TIMIT Speech Recognition Using CTC

MLMI2 - Speech Recognition

16/12/2024

Candidate no.: 2108I

Word count: 2998[1]

[1]Excluding appendix, tables, footnotes, images and titles

Trumpington Street
Cambridge CB2 1PZ
Tlf.: +44 01223 332600
http://www.eng.cam.ac.uk/

UNIVERSITY OF CAMBRIDGE

Table of Contents

1 Introduction

This coursework investigates the development of an end-to-end automatic speech recognition (ASR) system using the Connectionist Temporal Classification (CTC) loss function, applied to the TIMIT acoustic-phonetic corpus.

In particular, the coursework explores the impact of model regularisation techniques in mitigating overfitting and improving generalisation, as well as investigates various optimisers such as stochastic gradient descent and Adam to understand their influence on model convergence and stability.

Furthermore, the study incorporates experiments on model complexity, evaluating how architectural choices such as the number of LSTM layers and hidden units affect the trade-off between performance and computational cost.

This project also involves visualising and interpreting key metrics such as training loss, validation loss, and decoding accuracy over time, along with phoneme-level alignment from the CTC decoder.

The total time spent completing this coursework was 1 hour on the CPU and 12 hours on the GPU. I appoximately spent an euqal amount of GPU time on each part.

2 Fundamentals

2.1 TIMIT Database

TIMIT (Texas Instruments/Massachusetts Institute of Technology) dataset is a widely used speech corpus for speech recognition research. The dataset is phonetically transcribed at the word and phoneme level, making it ideal for tasks such as automatic speech recognition (ASR) and phoneme recognition.

2.2 LSTM Model

LSTM (Long Short-Term Memory) models are a type of recurrent neural network (RNN) designed to handle sequential data, such as speech or text, by maintaining a memory of past inputs in a sequence.

In this coursework, a Bidirectional LSTM (BiLSTM) is used for speech recognition tasks. Based on the BiLSTM model, this coursework investigates various training strategies and LSTM structures to explore optimal configurations that can achieve higher accuracy.

2.3 CTC Loss

CTC Loss is widely used in sequence-to-sequence tasks, such as speech recognition, to address the challenge of aligning input and output sequences of varying lengths without requiring explicit alignment labels.

CTC introduces a blank token b, which allows the model to learn flexible alignments between input and target sequences by enabling silent or non-output time steps.

2.4 Evaluation Metrics

This course work evaluates the model's performance using multiple methods, including CTC Loss across various datasets, as well as Substitution Rate, Deletion Rate, Insertion Rate, Correct Rate, and Error Rate.

3 Data Processing and Exploration

3.1 Creating Input Acoustic Features

In this coursework, the dataset used consists of speech samples from the TIMIT corpus, which is structured in a JSON format. Each entry in the dataset corresponds to a specific audio file, identified by its filename (e.g., "FVFB0_SX222.WAV"). The key attributes of each entry are as follows:

Based on the experimental guidelines, the feature generation process in this work is as follows:

1. **FBank Feature Extraction:** Using torchaudio.compliance.kaldi.fbank, the waveform source files (paths specified in the "wav" field of the JSON file) are converted into FBank features. These features are then stored within speaker-specific folders.

2. **JSON Metadata File Creation**: A JSON file is generated to serve as the source metadata for the FBank features and related information.

3.2 Phone Vocabulary Set

Since CTC includes an additional blank token, a set of 40 symbols (39 phones plus the blank token) was created to serve as the output vocabulary, and this file is named vocab_39.txt. This file functions as the output dictionary for the LSTM network, with each symbol representing a unique phone in the list.

3.3 Phone Frequencies Distribution Visualisation

After the above analysis, the generated files were used to read the dataset, and the word frequency of the true phonemes in the training set was counted and visualized. The frequency histogram and word cloud, are shown in Figures 3.1 and 3.2.

