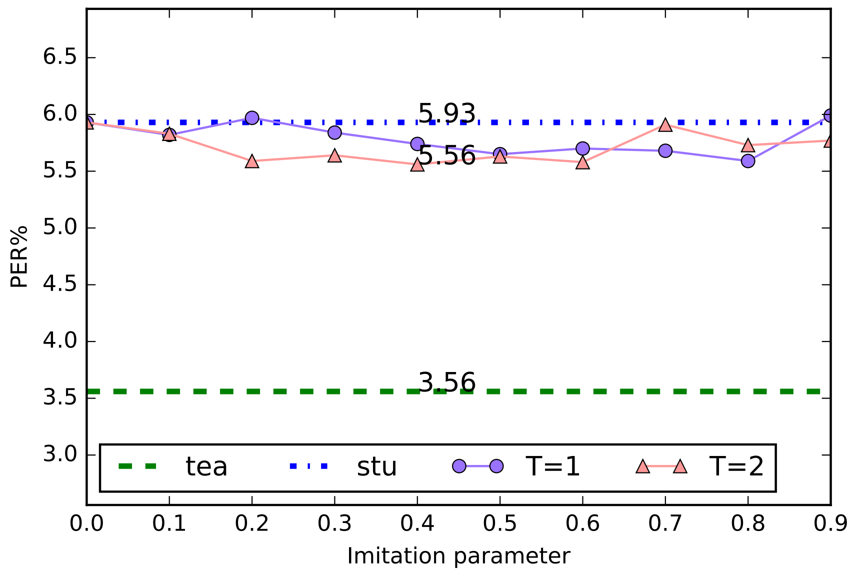


FIGURE C.1: *distillation results for fold 0*

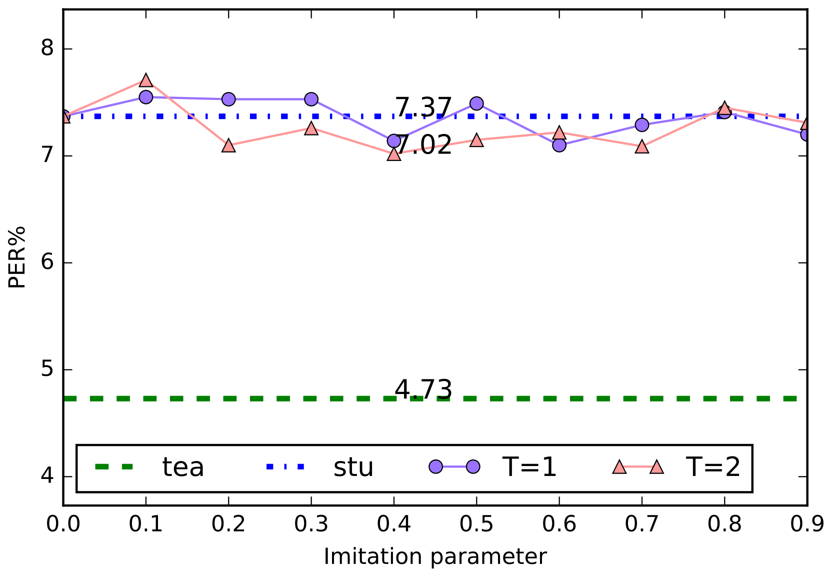


FIGURE C.2: *distillation results for fold 1*

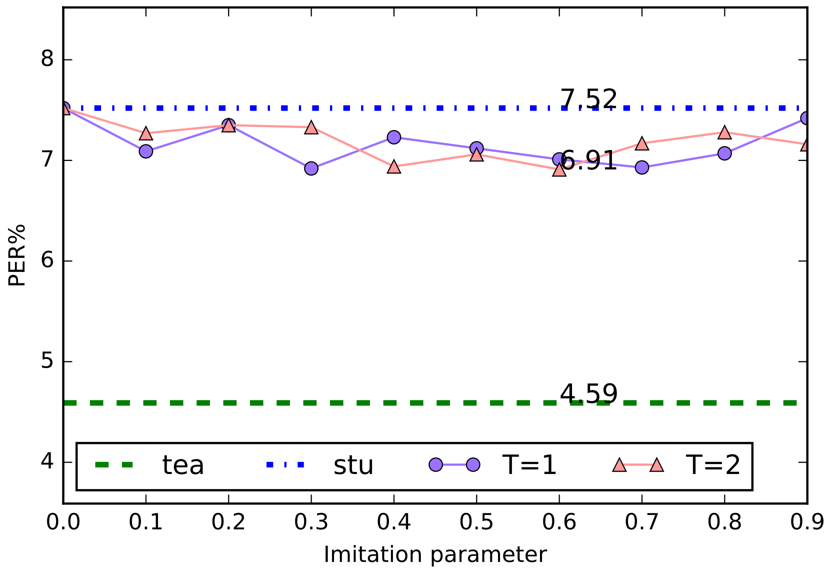


FIGURE C.3: *distillation results for fold 2*

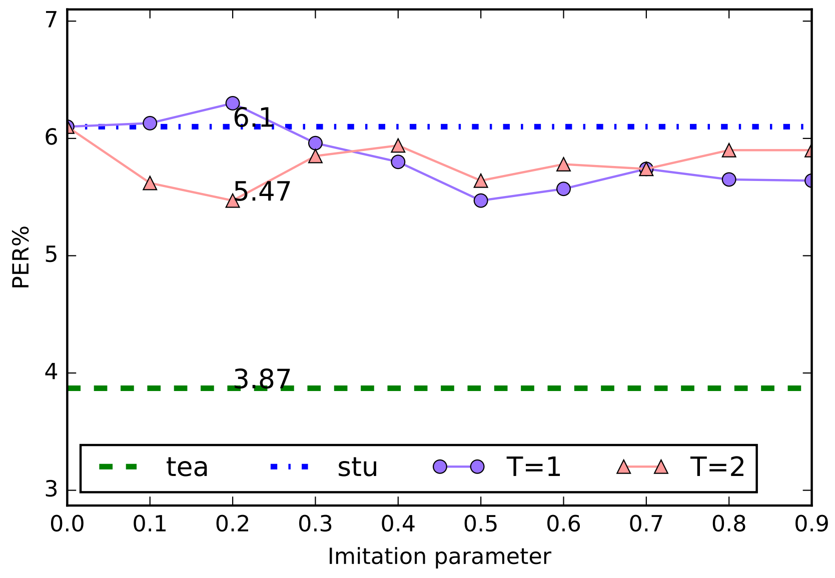


FIGURE C.4: *distillation results for fold 3*

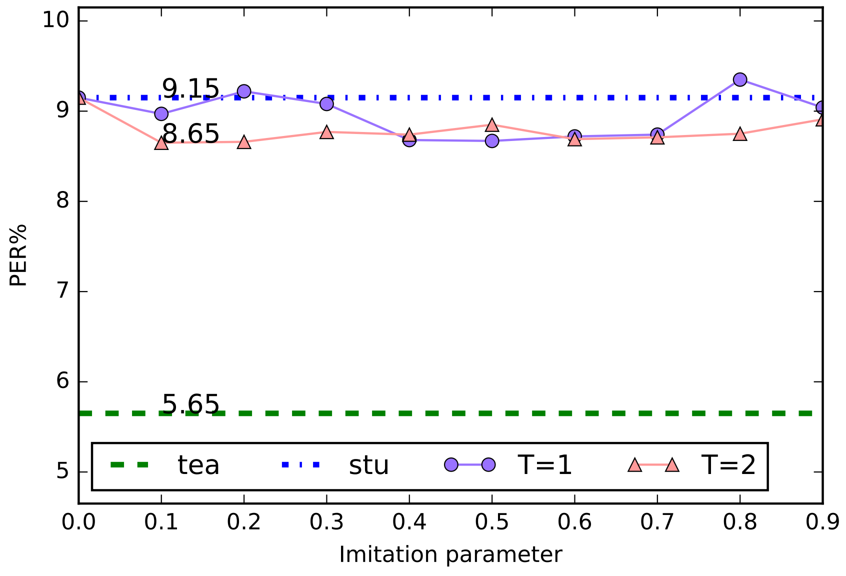


FIGURE C.5: *distillation results for fold 4*

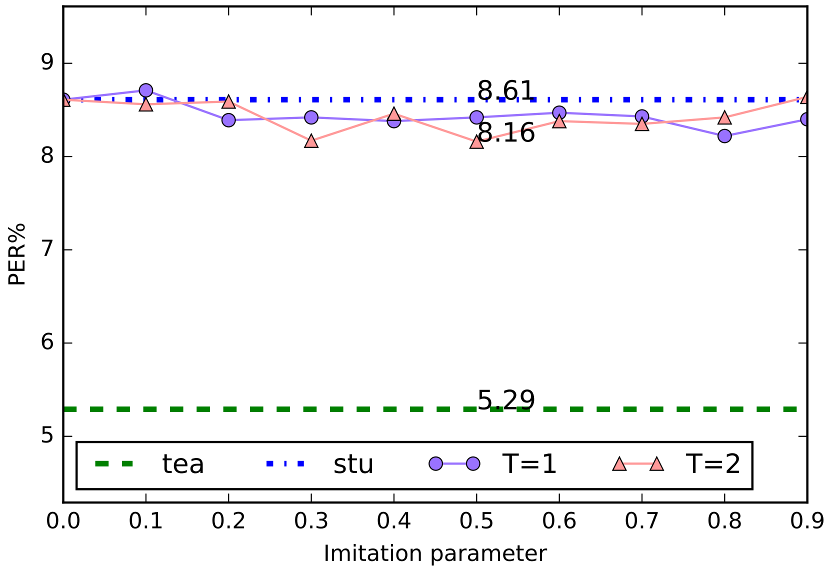


FIGURE C.6: *distillation results for fold 5*

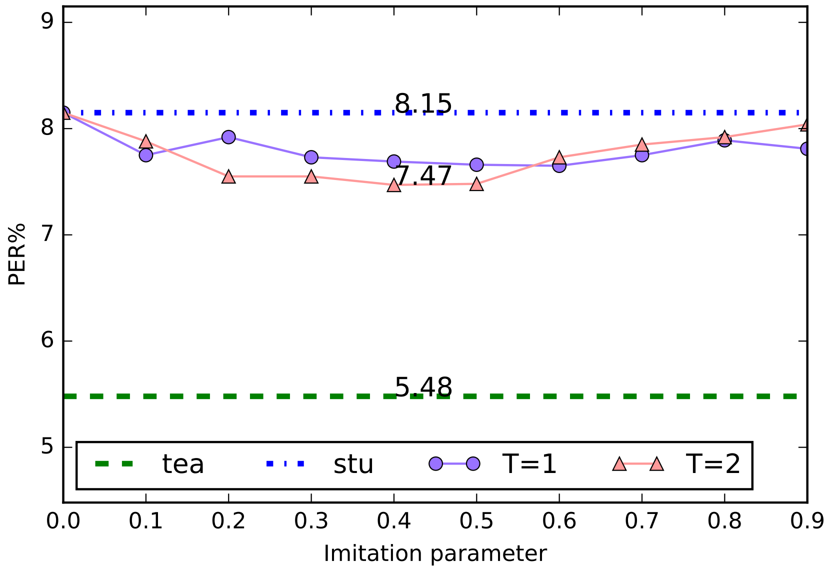


FIGURE C.7: *distillation results for fold 6*

Bibliography

[1] Akihiro Abe, Kazumasa Yamamoto, and Seiichi Nakagawa. "Robust Speech Recognition Using DNN-HMM Acoustic Model Combining Noise-Aware Training with Spectral Subtraction". In: *INTERSPEECH*. 2015.

[2] Rajkumar Arora and Karen Livescu. "Multi-view CCA-based acoustic features for phonetic recognition across speakers and domains". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 7135–7139.

