Final Report for 11785

Yilang Liu
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213
yilangl@andrew.cmu.edu

Yao Lu
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213
yaolu2@andrew.cmu.edu

Wuzhou Zu
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213
wuzhouz@andrew.cmu.edu

Xupeng Shi
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213
xupengs@andrew.cmu.edu

Abstract

Subtitles feature is an important task for content creators and media providers. Our project is to use LSTM with Listen, Attend and Spell (LAS) to transform audio, video input into transcripts and determine who is saying, to enable a better application of deep learning and NLP for animation creater. Our primary contribution will be either labeling the speaker or segmenting the transcript to create visually-aware subtitles with better performance.The data sets are captured from Japanese anime "君の名は" Our development tools will be facilitated by open source solutions such as the NLP From Scratch tutorial by PyTorch. We will choose a dataset that is reasonable for our hardware requirements. We expect a model with 20000 parameters to take 30 minutes long to perform(training / validation) on one batch of data. Therefore, for the entire dataset, we expect (training / validation) to take 180 minutes. This is where the 1.2 $ estimate comes from, since (3h $\times$ 0.4 $/hour = 1.2 $). Our main resource package is PyTorch. The TA mentor for our project is Joseph Konan. The full code can be seen in our GitHub and YouTube:

https://github.com/YilangLiu/11785_Project.git

https://youtu.be/30i1nj9rFe8

1 Introduction

In recent decades, speech recognition has been drawing much attention in applications of deep learning.It is particularly common in media tasks(e.g. Movie subtitles feature, speaker transcript). Currently, the mainstream method in the media and file industry to generate speaker transcripts is labeling manually. However, labeling manually have several drawbacks: (1) It usually requires the person a huge amount of time to label a whole movie. (2) The process of labeling may have some mistakes due to personal carelessness. (3) The portability of labeling is weak, which means the operator has to do many repetitive tasks in one or multiple media. Our project, on the other hand, is aiming to provide better applications, like speech recognition, of NLP to media providers and animation creators, by training and validating an LSTM model with Listen, Attend and Spell (LAS) using cross validation. After training, the model can label the transcripts precise and fast. There is lots of similar research and application on it, but the main problem of us is to improve the efficiency and performance of the dialogue labeling and speaker distinguishing. with multimodal approaches. Before the training, we need to translate the input data to spectrograms, which helps avoid the noise and distortion in the input data, for a better model input for training.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

2 Literature Review

As of today, the accuracy and reliability of automatic speech recognition have improved a lot. Recurrent neural networks (RNN) has been playing an important role in this remarkable progress RNN is a kind of neural network that contains a hidden state which can link the input and previous state together. Based on Long short term Memory Networks (LSTM), a variant category of RNN, many automatic speech recognition models have been developed and proven to be accurate. Graves et al. introduced a hybrid model, HMM-Deep Bidirectional LSTM (DBLSTM) that improves frame level accuracy[1]. DBLSTM makes use of previous and future contexts by setting two separate hidden layers and processing the data in both directions. After proper training, this network shows promising results on TIMIT experiments.

In the work done by Liu, Chaojun, et al.[2], the extensive studies of speaker adaptation of LSTM-RNN models for speech recognition are performed. With different adaptation methods combined with KL-divergence based regularization, observations and analysis indicate that higher performance can be achieved over the LSTM baseline model. According to the authors, in a large vocabulary speech recognition task, by adapting only 2.5% of the model parameters using 50 utterances per speaker, we obtained 12.6% WERR on the dev set and 9.1% WERR on the evaluation set, over a strong LSTM baseline model.

The main goal of this study is to find a way to adapt the speaker-independent model effectively while keeping the number of speaker-specific parameters to minimal. There are two approaches investigated: adapting existing network components and adapting inserted affine transformation between layers. The first approach tends to adapt weight matrices inside the recurrent loop and the second one is to insert an affine transform on top of each LSTM layer. Both methods are effective but the first method is adapting the top layer projection matrix gives a large improvement of 9.1% WERR with only 50 unsupervised utterances.

