Karlsruhe Institute of Technology

Neural Network-based Small-Footprint Flexible Keyword Spotting

Master Thesis of

Linchen Zhu

At the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Prof. Dr. Tamim Asfour
Advisor:	M.Sc. Markus Müller
Second advisor:	Dr. Sebastian Stüker

Duration: 1st March 2017 – 29th August 2017

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 29.08.2017

.....

(**Linchen Zhu**)

Abstract

Real-time keyword spotting (KWS) on mobile devices requires a small memory footprint, low latency, and low computational cost. Conventional approaches such as HMM-based KWS do not fulfill these requirements, as Viterbi decoding tends to lead to high computational cost. Therefore, this work proposes a neural network-based small-footprint flexible KWS system appropriate for mobile devices.

Our novel KWS system is composed of three modules: the feature extraction module, the posterior probability estimation module, and the posterior handling module. The feature extraction module produces acoustic features from the input audio stream, while the posterior probability estimation module generates posterior probabilities for subword units such as phonemes and senones using various types of neural networks including feed-forward neural networks and recurrent neural networks such as LSTM, TDNN-LSTM, and so on. Finally, the posterior handling module calculates final confidence scores for the predefined keywords based on the posteriors from networks.

Experiments have been performed to prove the effectiveness of this KWS approach. Experimental results on neural network training show that RNNs such as LSTM, TDNN-LSTM, etc., outperform FFNNs significantly on framewise phoneme/senone classification. Then, the KWS system is built based on the trained networks, and the performance is evaluated by detecting the predefined key-phrase ”okay cosa”. Evaluation results demonstrate that this KWS system achieves lower computational cost and a smaller memory footprint compared with conventional KWS approaches, and therefore makes a step towards real-time KWS on mobile devices.

Zusammenfassung

Das Echtzeit-Keyword-Spotting (KWS) auf Mobilgeräten erfordert einen kleinen Speicherbedarf, eine geringe Latenz und niedrige Rechenkosten. Konventionelle Ansätze wie HMM-basiertes KWS erfüllen diese Anforderungen nicht, weil die Viterbi-Decodierung zu hohen Rechenkosten führt. Daher schlägt diese Arbeit ein neuronales Netzwerk-basiertes flexibles KWS-System vor, das für mobile Geräte geeignet ist.

Unser neuartiges KWS-System besteht aus drei Modulen: dem Merkmalsextraktionsmodul, dem a posteriori Wahrscheinlichkeitsschätzungsmodul und dem a posteriori Wahrscheinlichkeitshandhabungsmodul. Das Merkmalsextraktionsmodul erzeugt akustische Merkmale aus dem Eingangs-Audiostrom, und das a posteriori Wahrscheinlichkeitsschätzungsmodul erzeugt a posteriori Wahrscheinlichkeiten für Subworteinheiten wie Phoneme und Senone mit verschiedenen Arten von neuronalen Netzwerken, einschließlich feed-forward neuronale Netze und rekurrente neuronale Netze wie LSTM, TDNN-LSTM und so weiter. Schließlich berechnet das a posteriori Wahrscheinlichkeitshandhabungsmodul endgültige Vertrauenswerte für die vordefinierten Schlüsselwörter basierend auf den a posteriori Wahrscheinlichkeiten aus den Netzwerken.

Es wurden Experimente durchgeführt, um die Wirksamkeit dieses KWS-Ansatzes zu beweisen. Experimentelle Ergebnisse für das neuronale Netzwerktraining zeigen, dass rekurrente neuronale Netze wie LSTM, TDNN-LSTM, etc., feed-forward neuronale Netze bemerkenswert über die frameweise Phonem/Senon-Klassifizierung übertreffen. Dann wird das KWS-System auf der Basis von trainierten Netzwerken aufgebaut und die Performance wird durch die Erkennung der vordefinierten Key-Phrase ”okay cosa” ausgewertet. Auswertungsergebnisse zeigen, dass dieses KWS-System im Vergleich zu konventionellen KWS-Ansätzen niedrigere Rechenkosten und einen kleineren Speicherbedarf erzielt und damit einen Schritt in Richtung Echtzeit-KWS auf Mobilgeräten macht.

Acknowledgements

First of all, I would like to thank Prof. Dr. Alexander Waibel for giving me the opportunity to perform the research at the Interactive Systems Labs for this thesis. I would also like to express my thanks to my advisors Markus Müller and Dr. Sebastian Stüker for their valuable suggestions and constant support during this thesis. Finally, I would like to thank Bastian Krüger for helping me with the data collection website.

Contents

1. Introduction

1.1 Motivation

Keyword spotting (KWS), as an important branch of automatic speech recognition (ASR), is a task aimed at detecting specific keywords in a continuous speech audio stream. Like ASR, KWS has a wide range of applications; for instance, voice control systems utilize KWS technology to detect whether predefined command words appear in a voice stream. Conventional large-vocabulary continuous speech recognition (LVCSR) based KWS systems employ an ASR system to convert the human speech signal into distinct word sequences or rich lattices, and after that, an efficient search engine is employed to detect the keywords in word sequences or lattices. However, unlike ASR tasks, for KWS tasks, it is not necessary to recognize all words contained in the vocabulary within utterances. Therefore, in order to save computing resources, the Keyword/Filler HMM-based KWS approach only recognizes predefined keywords within utterances.

In recent times, KWS has been used on an increasing number of applications on mobile devices. For example, the application "okay google" allows the users to waken their mobile devices through uttering the key-phrase "okay google". Such KWS systems are required to run in the background and continuously deliver the detection of the selected phrase in real-time using the limited computational resources provided by mobile devices. Therefore, real-time KWS systems on mobile devices should fulfill the requirements of having a small memory footprint, low latency, and low computational cost. However, conventional LVCSR and Keyword/Filler HMM based KWS approaches do not meet these requirements, as a great deal of computation is needed for Viterbi decoding. In 2014, a small-footprint KWS system, also known as Deep KWS, was proposed by Chen et.al[CPH14]; this KWS system achieves low latency and computational cost by replacing Viterbi decoding with simple confidence calculation based on posteriors from the deep neural network (DNN). Despite its utilization of a fast and efficient detecting algorithm, the Deep KWS retains some disadvantages. For instance, a considerable number of training examples are needed for each keyword; moreover, the system is not flexible, as new DNNs must be trained in order to detect new keywords. Thus, in this work, a novel KWS approach is proposed that is appropriate for real-time KWS on mobile devices; furthermore, our KWS system is also able to deal with the problems inherent on the Deep KWS system.

1.2 Contribution

This work proposes a novel neural network-based small-footprint flexible KWS system. With the help of this approach, a KWS system with a small memory footprint, low latency and low computational cost can be developed as a step towards real-time KWS on mobile devices. This KWS approach is also flexible, as a well-trained network can be employed to detect all kinds of keywords/key-phrases, and a large number of training examples per keyword/key-phrase are not necessary. Sequence classification using different types of NNs is also studied in this work. Comprehensive experiments have been carried out to explore the classification performance of recurrent neural networks (RNN), including LSTM, BLSTM and TDNN-LSTM. As RNNs improve the classification accuracy of feed-forward neural networks (FFNN) to a remarkable extent, the posteriors generated by RNNs can be also utilized in general ASR tasks.

1.3 Layout

In the next two chapters, some background knowledge about neural networks and acoustic modeling is provided, due to the fact that our KWS approach is based on various types of NNs and acoustic models. Chapter two briefly discusses the feed-forward neural network, then examines recurrent neural networks, in particular bidirectional RNN and LSTM. Chapter three briefly describes the basics of acoustic modeling, including context dependent acoustic modeling. In chapter four, the problem of real-time KWS on mobile devices and previous work in this area is discussed, at which point our novel KWS approach is introduced. The fifth chapter documents the implementation details of our KWS system. The experimental setup used to evaluate the KWS approach and the ensuing evaluation results are provided in chapter six. Finally, chapter seven concludes this work and gives a short discussion of potential further work.

2. Neural Networks

An artificial neural network (ANN), also referred to as neural network (NN), is one of the most important classes of machine learning algorithms. This chapter provides the background knowledge of various kinds of neural networks. Section 2.1 first introduces the feed-forward neural network (FFNN). After that, section 2.2 gives a brief description of the recurrent neural network (RNN), followed by an introduction to extensions of the RNN, namely the bidirectional RNN (BRNN) and the long short-term memory (LSTM).

2.1 Feed-Forward Neural Networks

The idea of the ANN is derived from the structure and functioning of the biological brain[Ros61]. Like the biological brain, an ANN consists of a large number of computing units called neurons or nodes. Moreover, neurons in the ANN are connected with each other according to different weights. The FFNN is a class of NNs, in which the connections of neurons are not allowed to form a circle.

2.1.1 Feed-Forward Pass

A FFNN performs a nonlinear transformation which maps the input vectors to the output vectors. According to the universal approximation theorem[HSW89], a FFNN with a single hidden layer has the capability to approximate any continuous function to arbitrary

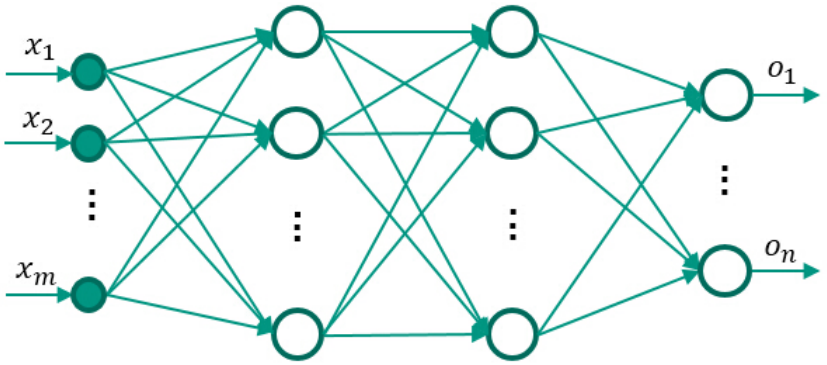

Figure 2.1: *Architecture of a FFNN with an input layer, two hidden layers and an output layer.* (Figure originated from [Zhu15])

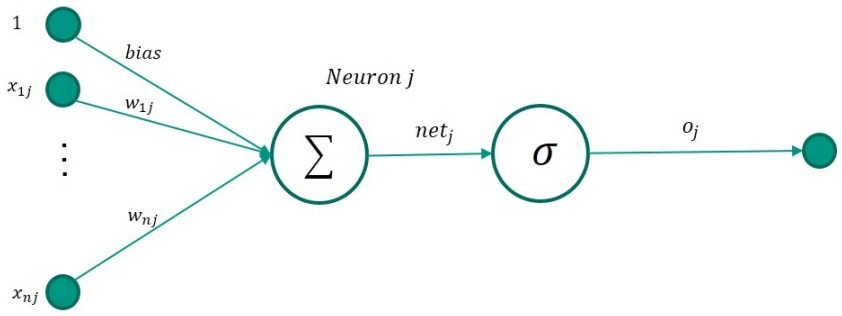

Figure 2.2: *Architecture of neuron j, the neuron first computes the weighted sum of input signals and a bias term, after applying the activation function σ the output signal of neuron j is obtained.*

accuracy. Thus, FFNNs are suitable for a variety of classification and regression tasks. The classification or regression is performed by projecting the activations from the input layer of the FFNN to the output layer through one or more hidden layers. This process is also called the feed-forward pass. Neurons in the FFNN are arranged in layers as shown in figure 2.1. Furthermore, neurons in the same layer do not have connections with each other, they only have connections with neurons in neighboring layers.

Figure 2.2 illustrates a single neuron j in the FFNN. The neuron j receives the input signal from input neurons in the previous layer, and generates an output signal to the output neurons in the subsequent layer. The neuron j is connected with other neurons according to different weights, and a higher weight corresponds to a stronger connection between two neurons. Furthermore, x_{ij} represents the input signal from neuron i to neuron j, and w_{ij} is the connection weight between neuron i and neuron j. The net input of neuron j is obtained by calculating the weighted sum of all the input signals and a bias term b, namely $net_j = \sum_i x_{ij} * w_{ij} + b$. Then the output signal of neuron j is computed: $o_j = \sigma(net_j)$, where σ is the activation function used to provide nonlinearity to neuron j, and the bias b shifts the activation function horizontally.

Various activation functions have been proposed, some of the most commonly used activation functions are sigmoid(x) = $\frac{1}{1+e^{-x}}$, tanh(x) = $\frac{e^x-e^{-x}}{e^x+e^{-x}}$, rectified linear unit ReLU(x) = $max(0, x)$, etc. Recent studies show that ReLU can solve the problem of vanishing gradient and result in faster learning[ZRM+13][MHN13]. Another special activation function is the softmax function [Bri90]:

$$softmax(net_i) = \frac{e^{net_i}}{\sum_o e^{net_o}},$$

which is exclusively used in the output layer to ensure that the activation of each output neuron is positive or zero, and the sum of all the activations of the output layer is one.

For a given input vector of the FFNN, it is first placed in the input layer of the FFNN. Each neuron in the first hidden layer is connected to all the neurons in the input layer according to their weight matrices respectively. Then the activation of each neuron in the first hidden layer is calculated using the formula $o_i = \sigma(net_i)$, and the activations are also regarded as the inputs of the second hidden layer. Similarly, the second hidden layer is also fully connected to the first hidden layer, and the activations of the second hidden layer are calculated in the same way. By this means, the input vector of the FFNN is propagated forward from the input layer to the output layer. Normally the FFNN outputs a discrete probability distribution due to the softmax activation function of the output layer.

2.1.2 Backpropagation Training

As mentioned in section 2.1.1, a FFNN is a universal approximator which can approximate any continuous function to arbitrary accuracy. For a specific classification or regression task, the FFNN is required to learn a specific function mapping input feature vectors to targets. The initial weights of the FFNN are normally initialized with random values drawn from a zero-mean probability distribution. Therefore, it is necessary to adjust the weights of the FFNN, so that the FFNN can approximate the corresponding function. This process is also called the training or learning process of neural networks, and the error backpropagation (BP) algorithm is the dominate algorithm used to train FFNNs[RHW88][WZ95][Wer88].