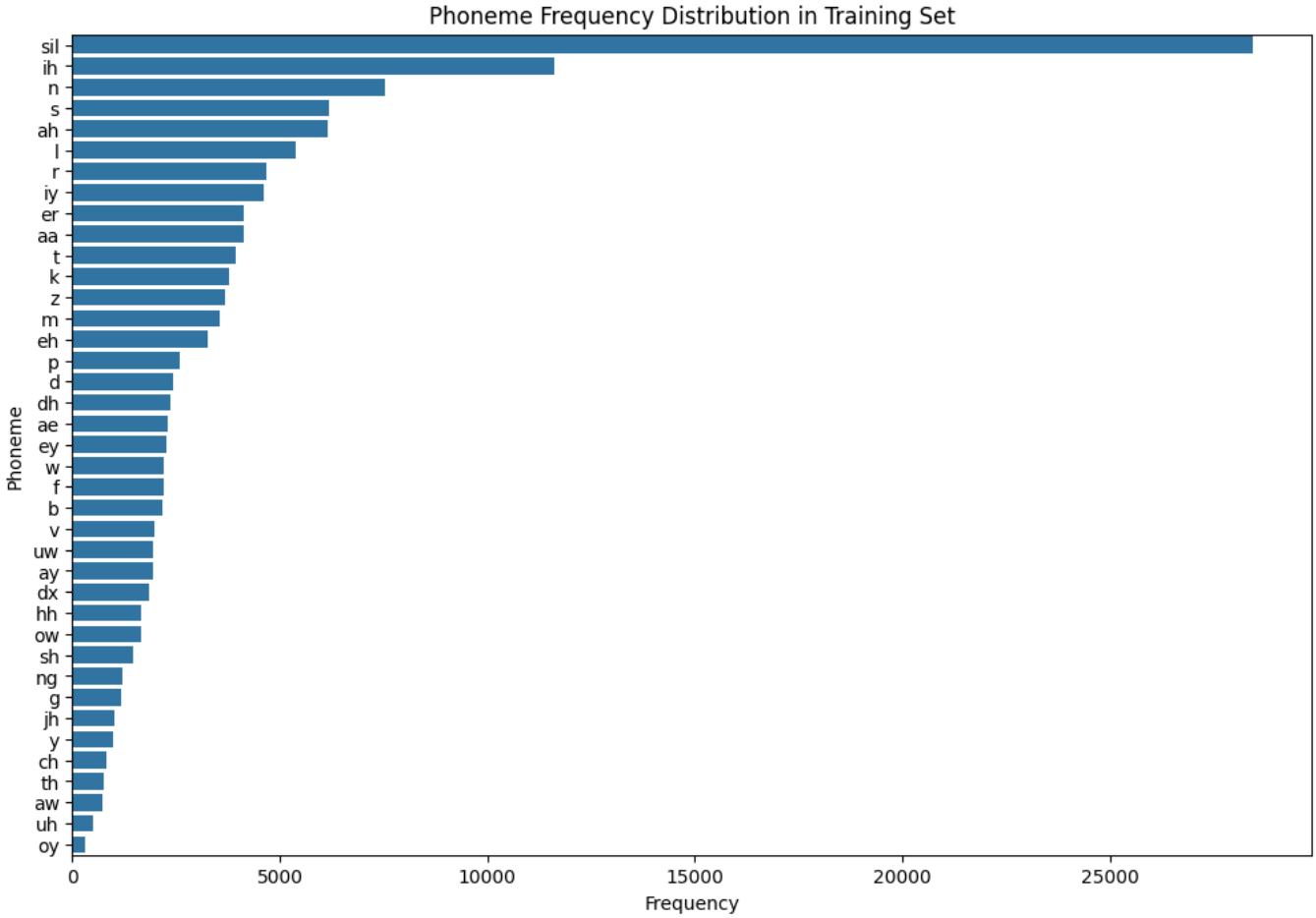


Figure 3.1 Frequency distribution in training set

The figures illustrate that the "sil" phoneme has the highest frequency, followed by "ih," "n," "s," and "ah." This is related to the characteristics of the language.

First, "sil" phoneme, representing silence, is frequently present in the training data due to natural pauses between words or sentences, especially at the beginning or end of utterances.

Figure 3.2 Word Cloud

Secondly, Phonemes like "ih" (a short vowel sound), "n" (a nasal consonant), "s" (a fricative consonant), and "ah" (a common vowel sound) are prevalent across a wide range of words in English. These phonemes appear frequently in many words and are often part of syllables that occur frequently in speech.

These frequencies reflect the natural structure of the language, with sounds like "s" and vowel sounds like "ih" and "ah" being used across various contexts. The training data likely mirrors these natural patterns, and preprocessing steps such as segmentation may also contribute to the prevalence of certain phonemes.

4 End-to-End ASR with CTC

4.1 Model configuration

Each model adjustment corresponds to a distinct configuration. In this course work, "args" is used to configure the model. The meanings of the different fields in "args" are as follows:

```
args = {
    "seed": Random seed for reproducibility,
    "train_json": Path to the training data with FBank in JSON format,
    "dev_json": Path to the validation data with FBank in JSON format,,
    "test_json": Path to the test data with FBank in JSON format,
    "batch_size": Number of samples per batch for training and evaluation,
    "num_layers": Number of LSTM layers in the network,
    "fbank_dims": Number of mel filterbanks,
    "model_dims": The dimensionality of the hidden states in the LSTM model
    "concat": Whether to concatenate features (1 is using concatenate),
    "lr": learning rate of the optomiser,
    "vocab": The output vocabulary for the model,
    "report_interval": Frequency at which to report training progress,
    "num_epochs": Number of epochs for training the model,
    "max_norm": Control the maximum gradient when using gradient clipping,
    "device": Device to run the model on, e.g., 'cpu' or 'cuda' for GPU
}
```

Subsequent modifications to the model are primarily made by adjusting the parameters here.

4.2 Changes in Loss on Training and Validation Set

By training the model, the losses over epochs were obtained. The losses for both the training set and the development (dev) set show a downward trend, indicating that the model is learning and improving over time, which are shown below.