A widely applied example of automatic speech recognition is the automatic captions feature on Youtube. In general, such a system works by first recognizing the speech in audio and matching it with an existed vocabulary as much as possible. Then, the generated text is modified according to the context to increase the closeness with original meanings[3]. We found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly[4], because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier. The crucial step is to transform the network outputs into a conditional probability distribution over label sequences. The network can then be used a classifier by selecting the most probable labelling for a given input sequence[5]. The project will utilize Long short term Memory Networks (LSTM), a variant category of RNN, as the core model.The architecture uses pyramidal LSTMs to reduce the spectral variation of the input feature, and then passes this to LSTM layers to perform temporal modeling, and finally outputs this to DNN layers, which produces a feature representation that is more easily separable[6]. The reason why we use the LSTM model to generate transcripts is that LSTM works better than standard RNNs for speech recognition and image captioning. Due to the property of this dataset which is continuous and lengthy, the LSTM(RNN) fits the task perfectly since the structure of RNN is continuous and RNNs effectively have an internal memory that allows the previous inputs to affect the subsequent predictions. It's much easier to predict the next word in a sentence with more accuracy, if the previous words are known.

In work Done by William Chan, Navdeep Jaitly, Quoc V. Le and Oriol Vinyals, who contribute to LAS, LAS is based on the sequence to sequence framework with a pyramid structure in the encoder that reduces the number of timesteps that the decoder has to attend to. LAS is trained end-to-end and has two main components. The first component, the listener, is a pyramidal acoustic RNN encoder that transforms the input sequence into a high level feature representation. The second component, the speller, is an RNN decoder that attends to the high level features and spells out the transcript one character at a time[7]. As mentioned, the ELTSM has achieved up to 30% increase in the labeled attachment score (LAS) as compared to LSTM and GRU in the dependency parsing (DP) task. the models also outperform other state-of-the-art models such as bi-attention and convolutional sequence to sequence (convseq2seq) by close to 10% in the LAS[8].

3 Project Overview

3.1 Problem Address

In order to enable content creators and media providers to create better NLP applications for animated film, we selected this project to create and train a LSTM model focused on NLP and Computer Vision in anime Kimi no Na Wa.[9] [4] Our main job in the project is to generate a labeled transcript based on the given anime video as the dataset. In the transcript, we first need to translate the dialogue between characters. There is lots of similar research and application on it, but the main problem of us is to improve the efficiency and performance of the translation with multimodal approaches. We also need to distinguish the speakers of the dialogue. During the project, we will show our process of establishing and training our neural networks. In the end, we will report our results for reproducible research. In the process of our project, we have to overcome many engineering problems with putting forward the end-to-end deeping learning solutions.

3.2 Contribution

The team created a model that take int the inputs which is transformed spectrogram of the audio recording and produce a transcript of the words and person who speaks the words as the output. The overall process is shown in Figure 1.

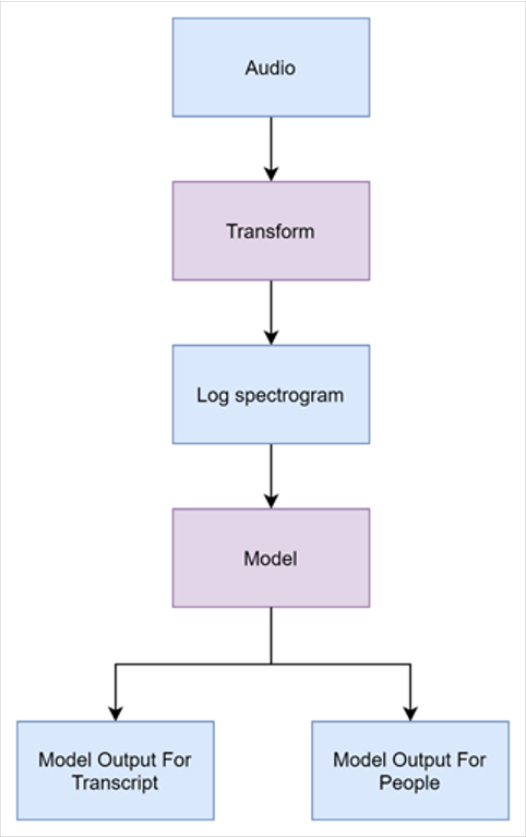


Figure 1: Forward propagation

To produce the transcript of a video from the spectrogram, the team decides to utilize a model structure of the LAS(Listen, Attention, Spell) as the base of the solution. The team made this decision because transcripts are consisted with sentences with different length and attention is a technique that mimics the human attentions, and with the effect of attention, it is possible to enhance the important parts of the input data and ignore the rest of the irrelevant information. With this mechanism, the team believes that the model will be able to produce filtered and more accurate result.