As the BP algorithm belongs to the category of supervised learning algorithms, it requires a set of labeled training instances. For a given training instance x, the configuration of the network weights is $\mathbf{w}$, the output and target of output neuron k for training instance x are o_{kx} and t_{kx} respectively. Then the error or loss of the network for this training instance can be computed with the help of different loss functions. Some of the most widely used error functions are the sum-of-squares error function:

$$E_x(\mathbf{w}) = \frac{1}{2} \sum_{k \in outputs} (t_{kx} - o_{kx})^2,$$

and the cross-entropy error function[Bis95]:

$$E_x(\mathbf{w}) = - \sum_{k \in outputs} \left[t_{kx} \log(o_{kx}) + (1 - t_{kx}) \log\left(1 - o_{kx}\right) \right].$$

Moreover, the total error of the network on the training dataset X can be calculated as the sum of errors for all training instances:

$$E(\mathbf{w}) = \sum_{x \in X} E_x(\mathbf{w}).$$

The goal of the BP training algorithm is to find the optimal configuration of the network weights which minimize the error E based on the gradient descent method. In the case of a great number of training instances, it is not efficient to update the network weights using the gradient computed on the entire training dataset. It is, however, beneficial to calculate the gradient using a single training instance or a mini-batch of instances, and then update the network weights according the gradient. These variants are also known as stochastic gradient descent (SGD) and mini-batch gradient descent (MGD) [Mit97].

The BP algorithm provides a very efficient way to calculate the gradients in neural networks. The core idea of the BP algorithm is to propagate the error of the network from the output layer back through the hidden layers. In the following derivation of the BP algorithm we assume the error function is the sum-of-squares error function, and the activation function is the sigmoid function.

Given a training instance x, the error of the network is E_x, according to the gradient descent method, the weight w_{ij} is updated:

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E_x}{\partial w_{ij}},$$

where η is the learning rate. After applying the chain rule, the partial derivative can be expanded:

$$\frac{\partial E_x}{\partial w_{ij}} = \frac{\partial E_x}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E_x}{\partial net_j}x_{ij}.$$

For the convenience of notation, we define an error term $\delta_j = -\frac{\partial E_x}{\partial net_j}$. Again with the help of chain rule, we obtain the error term of output neuron j :

$$\delta_j = (t_{jx} - o_{jx})o_{jx}(1 - o_{jx}),$$

where o_{jx} is the output signal of neuron j in the output layer and t_{jx} is the target of neuron j for the instance x, since

$$\frac{\partial E_x}{\partial net_j} = \frac{\partial E_x}{\partial o_j}\frac{\partial o_j}{\partial net_j},$$

$$\frac{\partial E_x}{\partial o_j} = -(t_{jx} - o_j),$$

$$\frac{\partial o_j}{\partial net_j} = o_j(1 - o_j).$$

Furthermore, the error term of hidden neuron j can be calculated based on the neurons which take activations of neuron j as input. We denote those downstream neurons of neuron j by $Ds(j)$, then the error term of hidden neuron j can be calculated:

$$\delta_j = o_j(1 - o_j)\sum_{k \in Ds(j)} \delta_k w_{jk},$$

because

$$\frac{\partial E_x}{\partial net_j} = \sum_{k \in Ds(j)} \frac{\partial E_x}{\partial net_k}\frac{\partial net_k}{\partial net_j},$$

$$\frac{\partial E_x}{\partial net_k} = -\delta_k,$$

$$\frac{\partial net_k}{\partial net_j} = \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j} = w_{jk}o_j(1 - o_j).$$

Finally, the weight w_{ij} can be updated: $w_{ij} \leftarrow w_{ij} + \eta\delta_j x_{ij}$.

2.2 Recurrent Neural Networks

In this section we introduce another important class of neural networks, which is the recurrent neural network. We first discuss the forward pass of the RNN, followed by the training algorithm for the RNN, namely the backpropagation through time algorithm (BPTT). After that, the bidirectional RNN (BRNN) and the long short-term memory (LSTM) are introduced as powerful extensions of the RNN.

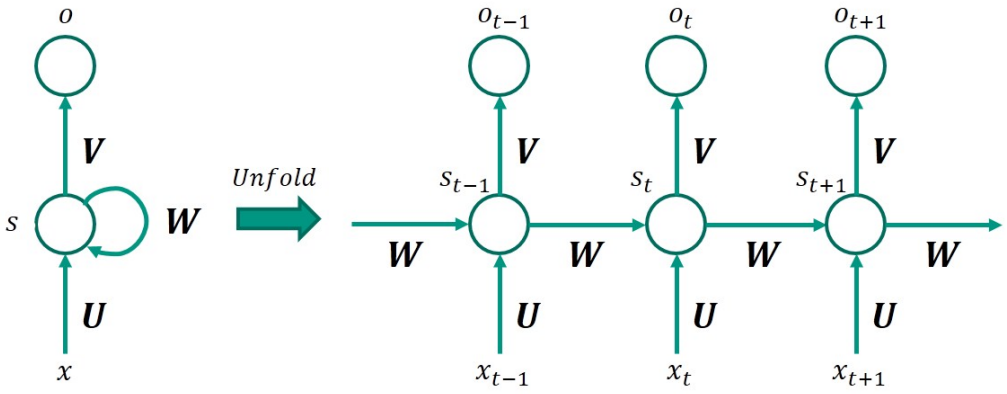

Figure 2.3: *Architecture of an RNN with one hidden layer, and the RNN is unfolded through three time steps.*

2.2.1 Forward Pass

Similar to the FFNN, the RNN is also composed of a number of basic computational units, namely the neurons, and neurons in the RNN are connected with each other through different weights. However, different from the FFNN, the RNN allows directed connections between neurons to form cycles, since the RNN is designed to solve the problems of learning time series data and modeling sequences of information. Figure 2.3 depicts an example of the RNN which is composed of one hidden layer and the unfolded RNN through time as well.

The input of the RNN is a sequence of feature vectors, x_t represents the input vector at the time step t. Moreover, the RNN has also a hidden layer, s_t denotes the states of the hidden layer at the time step t, which is equivalent to the memory of the RNN. Then, s_t is calculated based on the hidden states at the previous time step t-1 and the input at the current time step t:

$$s_t = f(Ux_t + Ws_{t-1}),$$

where f is the activation function, and W, U are the hidden-state-to-hidden-state weight matrix and the input-to-hidden-state weight matrix. Finally, the output of the RNN at time step t o_t is obtained:

$$o_t = softmax(Vs_t),$$

where V denotes the hidden-state-to-output weight matrix. In this way, the RNN accepts a sequence of input vectors, and generates a sequence of output vectors by calculating an output vector at each time step.

2.2.2 Backpropagation Through Time Training

Section 2.1.2 details the BP algorithm for training FFNNs, RNNs can be trained using the backpropagation through time algorithm (BPTT) in a quite similar way[Wer90]. The RNN takes a feature vector sequence $x_1, ..., x_n$ with the length n, and the corresponding target label sequence $t_1, ..., t_n$ as a training example.

As can be seen in figure 2.4, the recurrent hidden layers of the RNN are first unfolded n times by sharing the same weight matrices. Initial states s_0 of recurrent hidden layers are

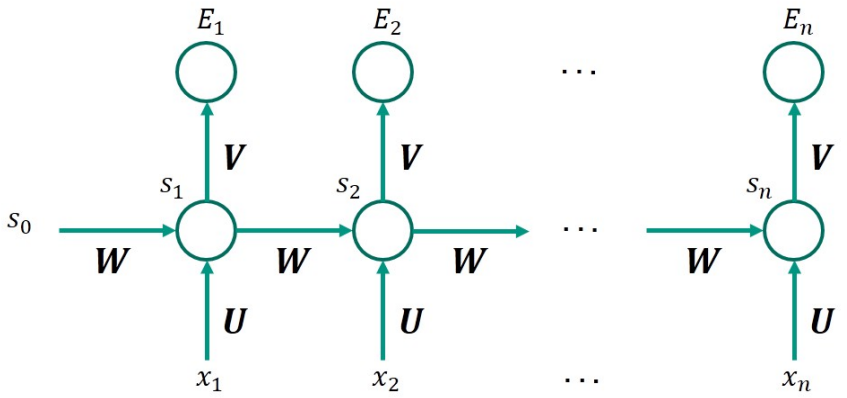

Figure 2.4: *Backpropagation through time algorithm, the RNN is first unrolled through time, then at each time step the error E_t is calculated, and the total error for the training sequence is $E = \sum_t E_t$.*

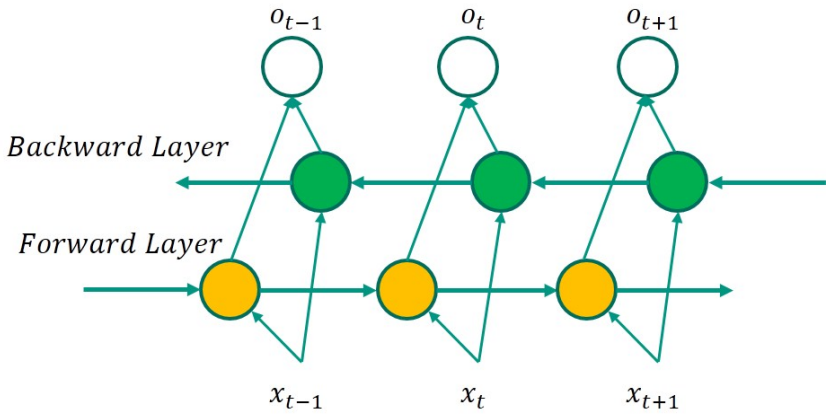

Figure 2.5: *Architecture of a Bidirectional RNN, the hidden recurrent layer of BRNN is composed of a forward layer and a backward layer, the training sequence is processed separately by the forward layer and the backward layer in opposite directions.*

typically set to zeros. At each time step, the RNN computes the current hidden states based on the current input vector and the hidden states at the previous time step, and then generates an output vector. For example, at the time step t the RNN computes the hidden states s_t using the input feature vector x_1 and the hidden states s_{t-1}. Then the RNN generates the output o_t, and the error at time step t E_t can be calculated based on o_t and t_t.

Since the entire training vector sequence is normally regarded as a complete training example, the total error for the training sequence is computed by accumulating errors across all n time steps: $E = \sum_t E_t$. Finally, the weight w_{ij} of the RNN can be updated using SGD with the aim to minimizing the total error for the sequence: $\frac{\partial E}{\partial w_{ij}} = \sum_t \frac{\partial E_t}{\partial w_{ij}}$. Moreover, the standard BP algorithm is applied on the unfolded RNN sharing the weight matrices to calculate the gradients, therefore this algorithm is also referred to as backpropagation through time.

2.2.3 Bidirectional RNN

In many sequence classification tasks, the target at the current time step is not only dependent on the current and past feature frames, but also on the future frames. Therefore, it is beneficial to allow the RNN to access the future frames. As shown in figure 2.5, BRNN is an extension of unidirectional RNN, in which each hidden recurrent layer is composed of a forward layer and a backward layer, and these two layers are fully connected to the same output layer[SP97].

The fundamental idea behind the BRNN is that, each training sequence is processed separately by the forward layer and the backward layer in opposite directions. The forward layer processes the training sequence in forward order, while the backward layer processes the training sequence in reverse order. Then, the output of the BRNN at time step t is calculated based on the activations of both the forward layer and the backward layer at time step t. Therefore, at each time step the BRNN has access to the full left/right context information of the current frame.

2.2.4 Long Short-Term Memory

In section 2.2.2, we introduce the BPTT algorithm, which can be utilized to train RNNs. However, the problem of vanishing gradient may occur, when we train an RNN which is unfolded through long time steps[Hoc91][HBF$^+$01]. The LSTM was proposed to deal with the problems of vanishing gradient and learning long-term dependencies in time series data[HS97]. The LSTM distinguishes itself from the conventional RNN by introducing a new type of neurons. Neurons in the LSTM utilize three gates to control the information flow between each other, namely the input gate, the forget gate and the output gate. In the practice, LSTM neurons are normally implemented in cells, and an LSTM cell can contain a number of LSTM neurons.

Given the weight matrices of the LSTM cell W, U and b, the cell gates and cell states at time step t can be updated according to the following steps[HS97]:

- Forget gate vector: $f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f)$. Forget gates control the influence of the previous cell states on current states.
- Input gate vector: $i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)$. Input gates control the influence of the current information flow on current states.
- Output gate vector: $o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o)$. Output gates control the influence of the current states on the activations of the LSTM cell.
- Cell state vector: $c_t = f_t \circ c_{t-1} + i_t \circ tanh(W_c x_t + U_c h_{t-1} + b_c)$. The operator $\circ$ represent the element-wise product.
- Output vector: $h_t = o_t \circ tanh(c_t)$.

By taking advantage of three types of gates the LSTM can deal better with the problem of vanishing gradient, which is suffered by the conventional RNN. Like the conventional RNN, the LSTM can also be extended to bidirectional LSTM (BLSTM) in the same way to allow the LSTM to access the future context[GS05]. Moreover, LSTMs can also be trained using the BPTT algorithm[GS05].

3. ASR Background

Automatic speech recognition (ASR) is a technique that allows the computer to translate the speech signal into corresponding text through a series of algorithms. A typical ASR system is composed of several modules, such as the preprocessing module, the language model, the acoustic model, the lexicon, the decoder and so on. Since this work focuses on the acoustic model, this chapter briefly provides the basic knowledge on the acoustic model. The acoustic model can be divided into two categories, which are the context-independent acoustic model (CI AM) and the context-dependent acoustic model (CD AM). In the next section we first introduce the context-independent acoustic model including a concrete approach, namely the GMM-HMM acoustic model. Finally, the context-dependent acoustic model, as well as the context dependent DNN-HMM hybrid system, is discussed in the last section.

3.1 Context Independent Acoustic Model

The acoustic model is employed by an ASR system to estimate the conditional probability $P(A|W)$ that an acoustic feature vector sequence A is generated given that the word sequence W is actually uttered. Acoustic features are vectors of coefficients extracted from frames of speech signal which contain the most relevant information of speech in the frequency domain. Various types of acoustic features have been proposed, for instance MFCC, log-MEL (lMEL), PLP, etc. Normally the basic modeling units for context-independent acoustic models are a set of phonemes, from which the pronunciations of all the words in a vocabulary can be constructed. Then, acoustic models of words or sentences are built based on acoustic models of phonemes.

3.1.1 GMM-HMM Acoustic Model

The GMM-HMM AM is a very successful acoustic model which combines the hidden Markov model (HMM) and the Gaussian mixture model (GMM). At the present day, it is still used by state-of-the-art ASR systems to generate initial alignments for the training corpus. After that, context dependent AMs, e.g. CD DNN-HMM hybrid models, can be trained based on initial alignments generated by CI GMM-HMM models.