[3] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. "Advances in optimizing recurrent networks". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8624–8628.

[4] Jeff A Bilmes et al. "A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models". In: (1998).

[5] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[6] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[7] David C.Lay. "Linear Algebra and Its Applications 4ed". In: (2012), pp. 62–67. URL: http://goo.gl/9PSzdi.

[8] Steven Davis and Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), pp. 357–366.

[9] Sean R Eddy. "Hidden markov models". In: *Current opinion in structural biology* 6.3 (1996), pp. 361–365.

[10] G David Forney. "The viterbi algorithm". In: *Proceedings of the IEEE* 61.3 (1973), pp. 268–278.

[11] William J Hardcastle and Nigel Hewlett. *Coarticulation: Theory, data and techniques*. Cambridge University Press, 2006.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[13] Andrej Karpathy. "The Unreasonable Effectiveness of Recurrent Neural Networks". In: (). URL: https://goo.gl/hwPVkF.

[14] Slava Katz. "Estimation of probabilities from sparse data for the language model component of a speech recognizer". In: *IEEE transactions on acoustics, speech, and signal processing* 35.3 (1987), pp. 400–401.

[15] Diederik Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[16] Bo Li and Khe Chai Sim. “Modeling long temporal contexts for robust DNN-based speech recognition”. In: *INTERSPEECH*. 2014, pp. 353–357.

[17] Peng Liu et al. “A deep recurrent approach for acoustic-to-articulatory inversion”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE. 2015, pp. 4450–4454.

[18] David Lopez-Paz et al. “Unifying distillation and privileged information”. In: *ICLR*. 2016.

[19] Konstantin Markov, Jianwu Dang, and Satoshi Nakamura. “Integration of articulatory and spectrum features based on the hybrid HMM/BN modeling framework”. In: *Speech Communication* 48.2 (2006), pp. 161–175.