The model is composed by three main blocks: Encoder(Listening), Attention, Decoder(Spelling). The encoder is composed by a layer of normal bidirectional LSTM and three layers of pyramidal LSTM. After the pyramidal LSTM, the results will be sent to two linear layers and out put the keys and values for later use in decoder and attention. The key and value will then be sent into decoder. In decoder, the values will be sent into two layers of LSTM cell to produce the output which is also the query for attention calculation, the query, key and value are then sent to attention to calculate the importance. In attention, the query and key are compared to determine the importance over the entire time sequence and create a mask of it, the value will be multiplied by this mask element wisely. The masked value will concatenate with the output from the LSTM cell which is also the query and be sent into a linear layer to produce output vector which consists with the probability for different character. The neural network model is also designed to output the name of the speaker, it is realized by adding a linear layer after the output of the encoder. So the overall structure diagram is shown in Figure 2. To improve the result, the team also utilizes some techniques to boost the performance. The

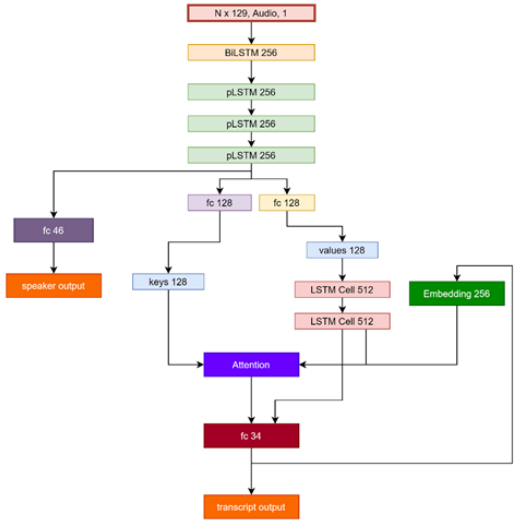


Figure 2: Sequence modeling

team utilizes weight tying between the embedding layer of the decoder that enrich the dimension of the character and the final linear layer that produce the probability vector. The team also utilizes variating teacher forcing rate: the teacher forcing rate is set very high at the beginning of the training and it gradually decreases as the model continues to learn. The team also utilizes locked dropout technique on the output from the pyramidal LSTM layer to prevent overfitting. For both outputs, the team utilizes cross-entropy cost function to produce loss for back propagation. The two losses are added together after calculation.

3.3 Dataset

Two datasets are used for training our model. The first dataset is from HW4P2 which is the audio recording from people's readings. The second dataset is from the audio recordings from a Japanese movie called "Kimi no Na Wa". The corresponding transcripts are generated along with the recorded waveform. However, raw audio recordings contain noises and distortion which are not suitable for training. We need to process the train and transform useful information into a melspectrogram. To dynamically process the audio recordings to melspectrogram, we use Short-Time Fourier Transform (STFT) to calculate power spectrum Instead of focusing on a whole lengthy recording waveform, we process the data on small, overlapping segments and store it in concatenated vectors.

3.3.1 HW4P2 Dataset

In our HW4P2 dataset, the information for each timestep is stored in 40-dimensional vectors whereas the Kimi No Na Wa dataset is stored in 129-dimensional vectors. The following table contains the information about those two datasets. The HW4P2 dataset is shown in Table 1. It is equally

split into three parts for training, cross-validation, and testing. Notice that each list entry includes a fully processed audio utterance with shape (audio length, feature length=40). Figure 3 shows the visualization of data structure copied from HW1P2.

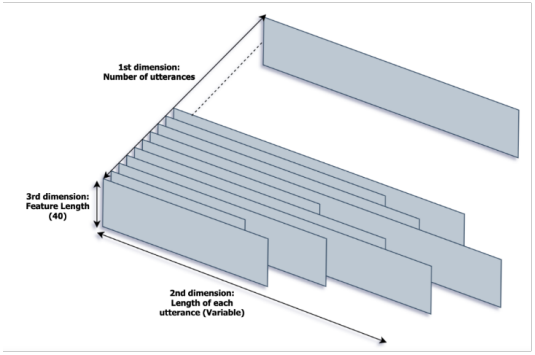


Figure 3: Dataset Structure for HW4P2

Table 1: Statistics of HW4P2 dataset

File Name	Size	Type	Shape
train.npy	237.85 MB	List object	(28539,)
dev.npy	237.17 MB	List object	(2703,)
test.npy	237.85 MB	List object	(2620,)
train transcripts.npy	12.53 MB	List object	(28539,)
dev transcripts.npy	783.54 KB	List object	(2703,)