The HMM is an important probability model for sequential data processing. An HMM can be formally defined by the following four components:

- The set of HMM states: $S = (s_1, s_2, ..., s_N)$.

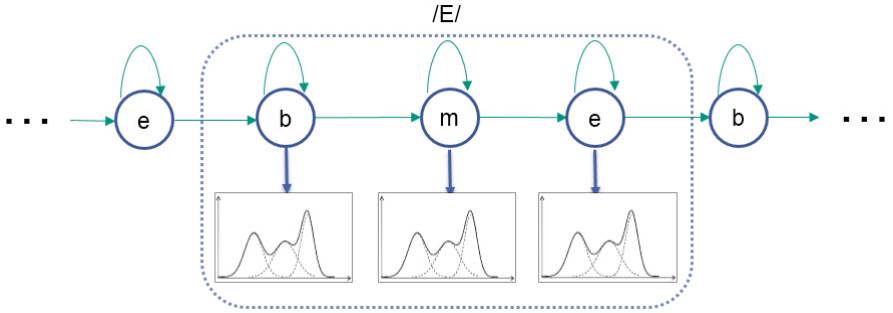

Figure 3.1: *GMM-HMM model for the phoneme /E/, the HMM has three states, i.e. b-state, m-state and e-state, and the observation distribution of each HMM state is modeled by a GMM.* (Figure originated from [Zhu15])

- The initial state probability distribution: $\pi = (\pi_1, \pi_2, ..., \pi_N)$, where $\pi_i = P(q_0 = s_i)$, and q_t denotes the HMM state at time step t.
- The state transition probability matrix: $A = (a_{ij})$, where $a_{ij} = P(q_t = s_j | q_{t-1} = s_i)$.
- The observation probability distributions: $B = \{b_j(x_t)\}$, where $b_j(x_t) = P(x_t | q_t = s_j)$, and x_t is the observation vector at time t.

The GMM is a continuous probability distribution represented by the weighted sum of a set of single Gaussian components:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x | \mu_k, \Sigma_k),$$

where μ_k denotes the mean vector of the k-th Gaussian component, Σ_k is the covariance matrices of the k-th Gaussian component, and π_k is the weight of Gaussian k. Mixture weights should subject to the normalization and positivity constraints, namely $\sum_{k=1}^{K} \pi_k = 1$ and $0 \leq \pi_m \leq 1$ for $1 \leq m \leq K$.

The GMM-HMM system takes advantage of HMMs to deal with the problem of varying speech rate[RJ86], and employs GMMs to model the observation distributions of HMM states. As discussed, phonemes are the basic modeling units for acoustic modeling, and each phoneme is modeled by a left-to-right HMM with three states, namely b-state, m-state and e-state. Each HMM state models a stage of the uttering of a phoneme. Figure 3.1 shows an example of GMM-HMM model for the phoneme "E". With the help of alignments of the training data, in which each frame of acoustic feature vector is aligned to a phoneme HMM state, we can collect a large amount of feature vectors for each phoneme HMM state. After that, the parameters of GMM are trained for each phoneme HMM state to fit the distribution of feature vectors, this training algorithm is also known as Viterbi training [RT03]. In order to bootstrap the training of the CI GMM-HMM system, the training corpus is usually equally aligned.

3.2 Context Dependent Acoustic Model

Context independent acoustic modeling discussed in the previous section builds a model for each phoneme regardless of its right/left contexts. However, a phoneme can have completely different pronunciations, and therefore different feature vector distributions, due to the influence of its neighboring phonemes, this effect is the so-called coarticulation[HH06]. Coarticulation is caused by the fact that people's vocal organs change continuously when

they speak. Hence, context dependent acoustic modeling was proposed to handle the problem of coarticulation, and the main idea behind the CD AM is to model phonemes with different right/left contexts separately.

A phoneme with its left and right contexts is also referred to as a polyphone. There are different types of polyphones according to the length of the contexts which are taken into consideration. For instance, triphone models take one phoneme in the left and right context into consideration respectively, while for quinphones two phonemes in the left and right context are considered. As an illustration of the polyphone model, we consider the phoneme sequence "ABEAA". The phoneme "E" in the center of the sequence can be either modeled by the triphone E(B|A), or by the quinphone E(A,B|A,A).

In our German context independent acoustic models there are 42 phonemes, and therefore $42^5 = 130691232$ possible quinphones. However, such a large number of quinphones can result in a lack of training data for each quinphone model and the number of quinphone models being intractable. The state tying approach, which is based on the idea of parameter sharing, is commonly used to deal with this problem. According to state tying, e.g. decision tree based state tying [YOW94], similar polyphone HMM states are clustered and tied to an identical observation probability distribution, and the clustered polyphone HMM states are also known as senones. A variety of distance measures can be used to calculate the similarity between two clusters of polyphone states, such as the entropy distance and the Euclidean distance. The CI GMM-HMM system described in section 3.1.1 can be extended to CD AMs in a simple matter by using senones as modeling units instead of phonemes[HH93].

3.2.1 Context Dependent DNN-HMM Hybrid System

In the GMM-HMM system, the observation distributions of HMM states are estimated using GMMs. However, the GMM as a generative model is normally surpassed by discriminative models, e.g. neural networks, in classification tasks. Therefore, some studies have been done on exploiting neural networks in the field of speech recognition. For instance, the ANN-HMM hybrid system employs the artificial neural networks (ANN) to estimate the joint posterior probabilities of context-independent phonemes and clustered context classes given acoustic feature vectors[BMWR92]. As next we present another neural network-based acoustic model, which was proposed in 2012 by Dahl et. al. in their work[DYDA12], namely the context dependent DNN-HMM hybrid system (CD-DNN-HMM).

The CD-DNN-HMM system differs from the traditional ANN-HMM hybrid system mainly in two aspects. Firstly, the CD-DNN-HMM system takes advantage of the unsupervised deep belief network (DBN) pre-training algorithm to pre-train deep neural networks before fine-tuning networks with the supervised BP algorithm. Secondly, the CD-DNN-HMM system utilizes DNNs to directly estimate the posterior probabilities of senones given acoustic feature vectors, namely $P(q_i|x)$, where x is a feature vector and q_i is a senone. After that, the observation probabilities of senones can be obtained with the help of Bayes' rule:

$$p\left(x|q_i\right) = \frac{P\left(q_i|x\right) p\left(x\right)}{P\left(q_i\right)},$$

where $P\left(q_i\right)$ is the priori probability of senone q_i estimated from the training data, and $p(x)$ is the priori probability of feature vector x which is commonly set to a constant value in practice. In this way, DNNs are able to replace GMMs to model the observation probability distributions of senones.

Similar to the CD-DNN-HMM system, in this work, our KWS system also employs neural networks to estimate the posterior probabilities of senones. In addition to the DNN, the senone classification performance of various types of recurrent neural networks, e.g. LSTM, BLSTM and TDNN-LSTM, is also explored.

4. NN-based Small-Footprint Flexible KWS

This chapter first briefly describes some conventional approaches to KWS. Then the problem this work attempts to solve, namely real-time KWS on mobile devices, is analyzed. After that, one recent approach proposed to solve this problem, and the disadvantages of this approach are discussed. Finally, our novel NN-based small-footprint flexible KWS is introduced.

4.1 Previous Work

The aim of KWS is to detect specific keywords/key-phrases in an audio stream, and there are several conventional KWS systems. For example, dynamic time warping (DTW) based KWS searches for matches between a sliding window of speech signal and the template of the keyword based on the DTW algorithm[SC78]. Large-vocabulary continuous speech recognition (LVCSR) based KWS first utilizes a LVCSR system to transcribe the input audio stream into text or rich lattices, then employs efficient searching algorithms to determine the position of the keyword in the text or lattices[GAV00][MKK+07]. Another widely used KWS approach is the Keyword/Filler HMM-based KWS; for each predefined keyword, this approach trains an HMM on a training dataset, and a filler HMM is trained for all non-keywords speech, noise, and silence[RRRG89][RP90][WMM91]. Both LVCSR based KWS and Keyword-Filler HMM-based KWS utilize Viterbi search for decoding.

Nowadays, many applications on mobile devices require real-time KWS. Such KWS systems should have a small memory footprint, low latency, and low computational cost without suffering loss of accuracy. However, the conventional KWS approaches do not meet these requirements, because a great deal of memory and computation are needed for Viterbi search.

4.1.1 DNN-based small-footprint KWS

In 2014, a small-footprint KWS system called Deep KWS was proposed by Chen et.al to solve the problem of real-time KWS on mobile devices[CPH14]. This system is composed of three modules, which are the feature extraction module, the deep neural network module, and the posterior handling module.

The feature extraction module produces 40-dimensional acoustic feature vectors from an audio stream using log-filterbank energies. In order to provide sufficient context information for the current frame, 30 past frames and 10 future frames are stacked with the

current frame to form a larger feature vector. The deep neural network is used to estimate the posterior probabilities of the entire keywords given the stacked input feature vectors. Keywords could be items such as “okay”, “google”, and so on. Finally, the posterior handling module calculates the confidence scores of keywords or key-phrases based on the posteriors produced by the DNN. However, raw DNN posteriors are normally noisy, so it is necessary to smooth the posteriors. The smoothed posteriors are thus taken to be the mean of posteriors over a time window of 30 frames. Then the confidence at the j-th time step is calculated based on smoothed posteriors using this formula:

$$confidence = \sqrt[n-1]{\prod_{i=0}^{n-1} \max_{h_{max} \leq k \leq j} p_{ik}},$$

where p_{ik} represents the smoothed posterior probability of word i given the features at time step k, $h_{max} = max\{1, j - w_{max} + 1\}$ and w_{max} is the sliding window size which is set to 100. Hence, the confidence score is therefore the geometric mean of the maximum posteriors of all keywords to be detected in the past 100 frames.

However, the Deep KWS system also suffers some disadvantages. Firstly, a large amount of training data is needed to train the DNN, according to the paper[CPH14], more than two thousand training examples for each keyword are used to train the DNN. However for keywords which rarely appear in the natural language, it is difficult to collect enough training data. Secondly, the output layer of the DNN is fixed, if new keywords are needed to be detected, then a new DNN should be trained. Therefore, the Deep KWS system is not flexible. Thirdly, Deep KWS system uses the DNN to estimate the posterior probabilities from a sequence of feature vectors. Graves et al. state that the BLSTM and LSTM outperform the DNN in the task of framewise phoneme classification in their work[GS05]. Hence, the LSTM is probably able to improve the KWS accuracy over DNN-based systems.

4.2 NN-based Small-Footprint Flexible KWS

The previous section reviewed various previous approaches to KWS, and this section introduces this work’s novel approach to KWS. This approach aims to build a flexible KWS system with a small footprint and high accuracy. The small footprint means that the KWS system does not occupy much RAM memory, which is vital for a system designed for mobile devices. Real-time capacity is also an important factor that needs to be taken into consideration. This means the KWS system should perform computations under time constraints in order to deliver KWS results continuously in real-time. Unlike the inflexible Deep KWS system that utilizes the DNN to predict the posteriors for entire keywords, a flexible KWS system is also desirable. This system should therefore be able to detect new keywords without collecting thousands of training examples for the specific keywords or retraining the neural network.

In order to achieve these goals, we propose a KWS system, which is composed of three components. They are the feature extraction module, the posterior probability estimation module and the posterior handling module. The following subsections detail these three components of our KWS system.

4.2.1 Feature Extraction

As with the Deep KWS system, the feature extraction module is used to generate acoustic features, i.e. 54-dimensional log-MEL features (lMEL), from the original speech signal. As this system does not estimate the posteriors for entire words, the context window used

is shorter than the context window used by the Deep KWS approach. Longer context windows normally lead to better performance at the expense of computational cost and latency; thus, as a compromise between latency and performance, 19 past frames and 7 future frames are used to provide context information for the current frame.

4.2.2 Posterior Probability Estimation

As introduced in section 3.1, the pronunciation of a word is composed of a sequence of phonemes. In context dependent models, these phonemes are further modeled with polyphones such as triphones or quinphones, taking their context information into consideration. In practice, different polyphone HMM states are clustered and tied to an identical distribution; these clustered polyphones states are also known as senones. Instead of using DNN to predict posteriors for entire keywords, the new KWS system therefore uses various NNs to predict posterior probabilities for subword units, which can be phonemes or senones.

As another improvement to the Deep KWS, which only uses FFNN for posterior probability estimation, our KWS system also employs RNNs such as LSTM, GRU, BLSTM, and TDNN-LSTM. The RNN is designed to solve the problems of learning time series data and modeling sequences of information. As human speech signal is time series signal in nature, it is therefore reasonable to use RNNs to model speech signal. In this work, the many-to-one RNN is used to estimate the posterior probabilities of phonemes/senones.

Quinphone-based context dependent acoustic models are employed, where the quinphone is dependent on two contiguous phonemes in its left and right context, respectively; thus, it is helpful to allow the LSTM to access the feature frames in the future context. There are two ways to provide the future context to the LSTM. One way is to train the LSTM with target delay, which introduces a delay between the target senones and LSTM outputs. The other way is to use an extension of unidirectional LSTM, i.e. bidirectional LSTM (BLSTM), in which each hidden recurrent layer is composed of a forward LSTM layer and a backward LSTM layer. At each time step, the BLSTM thus has access to the full right and left context information of the current frame in the input feature sequence.

TDNN-LSTM is inspired by the time delay neural network (TDNN) proposed by Waibel et al. [WHH+89]. As speech signals can have local dependencies across time and frequency domains, it is beneficial to perform a convolution operation on the input acoustic feature sequence before feeding acoustic features to an LSTM [SVSS15]. Our TDNN-LSTM employs TDNN to perform a convolution operation on acoustic features, then repacks the outputs of TDNN to pass to the LSTM.