[20] James Martens. “Deep learning via Hessian-free optimization”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 735–742.

[21] Hrushikesh Narhar Mhaskar and Charles A Micchelli. “How to choose an activation function”. In: *Advances in Neural Information Processing Systems* (1994), pp. 319–319.

[22] Vikramjit Mitra et al. “Articulatory information for noise robust speech recognition”. In: *IEEE Transactions on Audio, Speech, and Language Processing* 19.7 (2011), pp. 1913–1924.

[23] Vikramjit Mitra et al. “Gesture-based dynamic Bayesian network for noise robust speech recognition”. In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2011, pp. 5172–5175.

[24] Vikramjit Mitra et al. “Recognizing articulatory gestures from speech for robust speech recognition”. In: *The Journal of the Acoustical Society of America* 131.3 (2012), pp. 2270–2287.

[25] Hosung Nam et al. “A procedure for estimating gestural scores from speech acoustics”. In: *The Journal of the Acoustical Society of America* 132.6 (2012), pp. 3980–3989.

[26] Christopher Olah. “NN-Manifolds-Topology”. In: (). URL: http://goo.gl/yA7kUW.

[27] Christopher Olah. “Understanding LSTMs”. In: (). URL: http://goo.gl/hwG4OC.

[28] Sankaran Panchapagesan and Abeer Alwan. “A study of acoustic-to-articulatory inversion of speech by analysis-by-synthesis using chain matrices and the Maeda articulatory model”. In: *The Journal of the Acoustical Society of America* 129.4 (2011), pp. 2144–2162.

[29] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training recurrent neural networks.” In: *ICML (3)* 28 (2013), pp. 1310–1318.

[30] Jessica Ray, Brian Thompson, and Wade Shen. "Comparing a high and low-level deep neural network implementation for automatic speech recognition". In: *High Performance Technical Computing in Dynamic Languages (HPTCDL), 2014 First Workshop for*. IEEE. 2014, pp. 41–46.

[31] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (). URL: http://goo.gl/MYZmxA.

[32] Jürgen Schmidhuber. "Deep learning in neural networks: An overview". In: *Neural Networks* 61 (2015), pp. 85–117.

[33] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[34] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th international conference on machine learning (ICML-13)*. 2013, pp. 1139–1147.

[35] Tijmen Tieleman and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". In: *COURSERA: Neural Networks for Machine Learning* 4 (2012), p. 2.

[36] Tomoki Toda, Alan W Black, and Keiichi Tokuda. "Acoustic-to-articulatory inversion mapping with Gaussian mixture model." In: *INTERSPEECH*. 2004.

[37] Carnegie Mellon University. "The CMU dictionary". In: (). URL: http://goo.gl/xW7VAM.

[38] Benigno Uria et al. "Deep Architectures for Articulatory Inversion." In: *INTERSPEECH*. 2012, pp. 867–870.

[39] Vladimir Vapnik and Rauf Izmailov. "Learning Using Privileged Information: Similarity Control and Knowledge Transfer". In: *Journal of Machine Learning Research* 16 (2015), pp. 2023–2049.

[40] Vladimir Vapnik and Akshay Vashist. "A new learning paradigm: Learning using privileged information". In: *Neural Networks* 22.5 (2009), pp. 544–557.

[41] Weiran Wang, Raman Arora, and Karen Livescu. "RECONSTRUCTION OF ARTICULATORY MEASUREMENTS WITH SMOOTHED LOW-RANK MATRIX COMPLETION". In: ().

[42] Lloyd R Welch. "Hidden Markov models and the Baum-Welch algorithm". In: *IEEE Information Theory Society Newsletter* 53.4 (2003), pp. 10–13.

[43] J Westbury. "X-RAY MICROBEAM SPEECH PRODUCTION DATABASE USER'S HANDBOOK. 1994". In: *Waisman Center, University of Wisconsin: Madison, USA* (), pp. 1–100.

[44] Jiahong Yuan and Mark Liberman. "Speaker identification on the SCOTUS corpus". In: *Journal of the Acoustical Society of America* 123.5 (2008), p. 3878.

[45] Wojciech Zaremba. "An empirical exploration of recurrent network architectures". In: (2015).

[46] Le Zhang and Steve Renals. "Acoustic-articulatory modeling with the trajectory HMM". In: *IEEE Signal Processing Letters* 15 (2008), pp. 245–248.

[47] Pengcheng Zhu, Lei Xie, and Yunlin Chen. "Articulatory Movement Prediction Using Deep Bidirectional Long Short-Term Memory Based Recurrent Neural Networks and Word/Phone Embeddings". In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.