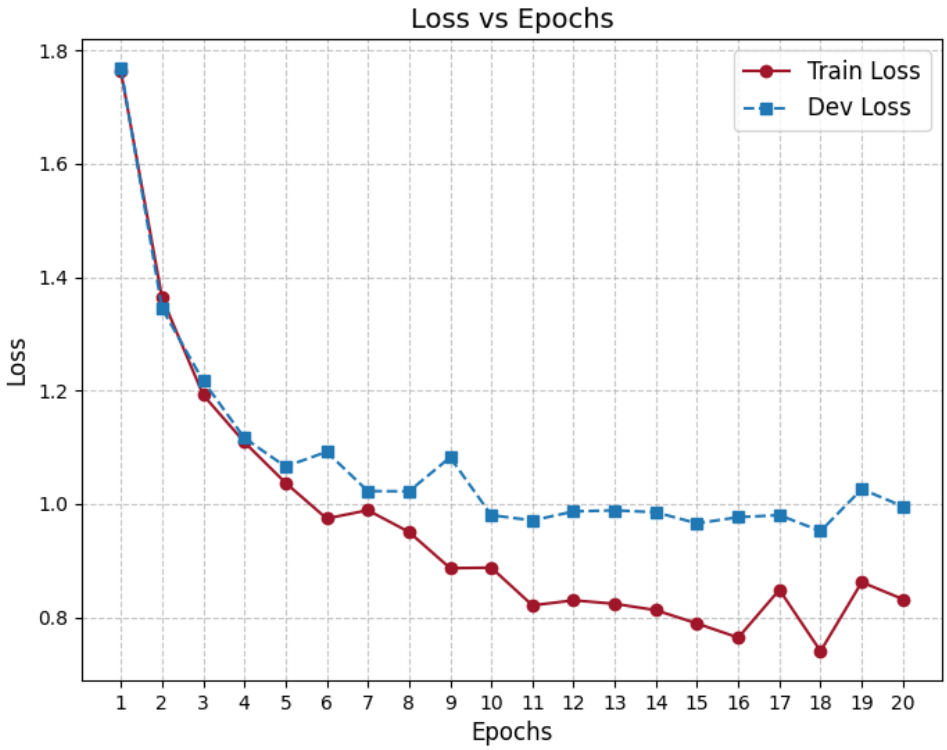


Figure 4.1 Loss vs epochs on training and validation set

However, the training loss decreases at a faster rate compared to the dev loss. This is a common phenomenon, as the model has access to the full training data and can adjust its weights more quickly to minimize the error. In contrast, the dev set, which is a subset of speakers from the original TIMIT test set, serves as a validation of the model's generalization capability.

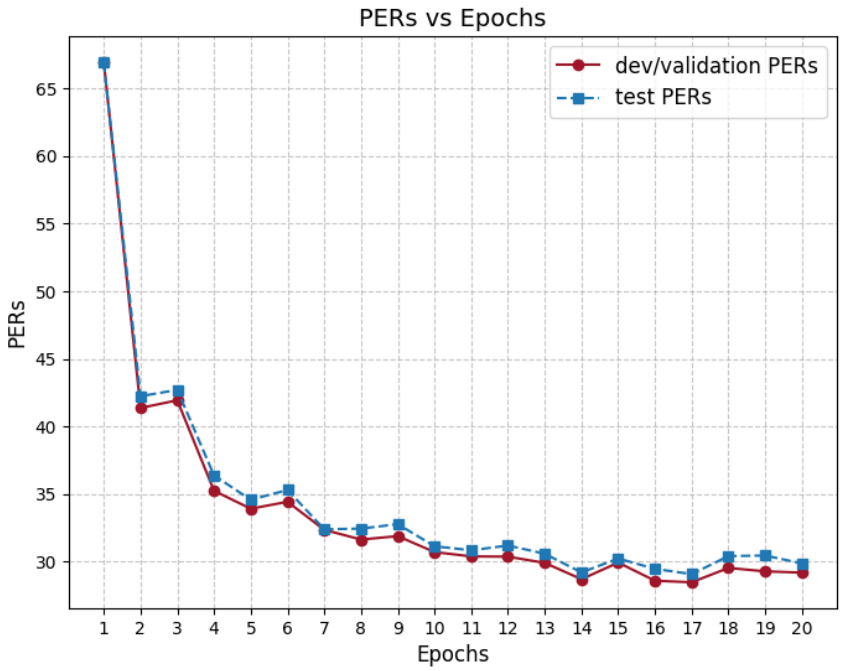


Figure 4.2 PERs vs Epochs

From Figure 4.2, it can be observed that the Phone Error Rates (PERs) for both the dev/validation set, and the test set follow the same downward trend and exhibit similar values. This is likely because the dev set is composed of a subset of speakers from the original TIMIT full test set, resulting in similar acoustic and phonetic characteristics between the two sets.

From this point onward, different model configuration will be explored by focusing on regularization techniques, optimizers, and the structure of the LSTM.

4.3 Regularisation

Based on the above model settings, and keeping other parameters at their default values, we investigate the impact of two regularization methods—dropout and gradient clipping.

4.3.1 Dropout

First, we study the dropout method.

Dropout is a regularization technique used to prevent overfitting by randomly setting a fraction of input units to zero during training. In the case of an LSTM, Dropout can be applied in two places in terms of coding:

(1) **Within the LSTM layers**: When args.num_layers > 1, Dropout is applied between the outputs of each LSTM layer, controlled by the dropout parameter in nn.LSTM. If args.num_layers = 1, this parameter is ignored;

(2) **After the LSTM output**: An additional self.dropout layer is applied to the final hidden state output from the LSTM to further regularize the model.

With other parameters kept at their default values, experiments were conducted by setting a series of dropout configurations. The table below presents the model configurations and experimental results. For instance, the configuration labeled "output/hidden, prob=0.5, 2 layers" indicates that the model applied dropout to the output or hidden layer with a dropout rate of 0.5, and the LSTM network consisted of 2 layers.

Table 4.1 Experiment results: different dropout configurations

Setting	Loss train	Loss Valid	SUB	DEL	INS	COR	PER
No dropout 1 layer*	0.78163	1.00539	17.08%	10.07%	3.14%	72.85%	29.90%
Output, prob=0.5 1 layer*	0.99778	0.99106	15.71%	14.65%	1.67%	69.64%	31.04%
Hidden, prob=0.5 2 layers	0.92782	0.84628	15.75%	9.24%	1.76%	75.01%	26.74%
Hidden, prob=0.1 2 layers	0.65813	0.81519	15.09%	8.08%	2.02%	76.83%	25.02%
Hidden, prob=0.3 2 layers*	0.68994	0.78127	14.67%	8.10%	1.96%	77.23%	24.73%
Hidden, prob=0.7 2 layers	1.07041	0.90865	16.14%	11.85%	1.18%	72.01%	29.17%
Hidden, prob=0.9 2 layers*	1.66123	1.35409	6.49%	46.86%	0.08%	46.65%	53.43%

Based on the table, key observations are discussed below.

In the absence of dropout: the model achieved a Phone Error Rate (PER) of 29.90%, with the train loss lower than the validation loss, indicating overfitting. The model memorized the training data but struggled with generalization.