4.2.3 Posterior Handling

Once the posterior probabilities are calculated for each frame of the feature vector sequence, the confidence scores of the predefined keyword/key-phrase can be computed based on the posterior probabilities derived from the neural networks. The posterior handling module of our KWS system employs a similar algorithm to the Deep KWS, which enables the simple and rapid calculation of confidence scores. As the original posterior probabilities generated by neural networks are normally noisy, before the confidence scores for the keyword/key-phrase are computed, the NN posteriors are also smoothed by calculating the mean of posteriors using the same formula as [CPH14]:

$$\bar{p}_{ij} = \frac{1}{j - h_{smooth} + 1} \sum_{k=h_{smooth}}^{j} p_{ik},$$

where p_{ik} and $\bar{p}_{ik}$ are the raw posterior and smoothed posterior of phoneme i given the features at time step k, $h_{smooth} = max\{1, j - w_{smooth} + 1\}$, and w_{smooth} is the size of smooth window which is set to 30. As the pronunciation of a word consists of a phoneme sequence, our KWS approach is based on the idea that, if the phonemes of a given keyword/key-phrase appear in a sliding window of the frame sequence with high probabilities, then this keyword/key-phrase also appears in the sliding window with high probability. Therefore, the confidence score of the keyword/key-phrase at the j-th time step is calculated using the following formula:

$$confidence = \sqrt[n-1]{\prod_{i=0}^{n-1} \max_{h_{max} \leq k \leq j} \bar{p}_{ik}},$$

where $\bar{p}_{ik}$ denotes the smoothed posterior probability of phoneme i given the features at time step k, $h_{max} = max\{1, j - w_{max} + 1\}$ and the sliding window size w_{max} is also set to 100. The confidence score is, therefore, the geometric mean of the maximum posteriors for all the phonemes, which compose the keyword/key-phrase, in the past $w_{max} = 100$ frames.

The highest computational cost of our KWS system results from the evaluation of neural networks and the calculation of confidence scores. Thus, compared with the conventional LVCSR or HMM-based KWS approaches, it requires less computation and has low latency due to the omission of the Viterbi search. Furthermore, unlike the Deep KWS's use of DNN to predict the posteriors for entire words, our KWS approach is flexible, as it is based on the posteriors of phonemes/senones and thus independent of specific keywords. This means that it does not require the collection of thousands of training examples for each keyword, or retraining of the NN in order to detect new keywords. Our KWS approach only needs less than one hundred examples for each keyword/key-phrase to calculate an appropriate range of the confidence threshold for the keyword/key-phrase. As another improvement to the Deep KWS approach, various NNs are explored in this work, with the aim of achieving higher accuracy.

5. Implementation

This chapter demonstrates the implementation details of our NN-based KWS system in order to enable other researchers to reimplement this system. First of all, a feature extraction module is necessary to realize our system, this module is used to extract acoustic feature vectors from the speech signal. Then, in the NN training phase different types of neural networks are trained based on the extracted features. Finally, in the NN-based KWS phase confidence scores of predefined keywords are calculated continuously from an audio stream with the help of the trained NNs and the feature extraction module. Figure 5.1 presents the implementation pipeline of our KWS system, and each implementation step in the pipeline is detailed in the following sections.

5.1 Feature Extraction Module

The feature extraction module is needed for both the NN training phase and the subsequent NN-based KWS phase. In the NN training phase, the extracted acoustic features are used to train the NNs, while in the NN-based KWS phase, the trained NNs generate posteriors from the acoustic feature vectors. Initially, we have a database which contains only speech audio WAV files sampled at 16 kHz. The feature extraction module computes 54-dimensional lMEL features from the audio signal over a sliding window with the size of 32 ms and a frame shift of 10 ms. In comparison to the Deep KWS system, which uses 30 past frames and 10 future frames to provide context information for the current frame, a much shorter context window is used. As this KWS system employs NNs to calculate posteriors for phonemes or senones rather than whole words, 19 past frames and 7 future frames are estimated to provide sufficient context information for the prediction of the phoneme or senone label of the current frame. By using a shorter context window, computation is reduced and lower latency achieved.

In this work, the feature extraction module is implemented using the Janus Recognition Toolkit[FGH$^+$97] (JRTk). JRTk is a framework for general-purpose speech recognition that was cooperatively developed by the Karlsruhe Institute of Technology and Carnegie Mellon University. JRTk consists of a range of various functional modules that include acoustic pre-processing, HMM based acoustic modeling with GMM-HMM, hybrid DNN-HMM, and IBIS one-pass decoder for decoding[SMFW01]. The functional modules of JRTk are implemented in highly optimized C code to provide maximum efficiency of program execution. In order to offer flexible and high-level manipulation of functional modules, JRTk also supports an object-oriented programming interface in Tool Command Language (Tcl).

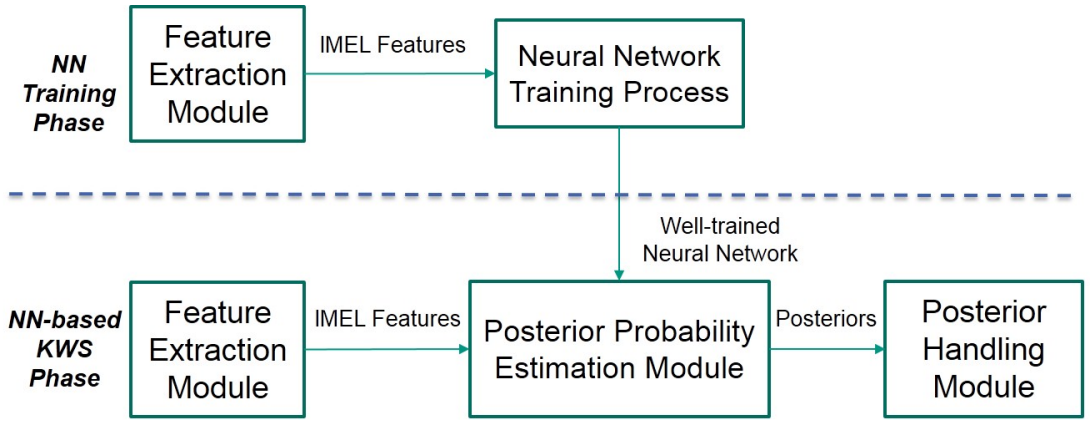

Figure 5.1: *Implementation pipeline of our KWS system. First, in the NN training phase, different neural networks are trained using the lMEL features extracted by the feature extraction module from the labeled training dataset. Then, in the NN-based KWS phase, the well-trained neural network generates the posteriors using lMEL features generated by the feature extraction module, and the posterior handling module calculates the confidence scores based on the NN posteriors.*

5.2 Neural Network Training Phase

In this work, all neural networks are trained using the deep learning frameworks Lasagne and Pytorch. Lasagne is a lightweight framework based on Theano[BLP+12][BBB+10], which allows users to construct and train neural networks with great ease by encapsulating the functionality of Theano in a higher level. In January 2017, however, a novel framework, Pytorch, was released by Facebook AI Research. Pytorch is also a fully-featured deep learning framework, but it offers particularly strong GPU acceleration. Both frameworks support a flexible Python interface and GPU computation based on Nvidia's CUDA library.

The following subsections first discuss the labeling and the training data for NNs, and then describe the architectures and the training details of the FFNN, the LSTM, the BLSTM, and the TDNN-LSTM respectively, as we employ these different types of neural networks for the KWS task. We use the FFNN as the baseline, and compare the performance of the LSTM, BLSTM and TDNN-LSTM models with the FFNN.

5.2.1 Labeling and Training data

Before we start to train the NNs, the training data should be labeled, which means each acoustic feature frame is assigned a target label. The target label set is dependent on the acoustic model employed. In order investigate the impact of different AMs on the KWS performance, the 42 phonemes CI AM, the 6k senones CD AM and the 18k senones CD AM are utilized in this work. Like the feature extraction module, the labeling is also implemented using the Janus Recognition Toolkit, and performed with the help of a well-trained German ASR system. After the labeling of the training data, the acoustic feature vectors are extracted from the audio WAV files, and then stored in the PFile format. Table 5.1 shows some example rows of the PFile format.

A full training dataset is normally split into two non-overlapping subsets. One subset is used for the actual neural network training, and the weights of the network are fitted on this subset of data through BP training. The other data subset is used for validation and helps to avoid the problem of overfitting. During the training procedure, after each epoch, the frame error rate of the network is measured on the validation subset, and based on

Utterance Index	Frame Index	Feature Vector	Class Label
0	0	[0.5, 0.6, ..., 0.9]	120
0	1	[0.3, 0.2, ..., 0.4]	34
0	2	[0.2, 0.9, ..., 0.5]	733
1	0	[0.5, 0.3, ..., 0.1]	703

Table 5.1: *PFile* format, feature vectors are arranged in rows.

the resulting validation error, the learning rate is adjusted until the stopping point of BP training can be determined.

Lasagne and Pytorch operate on input data in the form of Tensor data types. Thus, it is necessary to convert the training data stored in PFile format on the hard disk into Tensor data types. In our experiments, training data is partitioned into 1000MB segments, which are loaded from PFile into main memory successively. Then the training instances of the data partition are shuffled, and each partition is iterated in mini-batches of a fixed size. Each batch of training data, which is composed of sequences of feature vectors and corresponding labels, is converted to Tensor variables, and passed to the neural network training process for use by the BP algorithm.

5.2.2 FFNN

5.2.2.1 Architecture

The standard FFNN with fully connected layers introduced in section 2.1 is trained as the baseline. The input feature vector of the FFNN is 54-dimensional lMEL features with 19 frames in the left context and 7 frames in the right context, which are 27 frames in total. Therefore, the input layer of the FFNN has $54 * 27 = 1458$ neurons. The number of hidden layers and the number of neurons in each hidden can be optimized in the experiments. Normally, a larger FFNN offers more powerful modeling capability, however, the larger FFNN may suffer the problem of overfitting and lead to higher latency and a larger memory footprint in the subsequent KWS phase. For the activation function of hidden layers, we can apply ReLU, sigmoid or tanh functions. The size of output layer corresponds to the number of phonemes or senones of the AMs. Moreover, the output layer uses the softmax activation function and generates the estimated posterior probability distribution over the target phoneme or senone sets.

5.2.2.2 Training

Before starting the neural network training with the BP algorithm, there are multiple ways to initialize the weights of NNs. For example, we can initialize the weights randomly with values from a uniform distribution or a Gaussian distribution. In this work, the initial weights of all NNs are sampled from uniform distributions with zero mean. After initialization, weights of the FFNN are optimized using a variant of the BP algorithm, namely mini-batch gradient descent (MGD), over a mini-batch with the fixed size of 256. The aim is to minimize the cross entropy loss of the FFNN over the mini-batch.

As a momentum term can help to accelerate the training process of neural networks and avoid sinking into local minima[PNH86], our implementation employs a variant of momentum, i.e. Nesterov momentum[Nes83]. We set the initial learning rate of the gradient descent method to 0.01 and Nesterov momentum to 0.9. Furthermore, Newbob strategy is employed for learning rate scheduling. After each epoch of training, we measure the frame error rate (FER) of the FFNN on the validation dataset, and compare the current FER with the FER of the previous epoch. Once the decrease of the FER between two epochs

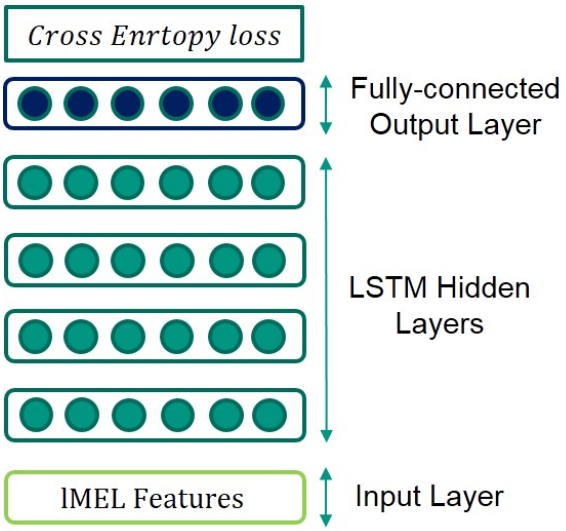

Figure 5.2: *Architecture of the LSTM implemented in this work, the LSTM hidden layers can also be replaced by GRU or BLSTM hidden layers.*

drops below 0.005, the current learning rate starts to decay exponentially with a factor of 0.5. When the decrease of validation error is less than 0.0001, we terminate the network training process.

5.2.3 LSTM and BLSTM

5.2.3.1 Architecture

In this work, the many-to-one LSTM is employed for sequence classification. This type of LSTM accepts a sequence of input feature vectors and generates a final output prediction for the input sequence. Figure 5.2 shows the architecture of the LSTM we implemented in this work. In addition to the LSTM, the sequence classification performance of the gated recurrent unit (GRU), as a variant of the LSTM, is also investigated. The GRU is implemented in the exactly the same way as the LSTM; however, for reasons of space, only the implementation details of LSTM are discussed in the following section.

At each time step, the LSTM takes a 54-dimensional lMEL feature vector as input, which means the size of the input layer of the LSTM is also 54. The different modeling capacities of LSTM are explored by varying the depth and the size of recurrent hidden layers. For the activation function of recurrent hidden neurons, the tanh function is utilized. A fully connected layer with softmax function is also added on top of the LSTM layers; the size of this output layer corresponds to the number of phonemes or senones.

The whole feature sequence LSTM can access includes the current feature frame, 19 past frames, and 7 future frames. Thus, the sequence length of input feature vectors in 27, which is the same as the length of the input feature vector for the FFNN discussed in the previous section. Given a sequence of input feature vectors with length 27, LSTM generates an output vector after accepting and forward propagating each input feature vector; thus, the length of the output sequence is also 27. The LSTM is trained with a target delay of 7 frames by taking the outputs of LSTM at the last time step, i.e. when the LSTM observes the 27th input feature vector, as the estimated posteriors for the input feature sequence.

The ability of the BLSTM to do sequence classification is also exploited in this work. Each recurrent layer of BLSTM consists of a forward LSTM layer and a backward LSTM layer;

Newbob Decay	Newbob Threshold	Batch Size	Learning Rate	Nesterov Momentum	Loss Function
0.5	[0.005, 0.0001]	256	0.01	0.9	Cross Entropy

Table 5.2: MGD training hyperparameters of LSTMs, BLSTMs and GRUs.

the two layers process the input sequence and are trained independently. The forward and backward hidden layers are concatenated to form a larger layer, and this is then connected to the next hidden BLSTM layer or the output layer. The size of the output layer also depends on the number of modeling units of AMs. For BLSTM, the feature sequence also includes the current feature frame, 19 past frames, and 7 future frames. This work takes the BLSTM output at the 20th time step, which is a combination of the forward hidden layer output at the 20th frame and the backward hidden layer output at the 7th frame in the reverse direction, as estimated posteriors for the input feature sequence. This also corresponds to the left and right context information used.

5.2.3.2 Training

For simplicity, LSTM, BLSTM and, GRU are collectively called LSTM in this section. As with FFNNs, the weights of LSTMs are also initialized with random values from a uniform distribution with zero mean. All initial hidden states and cell states of LSTMs are set to the constant value zero for each training instance in the batch. The input training data of the LSTM is organized in a 3-dimensional tensor with the shape (batch size, sequence length, input dimensionality), where the input dimensionality is 54, based on the dimension of the lMEL features; the sequence length is 27; and the batch size is 256. The weights of LSTM are updated with each batch.

The standard backpropagation through time (BPTT) algorithm is applied to train LSTMs to predict the phoneme or senone labels of the feature sequence. The input feature vectors are processed in sequential order, and the cross-entropy loss for the training sequence is computed based on the target and the output of the LSTM at the last time step. The weights of the LSTM are optimized with the help of MGD over mini-batch of size 256. The hyperparameters of MGD training are listed in table 5.2, and these are almost the same as the FFNN, which allows a fair comparison among different types of NNs such as LSTM, FFNN and TDNN-LSTM.

5.2.4 TDNN-LSTM

5.2.4.1 Architecture

The TDNN-LSTM architecture adopted in the work is modeled after the Deep Speech 2 architecture proposed in [AAA+16]. The TDNN-LSTM architecture is shown in figure 5.3. A TDNN with one or more layers forms the bottom of a TDNN-LSTM. The TDNN accepts a 3D feature map as input with the shape (channels, height, width), where the parameter channels refers to the number of input feature channels. In this case, only one channel feature map, the lMEL features, is used as TDNN input; thus, the parameter channels is always set to one in our implementation. The other two parameters, height and width, represent the height and width dimensions of the input feature map, which are the dimension of lMEL feature vector 54 and the length of feature vector sequence 27, respectively.

The architecture of TDNN has several hyperparameters, including the number of TDNN hidden layers, the size of convolution kernels, and the number of output channels of each convolution layer. A smaller subset of the entire training dataset is used to determine

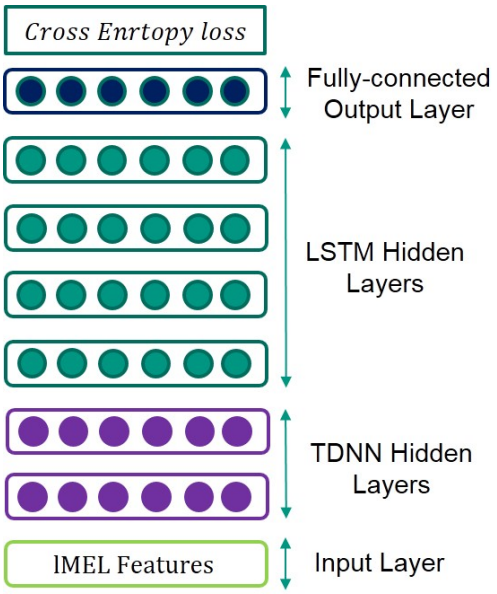

Figure 5.3: *Architecture of the TDNN-LSTM implemented in this work, the TDNN layers form the bottom of the TDNN-LSTM, hyperparameters of TDNN layers, such as the depth of TDNN hidden layers, the size of convolution kernels, and the number of output channels of each convolution layer, are optimized using a subset of the training data.*

the optimal values of these hyperparameters. As an extension to the conventional TDNN, which performs a convolution operation in the time domain with a stride of 1, the TDNN utilized in this work performs the convolution operation across both the time domain and the frequency domain with a stride of 2. This can decrease the required computational effort and the number of time steps in the ensuing LSTM layers compared with using a stride of 1. Furthermore, each TDNN layer is followed by a batch normalization layer[IS15] and the hardtanh activation function with a minimum value 0 and a maximum value 20.

The output feature maps of the last TDNN layer are repacked and passed to the LSTM layers. Each output channel of the TDNN layer generates an output feature map. The features on the same column of all output feature maps are stacked to form a larger feature vector, and the stacked feature vector is used as the input of the LSTM for one time step. The size of the LSTM input layer is calculated using the formula:

$$input\_size = [(W - F + 2P)/S + 1] * C,$$

where W and F denote the input feature map size and the filter size, P is the zero padding amount, S is the stride size and C represents the number of output channels. GRU can also be built upon TDNN in a similar manner to LSTM. Finally, the outputs of the LSTM are fed to the fully connected output layer with the softmax function.

5.2.4.2 Training

The training data for TDNN-LSTM is organized in a 4-dimensional tensor variable with the shape (batch size, channels, input dimensionality, sequence length), where the value of channels is set to the constant value one; the values of batch size, and input dimensionality, sequence length are set to 256, 54 and 27, respectively. These are the same as in the LSTM training discussed in the previous section.

Newbob Decay	Newbob Threshold	Batch Size	Learning Rate	Nesterov Momentum	Loss Function
0.5	[0.005, 0.0001]	256	0.01	0.9	Cross Entropy

Table 5.3: MGD training hyperparameters of TDNN-LSTMs.

In this work, the weights of TDNN layers and LSTM layers are jointly trained with the BPTT algorithm. The lMEL feature maps are fed to the input layer of the TDNN-LSTM, and then propagated through the TDNN layers. After that, the output feature maps of the last TDNN layer are repacked and passed to the LSTM layers. Finally, the cross-entropy loss is computed based on the target and the output of the TDNN-LSTM at the last frame. As with FFNN and LSTM, TDNN-LSTM is optimized using MGD with a mini-batch size of 256. Table 5.3 lists the MGD training hyperparameters of TDNN-LSTM.

5.3 NN-based KWS Phase

The previous section describes the architecture and the training procedure of different neural networks, in this section we detail our novel KWS approach using the trained NNs. We use the key-phrase "okay cosa" as a concrete example to illustrate our KWS approach, and show the steps to implement the KWS system detecting the key-phrase "okay cosa". We also use this key-phrase to evaluate the final KWS system in the subsequent experiments. As introduced, our KWS framework consists of three modules, which are the feature extraction module, the posterior probability estimation module and the posterior handling module. We use the same feature extraction module for both the NN training phase and the NN-based KWS phase, therefore this section only discusses the posterior probability estimation module and the posterior handling module.

5.3.1 Posterior Probability Estimation

For each frame of feature vector, the same context window is used as in the NN training phase; this consists of 19 frames in the left context and 7 frames in the right context. For the FFNN, the feature vector sequence is concatenated to form a larger feature vector with dimension $54 * 27 = 1458$, which is placed in the input layer of the FFNN. Activations of the network are propagated from the input layer, through all hidden layers, to the output layer, and the outputs of the final output layer with softmax function are the estimated posteriors.

For LSTM, an input feature sequence is fed to the network frame by frame; at each time step, the LSTM accepts a feature vector of dimension 54, propagates the signal in a feed-forward fashion, and stores the values of the hidden states and hidden cells. The output of the LSTM at the last time step is used as the estimated posteriors for the whole input feature sequence. For BLSTM, the output at the 20th time step is used as the estimated posteriors. The TDNN layer of the TDNN-LSTM takes a 3D feature map as input, before the outputs of the last TDNN layer are passed to the LSTM framewise; the outputs of the LSTM at the last frame are used as the estimations of posteriors.

The posterior probability estimation module is also implemented using the deep learning frameworks Lasagne and Pytorch. Furthermore, this module is independent of keywords/key-phrases to be detected, as NNs are used to estimate the posteriors for subword units like phonemes or senones, which makes the output layer of the NN independent of keyword/key-phrases. Therefore, the same NN can be used for posterior estimation regardless of the keywords/key-phrases.

	Pronunciation	Denoted by
Without Word boundary	"O K EH K OH Z AH"	O
	"OH K EH K OH Z AH"	OH
	"O U K E I K OH Z AH"	OU
With Word boundary	"O K EH" (WB) "K OH Z AH"	O_WB
	"OH K EH" (WB) "K OH Z AH"	OH_WB
	"O U K E I" (WB) "K OH Z AH"	OU_WB

Table 5.4: Six pronunciation variants of the key-phrase "okay cosa".

5.3.2 Posterior Handling

5.3.2.1 Pronunciation Variants

The pronunciation of the key-phrase "okay cosa" is composed of a sequence of phonemes. In the dictionary, the word "okay" has three pronunciations, which are "O K EH", "OH K EH", and "O U K E I". The word "cosa" has one pronunciation, namely "K OH Z AH". Hence, the three pronunciations of the word "okay" and one pronunciation of the word "cosa" result in three combinations of pronunciation for the phrase "okay cosa".

Furthermore, the word boundary can also be taken in into consideration when we investigate the pronunciations of key-phrases. For instance, the pronunciation of the phrase "okay cosa" can be modeled by a sequence of continuous phonemes according to the dictionary, e.g. "O K EH K OH Z AH". The fourth phoneme "K" in the phoneme sequence can be modeled by the quinphone K(K,EH|OH,Z). Due to the fact that the phrase "okay cosa" is composed by the two words "okay" and "cosa", it is also reasonable to take the word boundary between these two words into consideration. In this case, the pronunciation of the phrase is modeled by two separate phoneme sequences derived from the pronunciations of words "okay" and "cosa", e.g. "O K EH" (WB) "K OH Z AH". The fourth phoneme "K" can therefore be modeled by the quinphone HMM K<WB>(SIL<WB>|OH,Z), and "SIL" represents silence.

Thus, there are in total six possible pronunciations of the key-phrase "okay cosa", which are listed in the table 5.4. For simplicity, these six pronunciations are denoted by O, OH, OU, O_WB, OH_WB, and OU_WB in the following chapters.

5.3.2.2 Representation Variants of Phoneme Posteriors

As introduced in section 4.2.2, the confidence score of a keyword is calculated based on the posterior probabilities of all the phonemes which construct that keyword. There are several different ways to model the phoneme posteriors. In a context dependent AM, a phoneme can be represented by the b-state, m-state, or e-state of the quinphone HMM which models this phoneme depending on its left and right contexts.

For instance, the phoneme "EH" in the pronunciation "O K EH K OH Z AH" is modeled by the quinphone HMM EH(O,K|OH,Z) with three states, i.e. b-state, m-state, and e-state. Therefore, given the input feature vectors, the posterior of the b-state, m-state, or e-state of the quinphone HMM EH(O,K|OH,Z) can be regarded as the posterior of the phoneme "EH". In addition to the posterior of a single quinphone HMM state, the mean or maximum of the posteriors of the three HMM states can also represent the posterior of the phoneme "EH". Thus, the posterior of a phoneme can be represented by the posterior of one of its three corresponding quinphone states, or the mean or maximum of the posteriors of all three corresponding quinphone states.

Our KWS approach employs NNs to predict the posterior probabilities of senones, i.e. the clustered HMM states of quinphones, when a context dependent acoustic model is

employed. With the help of the phonetic cluster tree, the senone to which a quinphone HMM state is tied can be retrieved. Then, based on the mapping from the senone set to the target label set used for NN training, the corresponding entry index in the NN output vector for the quinphone state can be obtained.

5.3.2.3 Confidence Calculation

Given a sequence of posteriors generated by the NN, the posterior vectors are first smoothed by taking the mean of posteriors over a time window of 30 frames. Then, the confidence score of the keyword/key-phrase is calculated based on the smoothed posteriors using the formula as introduced in section 4.2.3:

$$confidence = \sqrt[n-1]{\prod_{i=0}^{n-1} \max_{h_{max} \leq k \leq j} \bar{p}_{ik}},$$

where the phoneme posterior $\bar{p}_{ik}$ can be the posterior of the b-state, m-state, or e-state of the corresponding quinphone HMM of phoneme i, or the mean or maximum of the posteriors of the three states.

It is important to investigate the pronunciation variants of the keyword/key-phrase to be detected. The first step is to find the best match pronunciation and then use this fixed pronunciation consistently for confidence calculation. Besides, for each sliding window it is also possible to calculate the confidence scores for all pronunciation variants, and choose the one with the maximum confidence as the final confidence for a given window.

6. Experiments

This chapter documents the experimental setup used to evaluate our KWS approach and also the evaluation results. We evaluate our KWS system by detecting the keyword "okay cosa" in audio WAV files. Evaluation results are arranged in two parts, the first part includes comprehensive experimental results on neural network training, and the second part presents the performance of our KWS approach using the trained neural networks. Furthermore, a website was developed to collect the evaluation data. The first section of this chapter introduces our data collection website, followed by a description of the experimental setup, and the last section presents the experimental results.

6.1 Data Collection

We developed a website for the collection of evaluation data, the layout of our website is shown in figure 6.1. When the website is opened at the first time, the browser immediately requests the access to the microphone of the computer. After the access to the microphone is granted, website users can press the REC button in the middle of the website to start the recording. However, there are normally about one second of delay between when the REC button is pressed and the recording actually starts, as the microphone and the recorder program may need time to initialize and run. Therefore, in order to avoid the problem that we receive recordings which are truncated at the beginning, we added a recording indicator, which is a circle, next to the REC button. The circle turns red after 1s of button pressing indicating that the microphone and the recorder program is initialized, and then the user can pronounce the phrase "okay cosa".

When the user releases the REC button, the recorder ends the recording. Then the recording is sent from the browser to our server immediately, and stored on the server in WAV format. We also implement the recording playback function and voice activity detection for the website, as problems such as wrong microphone or mixer settings can result in recordings containing only noise. After the recording, users can play their recordings back on the website, and decide whether to discard the recordings or not. At the bottom of the website there is also an example of "okay cosa" pronunciation, which can be played. Moreover, voice activity detection is implemented for this website, if a user presses the REC button without saying anything, a window will appear indicating that no voice has been detected.