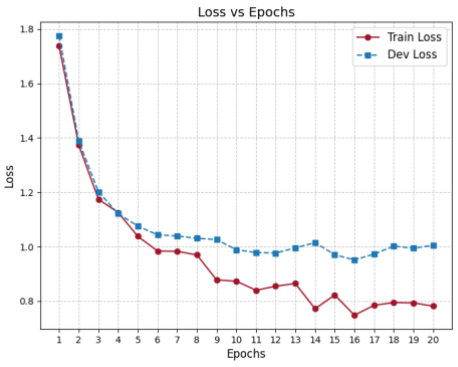
Dropout on Output Layer (Probability = 0.5): the performance slightly worsened, with PER increasing to 31.04%. This suggests that dropout on the output layer alone is not sufficient for regularization in this setup.

Dropout on Hidden Layers: When dropout was applied to the hidden layers of a two-layer LSTM, varying probabilities were tested to examine the trade-off between regularization and performance.

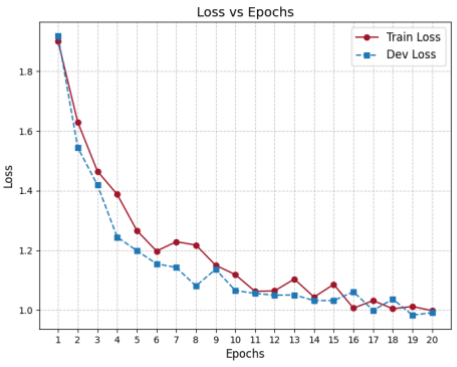
(1) With a **0.3 dropout probability**, the PER decreased to 24.73%, demonstrating enhanced generalization while maintaining low error rates;

(2) At a **higher dropout rate of 0.7**, the model's performance declined, with COR dropping to 72.01% and PER rising to 29.17%, indicating excessive regularization that hindered the model's performance.

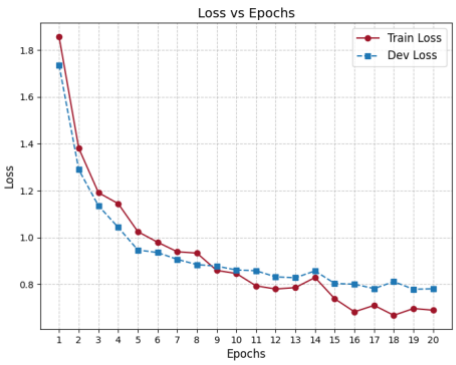
(3) When **the dropout probability was set to 0.9**, the model experienced severe underfitting. This extreme dropout rate overly penalized the model's learning capacity, resulting in poor generalization and high error rates.

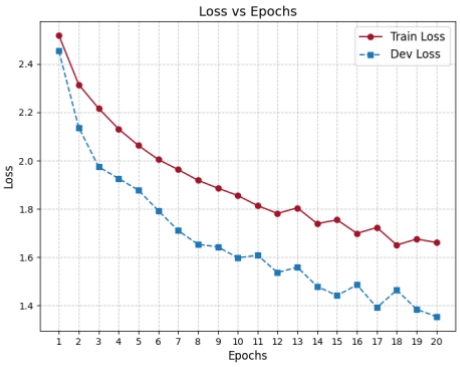


(a) No dropout



(b) Output, prob=0.5,1 layer





(c) Hidden, prob=0.3, 2 layers (d) Hidden, prob=0.9, 2 layers

Figure 4.3 Loss vs Epochs with different settings

From Figure 4.3, it can be observed that compared to the setup without dropout (Figure a), adding dropout layers (Figures b and c) significantly alleviates the situation where the Dev Loss is noticeably higher than the Train Loss in the later stages, indicating that dropout layers can reduce overfitting to some extent.

However, with a higher dropout rate, the Loss on the dev dataset becomes much lower than that on the train dataset. This may be because dropout is turned off during evaluation, allowing the model to utilize its full capacity, thereby reducing the dev loss. Additionally, the extreme dropout rate prevents the model from fitting the training data well, but it avoids overfitting and improves generalization on the dev dataset.

4.3.2 Gradient Clipping

Next, we applied the gradient clipping method, a technique used to prevent exploding gradients during training by capping the gradients' norm to a predefined threshold.

Setting other parameters to the optimal configuration identified in Section 4.3.1—namely, using a 2-layer LSTM with dropout applied to the hidden layers at a probability of 0.5 (as shown in the third row of Table 4.1)—we explored the impact of gradient clipping on model performance by varying the thresholds for the max_norm

parameter. By setting different thresholds for the “max_norm” parameter, we investigated the effect of gradient clipping on model performance.

Table 4.2 Experiment results: different gradient clipping configuration

Setting	Loss Train	Loss Valid	SUB	DEL	INS	COR	PER
max_norm=1	0.77857	0.78676	14.98%	7.88%	1.93%	77.13%	24.79%
max_norm=3	0.79947	0.79175	14.77%	9.08%	1.58%	76.15%	25.43%
max_norm=5	0.79625	0.78963	14.52%	9.37%	1.47%	76.11%	25.36%
max_norm=∞	0.92782	0.84628	15.75%	9.24%	1.76%	75.01%	26.74%

The last row represents no gradient clipping, which is the same as the third row in Table 4.1. Compared to not using gradient clipping (where max_norm is set to infinity), models with a maximum gradient value show a reduction in both loss and PER. The reason is that by capping the gradients, the model is less likely to make large, erratic weight updates, leading to a more stable and effective training process.

It can be observed that the model performs best when max_norm=3, this value will be selected for further experiments.

4.4 Optimiser

4.4.1 Comparation between SGD and Adam

Stochastic Gradient Descent (SGD) is a widely used optimization algorithm that updates model parameters by calculating the gradient of the loss function with respect to each parameter. It uses a fixed global learning rate throughout training.

In contrast, Adam (Adaptive Moment Estimation) is an optimization method that combines the benefits of momentum and adaptive learning rates, automatically

adjusting the learning rate for each parameter (but not global learning rate) based on the first and second moments of the gradients.

Next, we investigate how different optimizers, such as SGD and Adam, and varying global learning rate values impact the performance of the model. The experiment results are as follows.

Table 4.3 Experiment results: different optimizer and lr configuration

Setting	Loss Train	Loss Valid	SUB	DEL	INS	COR	PER
SGD, lr=0.01	2.29097	2.27900	4.50%	74.21%	0.01%	21.30%	78.71%
SGD, lr=0.05	1.34279	1.20544	16.13%	23.29%	1.06%	60.57%	40.49%
SGD, lr=0.1	1.08439	0.98454	16.33%	13.85%	1.55%	69.83%	31.72%
SGD, lr=0.5	0.77800	0.80994	14.91%	8.44%	1.80%	76.65%	25.14%
SGD, lr=0.75	0.77154	0.79660	14.81%	8.68%	1.64%	76.51%	25.13%
SGD, lr=1.0	0.77105	0.81567	14.75%	8.60%	1.71%	76.66%	25.05%
Adam, lr=1e-5	2.26475	2.25189	0.43%	82.31%	0.00%	17.26%	82.74%
Adam, lr=1e-4	1.15903	1.07193	14.58%	20.31%	1.37%	65.11%	36.26%
Adam, lr=5*1e-4	0.71564	0.77347	14.69%	7.41%	2.07%	77.90%	24.17%
Adam, lr=1e-3	0.65431	0.78876	14.32%	7.39%	1.90%	78.30%	23.60%
Adam, lr=5e-3	1.05901	0.98560	18.24%	11.42%	1.47%	70.34%	31.14%

From the table, some key conclusions can be drawn.

For **SGD optimiser**, a learning rate of 0.01 resulted in high losses and the highest PER (78.71%), indicating slow model learning and poor performance. As the learning rate increased to 0.05 and 0.1, both train and validation losses improved significantly, and PER decreased to 40.49% and 31.72%, respectively, with reduced overfitting. Further increasing the learning rate to 0.5 and 1.0 resulted in similar performance, with slight variations in loss and PER, achieving a good balance between training and

validation loss but showing signs of overfitting as the validation loss slightly increased.

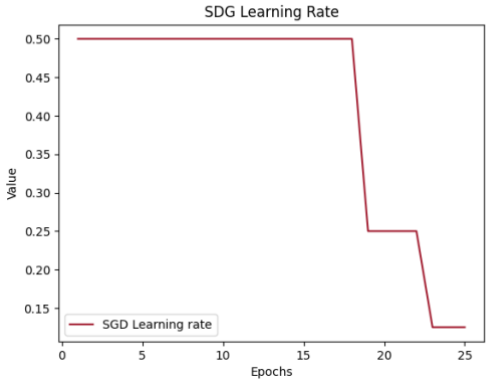
For the Adam optimizer, very low learning rates (1e-5 and 1e-4) led to poor performance and underfitting. Increasing the learning rate to 5e-4 and 1e-3 significantly improved performance, reducing both training and validation losses, and PER to 24.17% and 23.60%, respectively, while maintaining a good balance and generalization. However, a higher learning rate of 5e-3 caused an increase in training loss and PER (31.14%), suggesting convergence issues and potential overfitting.

Above all, Adam generally performs better than SGD at lower learning rates, as Adam allows for more efficient training and optimization. While SGD struggles with very low learning rates, Adam adapts better to smaller learning rates, leading to better convergence.

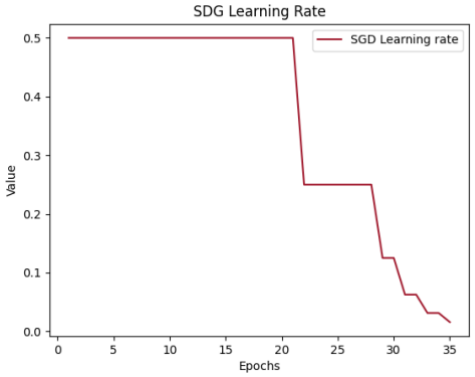
4.4.2 Learning Rate Scheduler

A learning rate scheduler is a technique used to adjust the learning rate during training to improve model performance and convergence speed. One commonly used scheduler is ReduceLROnPlateau, which reduces the learning rate when a monitored metric (such as validation loss) stops improving for a specified number of epochs. This adjustment helps fine-tune the model during the later stages of training, preventing overshooting and promoting better convergence.

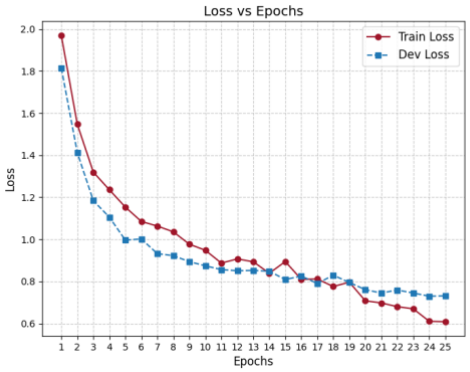
In this experiment, the SGD optimizer is used with an initial learning rate of 0.5. The model will be trained for 20, 25, and 35 epochs, with changes in loss and learning rate being recorded. This setup will help observe the effects of learning rate decay and its impact on the training process.