We take advantage of HTML5 and the JavaScript library jQuery for front-end web development. The library jQuery significantly simplifies programming with JavaScript by

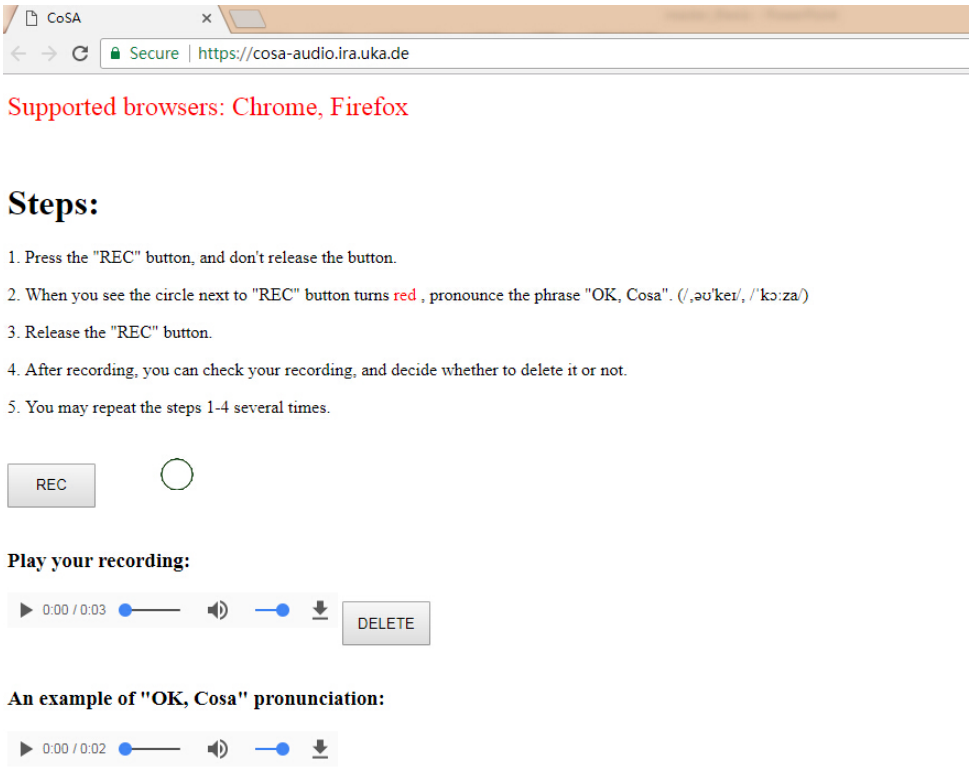

Figure 6.1: *Website for data collection.*

encapsulating common functionalities of JavaScript. The front-end process demonstrates our website to users and responds to users' requests such as starting the recording, ending the recording and so on. The Python based framework Django is utilized for back-end development. The recordings from the web browser are stored and processed on the back-end server. Besides, the back-end server also gives the front-end feedbacks when no voice is detected in the received recordings. For voice activity detection we use the WebRTC Voice Activity Detector developed by Google.

6.2 Experimental Setup

6.2.1 Training Dataset and Test Dataset

In this work, NNs are trained and tested on a German corpus derived from Quaero training data, along with audio data from broadcast news and the Baden-Wuerttemberg State Parliament[KHM+14]. The training dataset for NNs contains approximately 59M frames from 34k utterances, which total about 160 hours of audio. There are 3.5M frames (∼9.5 hours) in the validation dataset used for learning rate scheduling and determining the endpoint of NN training. The test dataset for measuring the frame error rate of trained NNs includes 2.7M frames (∼7.6 hours).

For the evaluation of our final KWS systems, we collected 348 positive examples containing the key-phrase "okay cosa" from various speakers with the help of our website. Each speaker made about ten recordings, so we have positive examples from more than thirty speakers. For negative examples, we selected 3011 utterances which do not contain the key-phrase "okay cosa" from the validation and test data used for NN training. Both positive and negative datasets contain some noisy utterances.

Truth \ Test	Positive	Negative
Positive	True Positive	False Negative
Negative	False Positive	True Negative

Table 6.1: Definitions of true positive, false positive, false negative and true negative.

6.2.2 Metrics

The performance of a binary classifier is normally measured using the receiver operating characteristic curve (ROC curve). The ROC curves of different classifiers are usually plotted in the same graph to provide a fair comparison between those classifiers. In this work, the ROC curve is also used as the metric to measure the performance of KWS systems.

Consider a binary classification problem, where class labels of the instances and outputs of the classifiers are either positive (p) or negative (n). At this point, the predictions of the classifiers have totally four possible outcomes, which are true positive (TP), false positive (FP), true negative (TN), and false negative (FN) respectively. The definitions of the four outcomes are listed in table 6.1.

The ROC curve of a classifier is generated by plotting the true positive rates (TPR = $\frac{TP}{TP+FN}$) and false positive rates (FPR = $\frac{FP}{FP+TN}$) of the classifier at various threshold settings. Depending on the requirements, an appropriate classifier can be chosen that provides the best trade-off between TPR and FPR. On an ROC graph, the area under curve (AUC) is also a useful metric, as a larger AUC implies superior classification performance.

In addition to the ROC curve, there is another effective metric employed to evaluate the performance of the KWS system, namely the F1-score. The F1-score is defined as:

$$F1 = 2 * \frac{precision * recall}{precision + recall},$$

where precision = $\frac{TP}{TP+FP}$, and recall = $\frac{TP}{TP+FN}$. It can be interpreted as the harmonic mean of precision and recall. The harmonic mean is then multiplied by the constant two so as to scale the F1-score to one when the values of precision and recall are both one.

6.3 Experimental Results

The following section presents and explains the experimental results of NN training, followed by the evaluation results of the final KWS systems based on the trained NNs. As one of our goals is to build a real-time small-footprint KWS system on mobile devices, the size of the NN models and the time needed for confidence calculation are also critical experimental results, alongside the accuracy.

6.3.1 Neural Network Training

The evaluation results of NN training for 6k senones CD AM are given first. These were gained from extensive experiments on NNs for 6k senones to explore the impact of network architectures on classification performance. After that, the final subsection presents the evaluation results of NN training for 42 phonemes CI and 18k senones CD AMs.

6.3.1.1 Initial Experiments

The training of LSTMs can take days or even weeks using the full training dataset; thus a subset containing 11M frames (∼30h) from the original training dataset was selected to create a smaller training dataset. Initial experiments were conducted on the smaller training dataset to help select a suitable deep learning framework and to ascertain a reasonable range of NN hyperparameters within a short time.

Different deep learning frameworks have different advantages and disadvantages respectively. In the initial experiments, the frameworks Pytorch and Lasagne were used to train the models. NNs trained using Pytorch and Lasagne achieved similar accuracies on the validation dataset; however, Pytorch was found to be much faster than Lasagne in terms of training LSTM. Therefore, in the subsequent experiments on training NN using all training data, only Pytorch was used to implement the models.

In the initial experiments we also investigated the optimal ranges of NN hyperparameters such as the learning rate, the momentum, the length of target delay and so on. Experimental results show that in many cases MGD with the learning rate 0.01 and the Nesterov momentum 0.9 can deliver the best accuracy compared with other learning rate scheduling strategies. As discussed in section 4.2.2, target delay provides LSTM with future feature frames, intuitively target delay of more frames can lead to better accuracy, and our experimental results also matches this hypothesis. We implemented LSTMs without target delay and with target delay of 5, 6, 7 frames, results show that target delay of more frames leads to higher accuracy. Since more frames in the future context can result in higher latency of our KWS system (one future frame increases 10ms of latency), we did not continue to do experiments on more future frames, and trained LSTM with target delay of 7 frames in all subsequent experiments.

There are some hyperparameters related to TDNN layers that are used exclusively for TDNN-LSTM training; these include the number of TDNN layers, the kernel size, and the number of channels in each TDNN layer. In the initial experiments, different designs of TDNN layer were thus investigated. For instance, TDNN-LSTMs with one hidden layer and two hidden layers were investigated. In TDNN-LSTMs with two hidden layers, the kernel size of the first hidden layer was always larger than the kernel size of the second hidden layer. For both architectures, different kernel sizes and different numbers of channels were investigated. Based on the experimental results, a single hidden layer TDNN with a kernel size (10, 5) and 32 channels can provide the best performance; thus, the TDNN with this architecture was employed in the subsequent experiments.

6.3.1.2 Impact of Hidden Layer Size and Dropout

Once an appropriate range of NN hyperparameters had been obtained from the initial experiments, further experiments on training NNs on the entire training dataset were undertaken. The aim of the experiments was to study those factors that can impact the performance of networks. One of the most influential factors is the size of the networks, so the experiments first investigated the influence of the hidden layer size on classification accuracy. As LSTMs with two hidden layers are able to provide sufficient modeling capacity according to initial experiments, LSTMs with two hidden layers were trained, with the size of the hidden layers being extended from 200 to 700. As dropout can usually improve the generalization ability of NNs [HSK+12], the effectiveness of dropout in NN training was also explored.

The evaluation results are showed in table 6.2. For each trained model, the model architecture, the size of the memory footprint occupied by the model, the validation error, and the test error are shown. Four types of NNs, LSTM, GRU, TDNN-LSTM and FFNN,

Network	Architecture	Memomry Footprint(MB)	Validation Error	Test Error
LSTM	2*200	7	44.16	43.76
	2*512	25	41.81	41.36
	2*700	40	42.02	41.50
GRU	2*200	7	45.31	44.89
	2*512	22	42.48	42.03
	2*700	34	42.39	41.92
TDNN-LSTM	2*200	9	43.81	43.45
	2*512	30	41.64	41.22
	2*700	47	41.62	41.21
FFNN	3*300	10	50.49	50.02
	3*720	25	46.23	45.80
	4*1000	40	44.52	44.07
LSTM Dropout	2*512	25	41.12	40.68
	2*700	40	40.87	40.39

Table 6.2: Impact of hidden layer size and dropout.

were implemented. For each type of RNN, i.e. LSTM, GRU and TDNN-LSTM, models were trained with three different architectures, which were all RNN architectures with two hidden layers, but with 200, 512, and 700 nodes per layer, respectively.

The first three lines show the evaluation results of LSTMs. The LSTM with 200 nodes per layer is the smallest LSTM; its memory footprint is about 7MB, and the test error is 43.76%. The next LSTM has 512 nodes per layer; its footprint is 25MB, and the test error is 41.36%, which is lower than seen for the smallest LSTM. The largest LSTM has 700 nodes per layer; however, the test error is higher than that seen for the smaller LSTM with 512 nodes per layer. This increase in test error may be caused by the problem of overfitting or data sparsity.

GRUs were trained with the same architectures as LSTMs. As the results show, a GRU always has a smaller memory footprint and higher test error than the LSTM with the same architecture. For the TDNN-LSTM, three networks were also trained with the same architectures as LSTMs. TDNN-LSTMs need larger memory footprints due to the TDNN layers; however, TDNN-LSTMs with 200, 512, and 700 nodes per layer have slightly lower test errors than the LSTMs with the same architectures.

The architectures of FFNNs we investigated are different from the architectures of RNNs, as we intended to compare the accuracy of different models with roughly the same memory footprint. The smallest FFNN has three layers and 300 nodes per layer, its memory footprint is 10MB, which is slightly larger than the smallest LSTM. However, its test error is 50.02%, which is dramatically higher than the smallest LSTM. The largest FFNN shown in the table has four layers and 1000 nodes per layer; it has roughly the same footprint as the largest LSTM, and its test error is 44.07%. This is probably the best performance the FFNN can achieve, as an FFNN with a larger architecture, consisting of four layers with 1600 nodes per layer, was also trained, but the test error was 44.14%, higher than the test error of the smaller FFNN.

The last two lines shows the experimental results of LSTMs trained with dropout. We trained two LSTMs with the dropout rate of 0.3. The smaller LSTM has two layers and 512 nodes per layer, and it has a test error of 40.68%, which is lower than the LSTM with the same architecture trained without dropout. The larger LSTM has 700 nodes per layer, and the test error is 40.39%, which is also lower than the LSTM with the same architecture

Network	Architecture	Memomry Footprint(MB)	Validation Error	Test Error
LSTM	2*512	25	41.81	41.36
	3*512	33	41.65	41.17
	4*512	41	41.49	41.03
TDNN-LSTM	2*512	30	41.64	41.22
	3*512	38	40.90	40.53
	4*512	46	40.63	40.18
BLSTM	2*512	57	39.98	39.47

Table 6.3: Impact of hidden layer depth and BLSTM.

trained without dropout. Therefore, according to our experimental results dropout can improve the accuracy of LSTMs.

6.3.1.3 Impact of Hidden Layer Depth and BLSTM

There are various methods available to increase the complexity of NNs and explore their maximum performance. For instance, the hidden layer size can be extended, the number of hidden layers can be increased, and for RNNs, bidirectional RNN is a more delicate way to extend the network architecture and enhance modeling capability.

Section 6.3.1.2 shows that for LSTMs with two hidden layers, the performance decreases when we simply extend the hidden layer size from 512 to 700. However, we still have alternative methods to increase the complexity of LSTM, e.g. LSTM with more hidden layers and BLSTM. Table 6.3 presents the experimental results. For LSTM and TDNN-LSTM, we trained networks with two, three and four layers, each layer has always 512 nodes. The performance of networks always gets improved, when we increase the depth of networks. Furthermore, each TDNN-LSTM has lower test error than the LSTM with the same architecture. BLSTM with two layers achieves the best performance of all NNs, its test error is 39.47%, and about 2% lower than the LSTM with the same architecture.

6.3.1.4 Evaluation Time

As our aim is to build a real-time KWS system appropriate for mobile devices, the evaluation time of networks is critical. Therefore, we also evaluate the LSTM and TDNN-LSTM on a normal CPU. We selected about 33k frames from 50 utterances, which total 5.6 minutes, and the CPU used is Intel Core 2 Quad CPU Q9400. The evaluation time of the smallest LSTM with two layers and 200 nodes per layer is approximately 160s. The evaluation time of the smallest TDNN-LSTM with two layers and 200 nodes per layer is about 90s, which is less than the evaluation time of the smallest LSTM due to the convolution operation with stride of 2 in the TDNN layer. Hence, for two seconds of acoustic features the smallest LSTM needs less than one second for evaluation; this appears to meet the real-time requirement.

6.3.1.5 NN training for 42 phonemes and 18k senones

In this work the experiments on NN training focus on NNs for 6k senones. However, it is also meaningful to investigate the performance of NNs for other AMs such as 42 phonemes CI AM and 18k senones CD AM. In this section, we present the results of NN training for 42 phonemes and 18k senones.

Table 6.4 lists the evaluation results of NNs for 42 phonemes. For each type of RNN, i.e. LSTM, TDNN-LSTM and BLSTM, we trained a network with two layers and each layer

Network	Architecture	Memomry Footprint(MB)	Validation Error	Test Error
LSTM	2*512	13	22.18	22.46
BLSTM	2*512	34	21.90	22.13
TDNN-LSTM	2*512	18	22.29	22.53
FFNN	3*1000	14	26.11	26.40

Table 6.4: NNs for 42 phonemes CI AM.