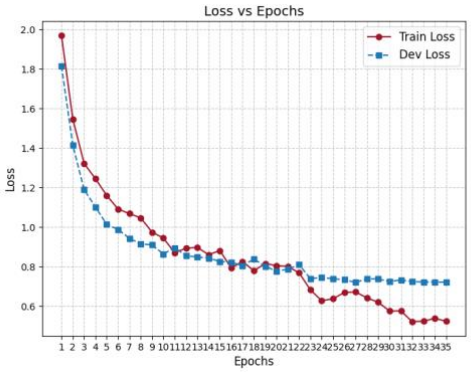
(a) lr - 25 epochs



(b) lr - 35 epochs



(c) Loss - 25 epochs



(b) Loss - 35 epochs

Figure 4.4 Change of learning rate and loss over epochs

From the figure above, it can be observed that at each point where the validation loss increases, the learning rate is halved. This often leads to a period of slower but more stable loss decrease, indicating that the model is fine-tuning its parameters more cautiously. These points where the learning rate is reduced can be clearly observed as periods in the graph where the loss shows a slower, more gradual decrease or even stabilizes for a few epochs before continuing to decline, reflecting the model's adaptation to the new, lower learning rate.

Based on the above study, the SGD optimizer is chosen with an initial learning rate of 0.5, and the ReduceLROnPlateau method is used for learning rate adjustment

to achieve optimal results. Subsequently, the model will be adjusted using the above settings.

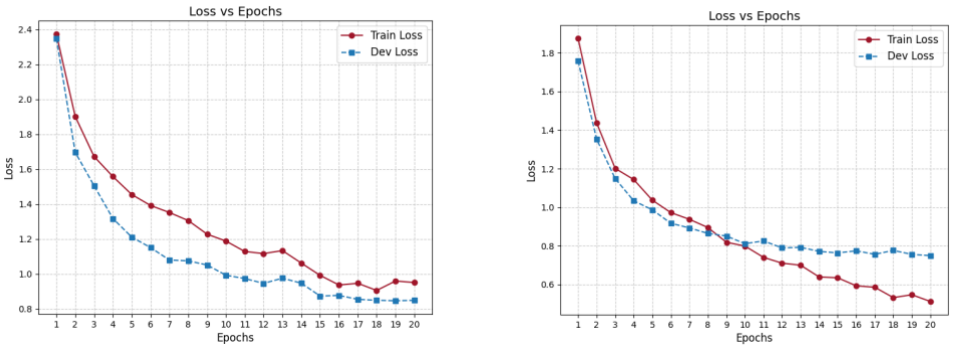
4.5 Model Complexity

Further exploration can be conducted by adjusting the model structure to find the optimal configuration. The following experiments are carried out by setting different numbers of layers, varying the number of hidden neurons, and modifying the bidirectional LSTM to a unidirectional structure. The results are shown in the table below.

Table 4.4 Experiment results: different structure of LSTM

Setting	Loss Train	Loss Valid	SUB	DEL	INS	COR	PER	#Params
1 layer	0.87984	0.91744	14.67%	13.98%	1.39%	71.35%	30.03%	166952
2 layer	0.71446	0.75305	14.19%	7.33%	1.97%	78.48%	23.49%	562216
3 layers	0.74149	0.73187	14.33%	7.06%	1.79%	78.61%	23.18%	957480
2-layer with multi feed forward layers	0.95041	0.84806	15.62%	7.48%	2.44%	76.90%	25.54%	589992
3-layer with muilt feed forward layers	0.96789	0.82776	15.47%	7.01%	2.47%	77.52%	24.95%	985256
2-layer with width 512	0.51201	0.74959	13.56%	6.37%	2.45%	80.08%	22.37%	8540200
Uni-directional LSTM	1.13784	1.03722	17.88%	12.75%	1.31%	69.37%	31.94%	215592

As shown in the table, with the increase in the number of layers, both loss and PER show a gradual decreasing trend, but the difference between the 2-layer and 3-layer networks is not significant, but 3-layer model has a great amount increase in the parameters, which may cause huge waste of training resources. Additionally, compared to models without added feed-forward layers, the models with feed-forward layers show a slight increase in PER and loss.



(a) 2 layer & multi feed-forward (b) 3 layer & multi feed-forward

Figure 4.5 Change of loss over epochs for different LSTM structure

As shown in the above figure, models with additional feed-forward layers cause the 2-layer LSTM loss to plateau, with little further decrease, indicating the model has reached its limit. The 3-layer LSTM model exhibits a generally higher Dev Loss, suggesting overfitting. Therefore, increasing the number of feed-forward layers is not a good choice.

In addition, for the 2-layer LSTM with 512 hidden neurons, although the PER is smaller, it can be observed that the train loss is significantly lower than the dev loss, indicating overfitting.

For the Uni-directional structure, the model performs worse across all metrics.

In conclusion, a simple 2-layer Bi-LSTM is the better choice.

4.6 Model Selection

Based on the above analysis, the optimal model structure is as follows:

- 2-layer Bi-LSTM.
- Dropout applied to the hidden layers with a probability of 0.5.
- Gradient clipping with a max_norm of 3.
- SGD optimizer with ReduceLROnPlateau for learning rate adjustment, and an initial learning rate of 0.5.

For this model, the final loss on the test and dev datasets is approximately 0.7, with a PER of around 23%.

5 Decoding

The decoding process starts by extracting the output probabilities of the trained LSTM model, which are then squeezed to remove any unnecessary dimensions. The phoneme vocabulary is loaded from a file, and the model's output is converted into a NumPy array.

To focus on a specific portion of the output, only the first 30 time steps are selected. Next, select the third data in test set, and then a heatmap is generated to visualize the predicted probabilities for each phoneme at every time step.