Network	Architecture	Memomry Footprint(MB)	Validation Error	Test Error
LSTM	2*512	48	44.89	45.57
BLSTM	2*512	104	42.52	43.28
TDNN-LSTM	2*512	54	44.77	45.45
FFNN	3*1000	83	49.42	50.18

Table 6.5: NNs for 18k senones CD AM.

has 512 nodes, respectively. The FFNN has three layers with 1000 nodes per layer, and roughly the same memory footprint as the LSTM. The LSTM has slightly lower test error than the TDNN-LSTM, and the BLSTM still achieves the best performance. Moreover, all RNNs significantly outperform the FFNN.

We trained networks for 18k senones CD AM with the same architectures as for CI AM. The evaluation results are shown in table 6.5. The test errors of the LSTM and TDNN-LSTM models are very close, which are 45.57% and 45.45% respectively. The best performance is still achieved by the BLSTM, its test error is more than 2% lower than the LSTM. The senone classification accuracies of all RNNs are remarkably higher than the accuracy of the FFNN.

6.3.1.6 Summary

In this section, a comprehensive study of sequence classification using RNNs and FFNNs is presented. First, the experimental results of NNs for 6k senones are demonstrated. According to the results, the classification accuracies of RNNs such as LSTM and TDNN-LSTM are much higher than the accuracies of FFNNs with similar architecture sizes. In addition, extending the hidden layer size of LSTMs can increase their accuracy; however when the hidden layer size exceeds a certain threshold, the accuracy of LSTMs decreases due to overfitting or data sparsity. The experimental results show that training LSTMs with dropout can alleviate the problems of overfitting and data sparsity, as hidden neurons are randomly masked out during training, and this can help to prevent the co-adaptations of hidden neurons[HSK+12]. Other methods to increase the RNN complexity were explored, such as RNNs with more hidden layers and BRNNs. The performance of LSTMs and TDNN-LSTMs improves when the recurrent hidden layer depth of the networks increases, and BLSTM achieves the best performance among all of the networks.

For the CI AM with 42 phonemes and the CD AM with 18k senones, the experimental results of NN training are quite similar to those with the CD AM with 6k senones. The performance of LSTMs and TDNN-LSTMs still significantly exceed FFNNs with respect to phoneme and senone classification, and the best performance is achieved by BLSTMs.

Furthermore, due to the convolution operation with stride of 2 in both the time domain and frequency domain, TDNN-LSTMs can reduce the evaluation time and training time

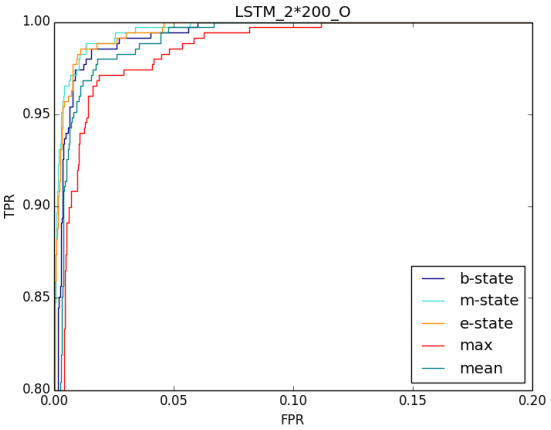

Figure 6.2: *The smallest LSTM using the pronunciation O and five different phoneme posterior representations.*

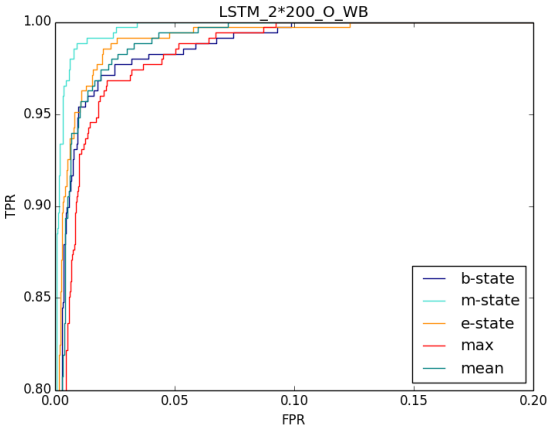

Figure 6.3: *The smallest LSTM using the pronunciation O_W and five different phoneme posterior representations.*

notably compared with LSTMs. TDNN-LSTMs can also slightly increase the accuracy of LSTMs with low memory cost.

6.3.2 Neural Network-based KWS

After NN training was complete, the KWS systems were created based on the trained NNs, and then evaluated on the evaluation dataset. This section first presents the evaluation results of KWS systems based on NNs for 6k senones, which are structured in three subsections. In the first subsection, different representation variants of phoneme posteriors are investigated; then, the best matched pronunciations for the key-phrase "okay cosa" are identified; and finally, a comparison among different NNs is offered with regard to their KWS performance. Finally, the evaluation results of KWS systems based on NNs for 42 phonemes and 18k senones are presented. As metrics we use the ROC curve and F-1 score.

6.3.2.1 Representation Variants of Phoneme Posteriors

The KWS performance of using different phoneme representations was investigated first. As discussed in section 5.3.2.2, there are five representations of phoneme posteriors, which are the posteriors of the single HMM states, that is, b-state, m-state, or e-state, or the mean or maximum posteriors of the three single states.

For a fixed NN and a fixed pronunciation, the ROC curves were plotted for all five phoneme posterior representations. Figure 6.2 shows the ROC graph of the smallest LSTM with two layers and 200 nodes per layer using the pronunciation O. The range of the x-axis is from 0 to 0.2, and the range of the y-axis is from 0.8 to 1. The dark blue curve, the light blue curve, and the orange curve represent using b-state m-state, and e-state posteriors as phoneme posteriors, respectively. The red curve and the teal curve represent using mean and maximum state posteriors. As shown in the ROC graph, the three curves of single states dominate the curves of the mean and max. In addition, the three curves of single states are close to one another, and the red curve of the max has the worst performance.

Figure 6.3 depicts the ROC graph of the smallest LSTM using the pronunciation O_WB. The light blue curve of m-state has the best performance, as all other curves are beneath the ROC curve of m-state. Different from the results of the smallest LSTM using the

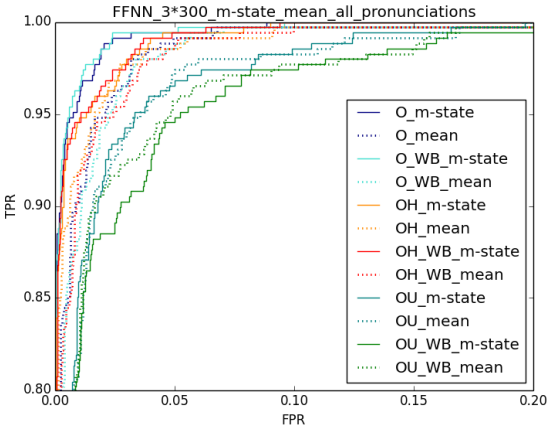

Figure 6.4: *The smallest FFNN using m-state posteriors, mean posteriors and six pronunciations.*

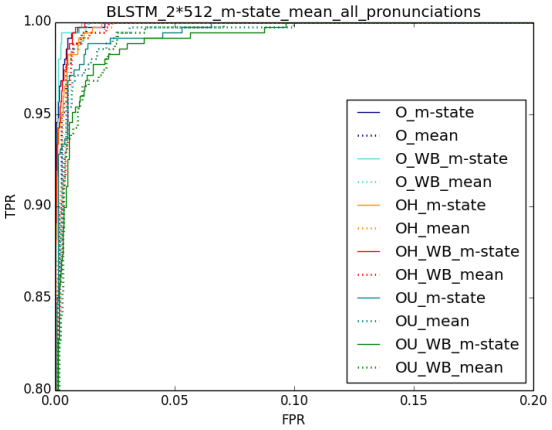

Figure 6.5: *BLSTM using m-state posteriors, mean posteriors and six pronunciations.*

pronunciation O, the teal curve of mean dominate the dark blue curve of b-state. However, the red curve of max still has the worst performance.

According to experimental results, using the maximum posteriors as phoneme posteriors always gives the worst performance. Using single state posteriors can give better performance in many cases and using mean posteriors can provide more stable performance. Thus, in subsequent experiments, only m-state and mean posteriors are used as phoneme posteriors.

6.3.2.2 Pronunciation Variants

Different pronunciations of the key-phrase "okay cosa" were investigated. Section 5.3.2.1 introduces the six pronunciations of "okay cosa", which are denoted by O, OH, OU, O_WB, OH_WB, and OU_WB. By asking volunteers to pronounce the phrase in the same way when collecting the evaluation data, best matched pronunciations for all the utterances collected were achieved. However for words or phrases uttered with different pronunciations, there are no fixed best matched pronunciations.

The ROC curves for a fixed NN were plotted using all six pronunciations. Figure 6.4 shows the ROC graph of the smallest FFNN with three layers and 300 nodes per layer. The solid curves are for using m-state posteriors as phoneme posteriors, and the dotted curves are for using mean posteriors as phoneme posteriors. The two dark blue curves belong to the pronunciation O, and the two light blue curves to the pronunciation O_WB. The dark blue curves and light blue curves dominate other curves for both solid and dotted curves.

Figure 6.5 presents the ROC graph of the BLSTM with two layers and 512 nodes per layer. The experimental results are quite similar to the results of the smallest FFNN. It is plain to see that the two dark blue curves and the two light blue curves still dominate other curves for both solid and dotted curves. The two teal curves for the pronunciation OU and the two green curves for the pronunciation OU with word boundary, i.e. OU_WB, are obviously the worst curves for both BLSTM and the smallest FFNN.

As demonstrated, the experiments show that the pronunciations O and O_WB are the best matched pronunciations for the key-phrase "okay cosa". Therefore, in the subsequent experiments, only the pronunciations O and O_WB were considered.

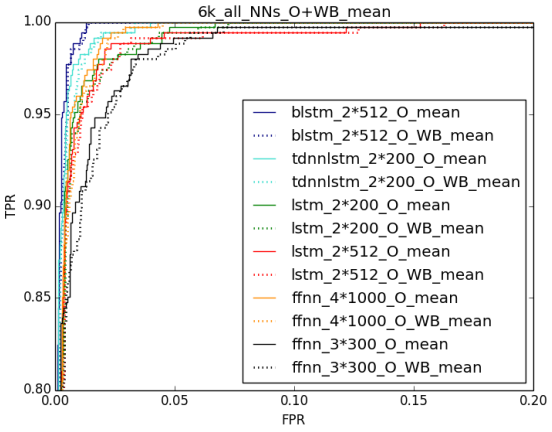

Figure 6.6: *ROC curves of six NNs for 6k senones using mean posteriors, the pronunciations O and O_W.*

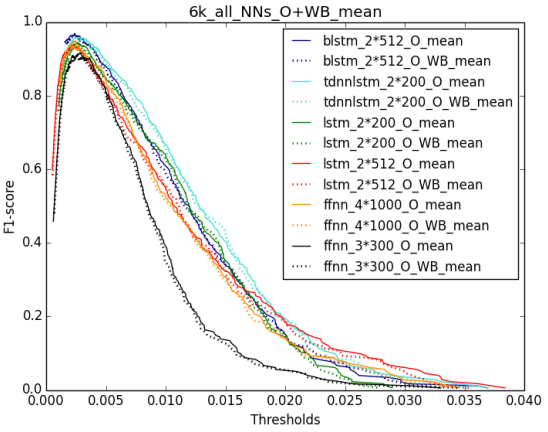

Figure 6.7: *F-1 scores of six NNs for 6k senones using mean posteriors, the pronunciations O and O_W.*

6.3.2.3 Comparison among Neural Networks

In the previous subsections the impact of different phoneme posterior representations and keyword/key-phrase pronunciations on KWS performance are analyzed. This subsection compares the KWS performance of different NNs based on the same phoneme posterior representations and key-phrase pronunciations. According to previous experiments, we employ mean state posteriors and m-state posteriors as phoneme posterior representations. For each phoneme posterior representation, two pronunciations, namely O and O_WB, are used as the best matched pronunciations for the key-phrase "okay cosa".

First of all, the performance of the smallest LSTM with two layers and 200 nodes per layer, the smallest TDNN-LSTM with two layers and 200 nodes per layer, and the smallest FFNN with three layers and 300 nodes per layer are evaluated, as a small memory footprint is critical to our KWS system. Besides, we also evaluated the performances of a larger LSTM with two layers and 512 nodes per layer, the best FFNN with four layers and 1000 nodes per layer, and the BLSTM with two layers and 512 nodes per layer. The ROC curves of all six NNs using mean state posteriors are presented in figure 6.6. The solid curves belong to the pronunciation O and the dotted curves belong to the pronunciation O_WB. The two curves of the same NN are plotted in the same color, for all NNs their two curves are very close to close to each other. Obviously, the two dark blue curves of BLSTM dominate all other curves. The two light blue curves of the smallest TDNN-LSTM are the second best, and the two black curves of the smallest FFNN are apparently the worst. The ROC curves of the rest three NNs including the smallest LSTM, the best FFNN and the larger LSTM, are very close to one another.

Figure 6.7 shows the F1-score graph of all NNs using mean state posteriors. The x-axis represents the threshold and the y-axis represents the F1-score. All NNs achieve their highest F1-scores at roughly the same threshold, and BLSTM still achieves the highest F1-score among all NNs.

Figure 6.8 displays the ROC curves of all six NNs using m-state posteriors as phoneme posteriors. The two dark blue curves of BLSTM still dominate all other curves. The red curves of the larger LSTM and the black curves of the smallest FFNN are the worst. The rest three NNs have similar performance. For all NNs except the larger LSTM, the performance of using m-state posteriors is better than using mean state posteriors.