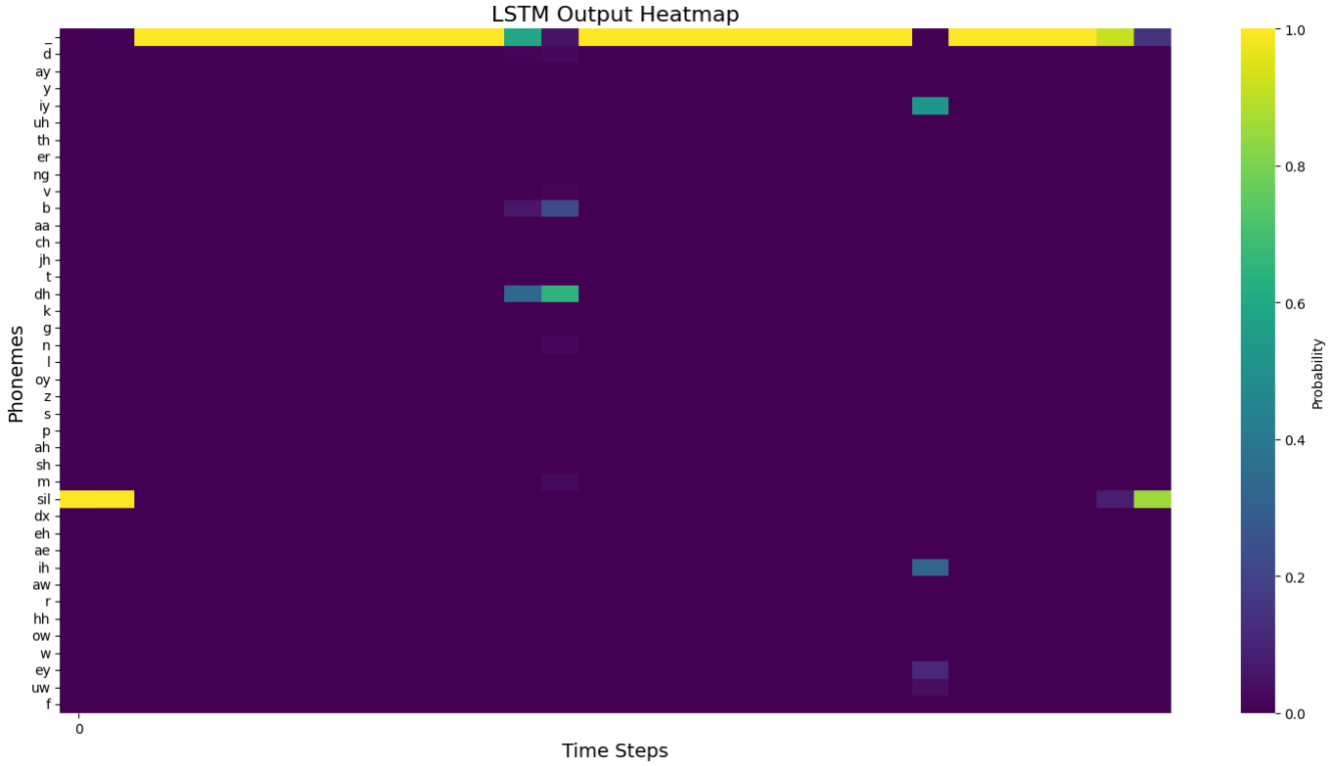


Figure 5.1 Heatmap of an utterance in test set - 30 steps

The heatmap for all time steps is shown below.

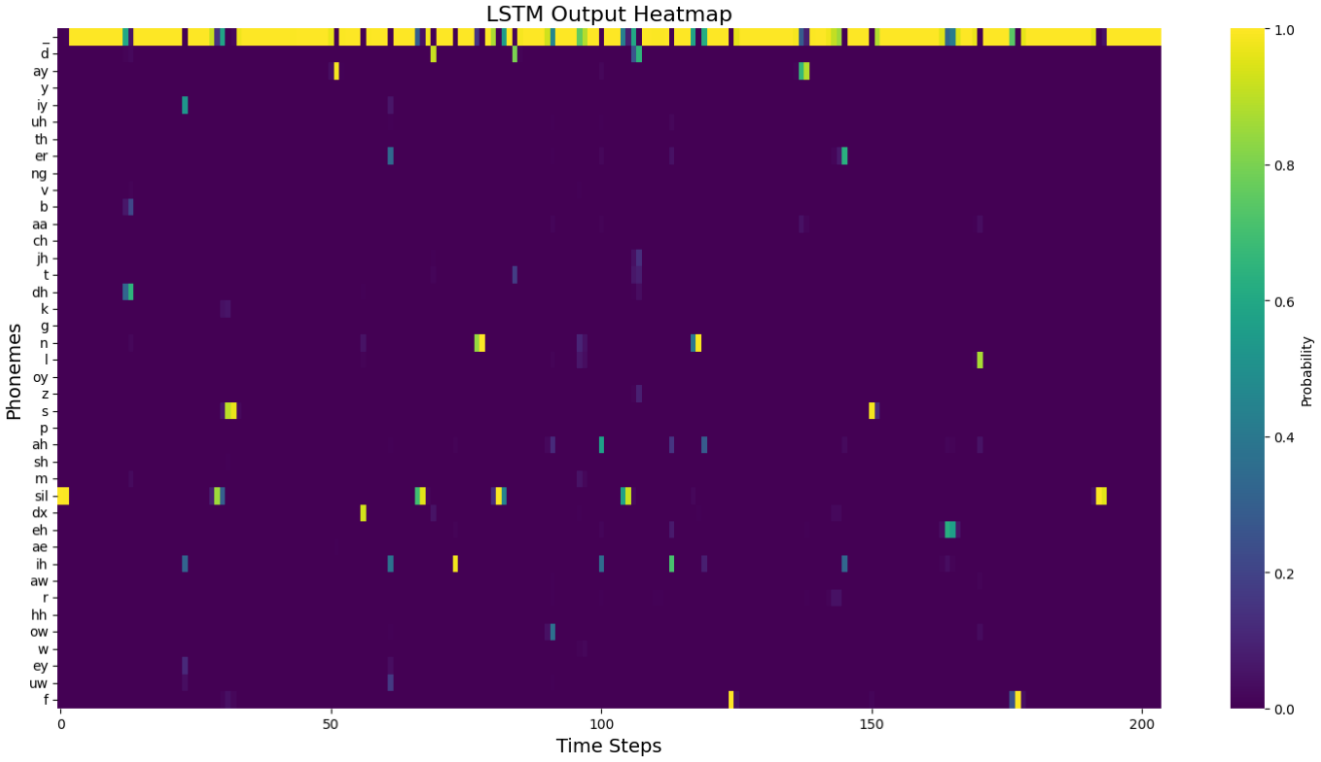


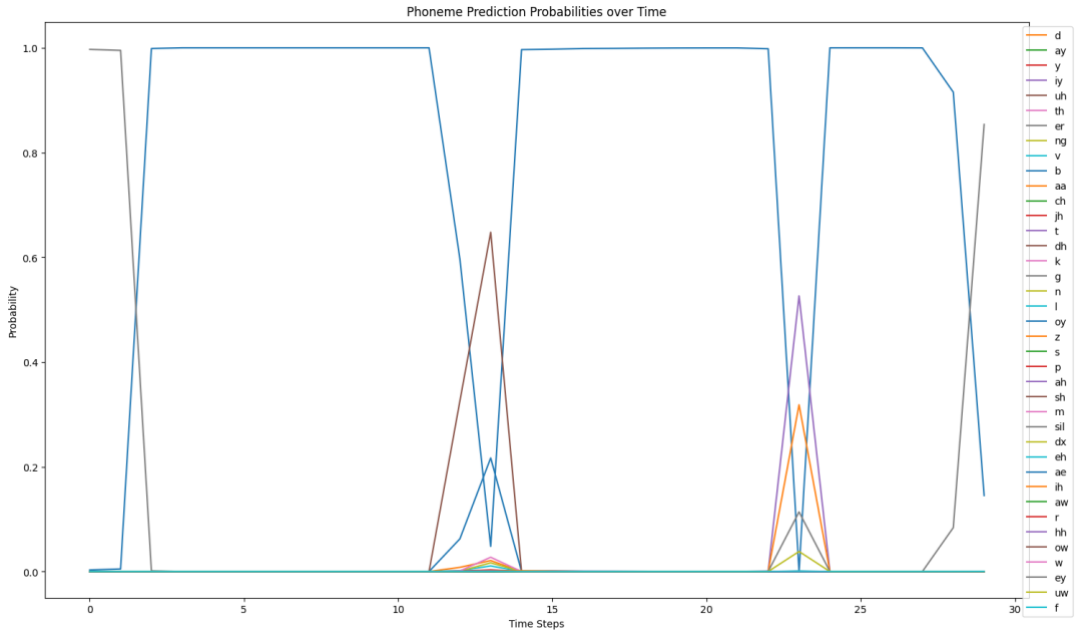
Figure 5.2 Heatmap of an utterance in test set - full steps

The correct output is:

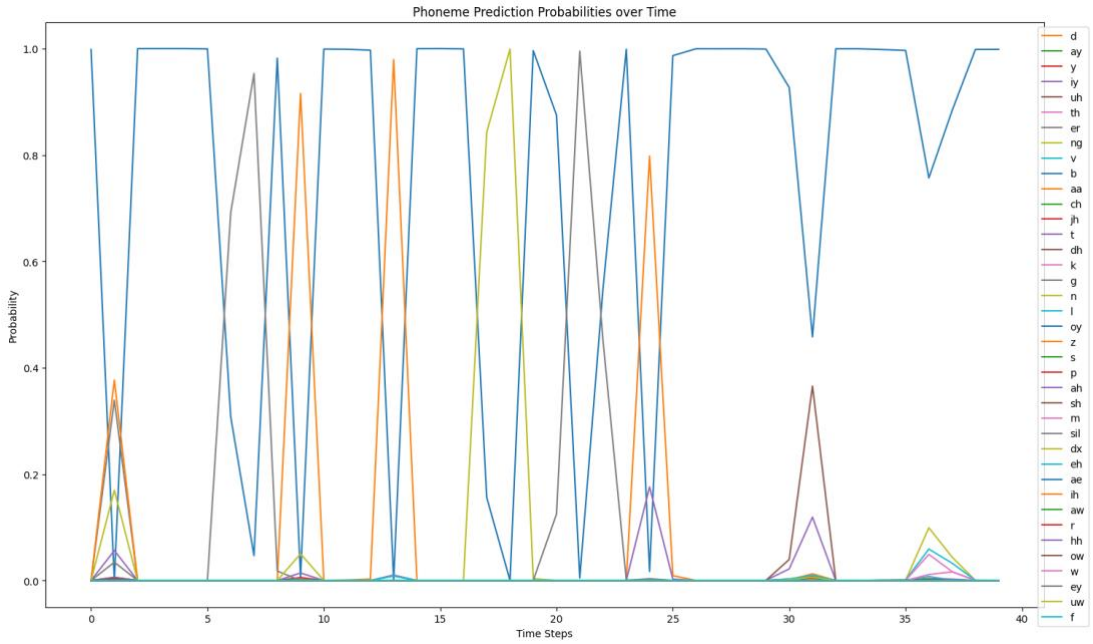
"sil b iy ih sil k s ay dx ih sil d ih n sil d ow n ah sil d eh n ah f ay er s eh l f sil."

By comparing the model's output probability predictions in the heatmap, it can be observed that the correct phoneme positions generally have higher probabilities, further validating the model's effectiveness.

Additionally, the predicted probability curve for each phoneme at each time step can be plotted to observe which phonemes have higher predicted probabilities at specific time steps. This helps to identify the model's decision process at different time steps. By selecting the phoneme with the highest probability at each time step, the predicted result can be obtained. The following are the visualization results of this method (from time steps 0 to 30 and from 60 to 100).



(a) 0~30 time steps



(a) 60~100 time steps

Figure 5.3 Probability for each phone at every step