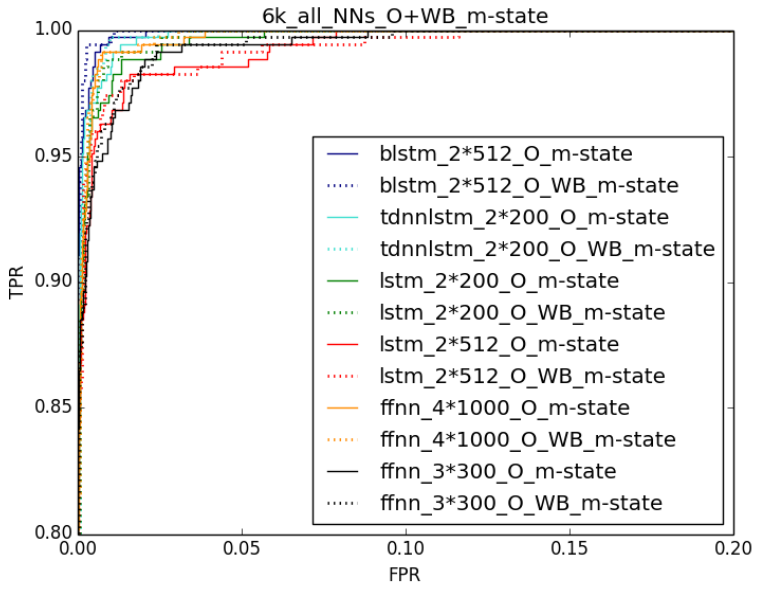

Figure 6.8: *Six NNs for 6k senones using m-state posteriors, the pronunciations O and O_W.*

6.3.2.4 KWS based on 18k senones CD AM and 44 phonemes CI AM

In this subsection the evaluation results of KWS systems based on NNs for 42 phonemes and 18k senones are demonstrated, and we use the KWS system based on the smallest LSTM for 6k senones with two layers and 200 nodes per layer as the baseline.

Figure 6.9 shows the ROC graph of all networks for 18k senones using mean posteriors. The red curves belong to the baseline, i.e. the smallest LSTM for 6k senones. As demonstrated in the ROC graph, all networks for 18k senones have similar performance and outperform the baseline notably, BLSTM for 18k senones has the best performance among all networks.

Figure 6.10 shows the ROC graph of all networks for 18k senones using m-state posteriors. In contrast with using mean posteriors, the ROC curves of all networks for 18k senones are beneath the red curves of the baseline. In order to investigate the KWS performance further, we plotted the ROC graph of BLSTM for 18k senones using the pronunciation O and all five phoneme posterior representations. As can be seen in figure 6.11, the performance of using m-state posteriors is obviously much worse than using other phoneme posterior representations. This is probably due to the problem of data sparsity.

Figure 6.12 depicts the ROC graph of all networks for 42 phonemes. The range of y-axis and x-axis is from 0 to 1. As the 42 phonemes AM is context independent, the word boundary is not modeled. For each NN for 42 phonemes, we only plot the ROC curve using the pronunciation O. The baseline is still the red curves of the smallest LSTM for 6k senones, which appear in the upper-left corner of the ROC graph. Evidently, the BLSTM achieves the best performance among all networks for 42 phonemes, and the performance of the baseline is much better than all networks for 42 phonemes.

6.3.2.5 Summary

This section first demonstrates experiments on KWS using NNs for 6k senones. The performance of five different phoneme posterior representations were investigated, and the results suggests that mean state posteriors can deliver more stable performance than single state posteriors, while single state posteriors can provide better performance in

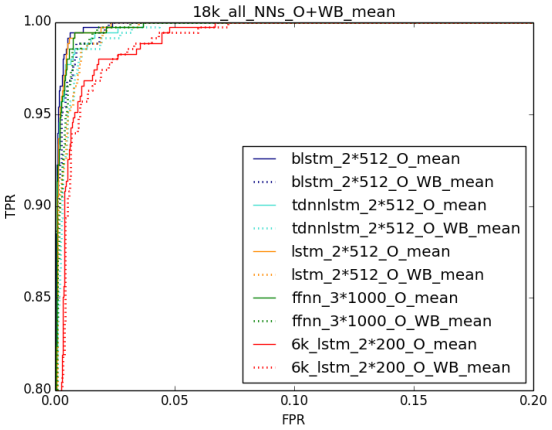

Figure 6.9: *Four NNs for 18k senones using mean posteriors, the pronunciations O and O_W.*

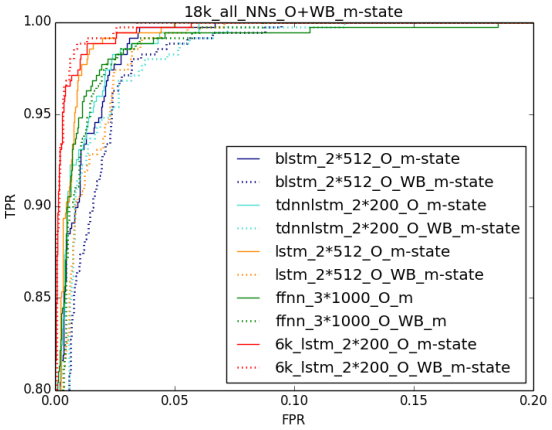

Figure 6.10: *Four NNs for 18k senones using m-state posteriors, the pronunciations O and O_W.*

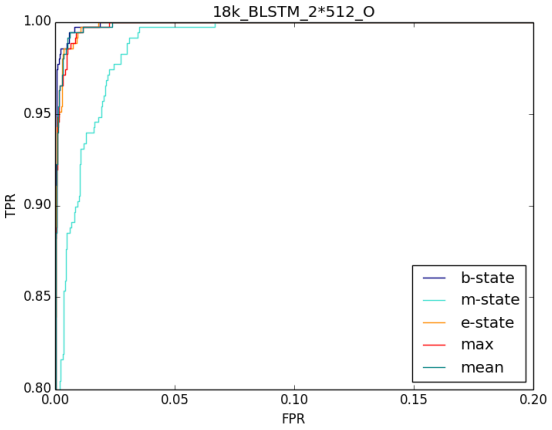

Figure 6.11: *BLSTM for 18k senones using the pronunciation O and five phoneme posterior representations.*

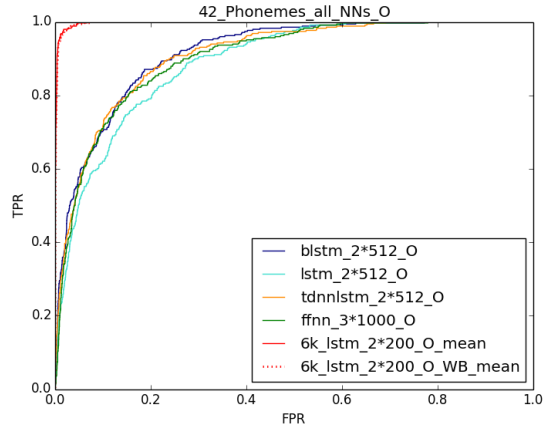

Figure 6.12: *Four NNs for 44 phonemes context-independent AM using mean posteriors and the pronunciation O.*

many instances. Afterwards, experiments were carried out to find out the best matched pronunciations of the key-phrase "okay cosa". Finally, the KWS performance of six different networks was compared based on the same phoneme posterior representations and key-phrase pronunciations. BLSTM achieves the best performance, while the smallest FFNN achieves the worst performance. Therefore, the KWS performance of NNs appears to be to some extent dependent on the senone classification accuracy.

The evaluation results of NNs for 42 phonemes and 18k senones are also presented. As has been pointed out, all networks for 18k senones outperform the baseline, which is the smallest LSTM for 6k senones, when mean state posteriors are used; in contrast to those for 18k senones, the smallest LSTM for 6k senones surpasses all NNs for 42 phonemes significantly. More senones can lead to improved KWS performance; however the network for more senones needs a larger memory footprint as a result of the larger output layer. Therefore, NNs for 6k senones provide the best trade-off between memory footprint and KWS performance. Further experiments on KWS using NNs for 18k senones show that the KWS performance of NNs for 18k senones using single state posteriors, e.g. m-state posteriors, is not stable, this is probably due to the problem of data sparsity.

7. Conclusion

In this work, a novel NN based approach to KWS has been proposed. Compared with conventional KWS systems, which have high computational costs due to Viterbi decoding, this KWS system achieves a small memory footprint, low latency, and low computational costs with high accuracy. Therefore, our KWS system is more appropriate for real-time KWS on mobile devices than conventional KWS systems. Moreover, our novel KWS approach also has advantages over another small footprint KWS approach, the Deep KWS. As our KWS approach uses NNs to estimate the posteriors for subword units, that is, phonemes or senones, rather than entire words, it is not necessary to collect a large amount of training data for each keyword and to train new NNs if new keywords are needed to be detected. Thus, this KWS approach is also flexible.

Furthermore, experiments on NN training demonstrate that RNNs such as LSTM, TDNN- and BLSTM outperform FFNNs in terms of phoneme/senone classification. TDNN-LSTM can reduce the training and evaluation time for LSTM, and also slightly improve the classification accuracy of LSTM. Further experiments on KWS using the trained NNs show that the small-footprint LSTM and TDNN-LSTM outperform FFNN on KWS. In addition, it is also evident from experiments that using mean state posteriors as phoneme posterior representation provides more stable KWS performance and that NNs for more senones can lead to higher KWS accuracy.

7.1 Further Work

This work has proven that the novel KWS system is able to detect the key-phrase “okay cosa” with high accuracy. It is, however, important to test the performance of the KWS system on other keywords/key-phrases. This can be realized by simply adapting the posterior handling module of the KWS system for the new keyword/key-phrase, and collecting an evaluation dataset for the new keyword/key-phrase.

As the goal of this thesis was to build a real-time KWS system for mobile devices, in further work, the deployment of the KWS system on mobile devices could be investigated. In order to achieve low latency, highly-optimized C code should be written to implement the evaluation of networks and the posterior handling module of the KWS system.

In this work, the phoneme and senone classification performance of different types of RNNs such as LSTM, TDNN-LSTM, and BLSTM were investigated. The metric used was the frame error rate, and experimental results show that RNNs have notably lower frame error

rates than FFNNs. Therefore, it may be also interesting to use posteriors generated by RNNs in general ASR tasks, and thus check whether the word error rate (WER) also decreases as compared with the use of FFNNs.

Bibliography

[AAA+16] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[BBB+10] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.

[Bis95] C. M. Bishop, *Neural networks for pattern recognition*. Oxford university press, 1995.

[BLP+12] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[BMWR92] H. Bourlard, N. Morgan, C. Wooters, and S. Renals, "Cdnn: A context dependent neural network for continuous speech recognition," in *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 2. IEEE, 1992, pp. 349–352.

[Bri90] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[CPH14] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4087–4091.

[DYDA12] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 30–42, 2012.

[FGH+97] M. Finke, P. Geutner, H. Hild, T. Kemp, K. Ries, and M. Westphal, "The karlsruhe-verbmobil speech recognition engine," in *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, vol. 1. IEEE, 1997, pp. 83–86.

[GAV00] J. S. Garofolo, C. G. Auzanne, and E. M. Voorhees, "The trec spoken document retrieval track: A success story," in *Content-Based Multimedia Information Access-Volume 1*. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 2000, pp. 1–20.

[GS05] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[HBF+01] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

[HH93] M.-Y. Hwang and X. Huang, "Shared-distribution hidden markov models for speech recognition," *IEEE Transactions on Speech and Audio Processing*, vol. 1, no. 4, pp. 414–420, 1993.

[HH06] W. J. Hardcastle and N. Hewlett, *Coarticulation: Theory, data and techniques*. Cambridge University Press, 2006.

[Hoc91] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," *Diploma, Technische Universität München*, vol. 91, 1991.

[HS97] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[HSK+12] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[HSW89] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[IS15] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[KHM+14] K. Kilgour, M. Heck, M. Müller, M. Sperber, S. Stüker, and A. Waibel, "The 2014 kit iwslt speech-to-text systems for english, german and italian," in *International Workshop on Spoken Language Translation (IWSLT)*, 2014.

[MHN13] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013.

[Mit97] T. M. Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.

[MKK+07] D. R. Miller, M. Kleber, C.-L. Kao, O. Kimball, T. Colthurst, S. A. Lowe, R. M. Schwartz, and H. Gish, "Rapid and accurate spoken term detection," in *Eighth Annual Conference of the International Speech Communication Association*, 2007.

[Nes83] Y. Nesterov, "A method of solving a convex programming problem with convergence rate o (1/k2)," 1983.

[PNH86] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning back propagation," Carnegie–Mellon University, Pittsburgh, PA, Tech. Rep. CMU–CS–86–126, 1986.

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, 1988.

[RJ86] L. Rabiner and B.-H. Juang, "An introduction to hidden markov models," *ASSP Magazine, IEEE*, vol. 3, no. 1, pp. 4–16, 1986.

[Ros61] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," CORNELL AERONAUTICAL LAB INC BUFFALO NY, Tech. Rep., 1961.

[RP90] R. C. Rose and D. B. Paul, "A hidden markov model based keyword recognition system," in *Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on.* IEEE, 1990, pp. 129–132.

[RRRG89] J. R. Rohlicek, W. Russell, S. Roukos, and H. Gish, "Continuous hidden markov modeling for speaker-independent word spotting," in *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on.* IEEE, 1989, pp. 627–630.

[RT03] L. Rodríguez and I. Torres, "Comparative study of the baum-welch and viterbi training algorithms applied to read and spontaneous speech recognition," *Pattern Recognition and Image Analysis*, pp. 847–857, 2003.

[SC78] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[SMFW01] H. Soltau, F. Metze, C. Fugen, and A. Waibel, "A one-pass decoder based on polymorphic linguistic context assignment," in *Automatic Speech Recognition and Understanding, 2001. ASRU'01. IEEE Workshop on.* IEEE, 2001, pp. 214–217.

[SP97] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[SVSS15] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on.* IEEE, 2015, pp. 4580–4584.

[Wer88] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural networks*, vol. 1, no. 4, pp. 339–356, 1988.

[Wer90] ——, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[WHH+89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[WMM91] J. Wilpon, L. Miller, and P. Modi, "Improvements and applications for key word recognition using hidden markov modeling techniques," in *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on.* IEEE, 1991, pp. 309–312.

[WZ95] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[YOW94] S. J. Young, J. J. Odell, and P. C. Woodland, "Tree-based state tying for high accuracy acoustic modelling," in *Proceedings of the workshop on Human Language Technology.* Association for Computational Linguistics, 1994, pp. 307–312.

[Zhu15] L. Zhu, "Gmm free asr using dnn based cluster trees," 2015.

[ZRM+13] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean *et al.*, "On rectified linear units for speech processing," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 3517–3521.

List of Figures

List of Tables