To make sure data will be trained in PyTorch, we need to construct a PyTorch version dataset by using two primitives: torch.utils.data.DataLoader and torch.utils.data.Dataset. This allows us to use pre-loaded datasets as well as our own data. Below Figure 4 shows one batch from the train.npy and train-transcripts.npy.

```
[[1.5781574e-04 4.6676921e-04 6.0391880e-04 ... 8.4340958e-05
  7.8386125e-05 5.9076814e-05]
 [3.9359243e-06 8.8821416e-06 6.0467682e-05 ... 1.1556587e-05
  1.5931026e-05 1.9300111e-05]
 [4.3008840e-06 4.1246236e-05 1.8424325e-04 ... 1.7501028e-05
  1.5845562e-05 7.0566102e-06]
 ...
 [2.1227181e-05 9.1362483e-05 2.2255529e-04 ... 1.4848149e-03
  2.7626174e-04 9.9138590e-05]
 [1.9874391e-05 1.1987342e-04 1.4569092e-04 ... 2.0866373e-03
  1.5945223e-04 5.8265438e-04]
 [1.3983450e-04 2.9113656e-04 6.1676337e-04 ... 2.3235427e-03
  5.4949796e-04 1.7151469e-03]]
[b'he' b'had' b'never' b'before' b'come' b'in' b'contact' b'with' b'such'
 b'an' b'agreeable' b'lot' b'of' b'companions' b'and' b'every' b'hour'
 b'of' b'the' b'day' b'he' b'tried' b'to' b'prove' b'himself' b'grateful'
 b'still' b'he' b'did' b'not' b'mention' b'a' b'word' b'about' b'what'
 b'he' b'might' b'possibly' b'know' b'of' b'the' b'dastardly' b'deed']
```

Figure 4: One batch from HW4P2 dataset

Noticing that the transcripts are English characters with bytes encoding, so we have to transform the characters into indexes in order to put data into a data loader. We find all possible characters in transcripts and stored in one letter list shown in Figure 5 which will be used to create transition dictionaries.

The position index of each character will be used to transfer character to index and vice versa. We also add the start of sentence symbol encoded with "<sos>" and end of sentence symbol encoded with "<eos>" to signal the start and stop of the model prediction. Here is an example of the first processed characters compared with original transcripts shown in Figure 6. In our data, b stands for bytes.

After converting characters into indexes, we are now able to put the processed transcripts and spectrogram in PyTorch dataset. Since we have variable length input sequences, we need to pad the sequences to maximum length in order to train and test them on LSTM. The PyTorch function offers

an easy way to pad input sequences with a maximum length which is torch.nn.utils.rnn.pad_sequence. We modify the collate function applying padding operation and return padded data with its length. Then, those data will be loaded to the data loader where the data will be converted to batches for further training. In HW4P2 data loader, we make batch size equal to 32 with shuffling enabled for train loader and validation loader. The number of workers in the data loader is 2.

3.3.2 Kimi No Na Wa (KNNW) Dataset

In the Kimi No Na Wa dataset, the original audio soundtrack is downloaded. Then, we use STFT to process the data and concatenate all recordings into one 2-D matrix with shape (feature dimension=129, utterance length). The transcripts and speakers are captured by reading English subtitles with its start time and stop time. In our data, the total duration of the soundtrack is 6396010 ms and we sample the soundtrack into 1370493 frames. Each frame contains audio information stored in 129 feature dimensions. Table 2 shows the information about the dataset after processing. In our project, we decided to split the data into 80% training and 20% validation dataset with shuffle enabled.

Table 2: Statistics of KNNW dataset

File Name	Size	Type	Shape
knnw_en_sub+speakers.srt	112 KB	Subtitle File	_
knnw_en_sub.csv	78 KB	CSV file	(1393,4)
log_spectrogram.npy	674 MB	numpy array	(129,1370493)

In knnw_en_sub+speakers.srt, we have four types of information which are start time, end time, subtitle, and speakers. We extract start time, end time, and subtitles in the knnw_en_sub.csv file where the first column stores the number counting of data rows. We use pandas to read the CSV and store it in DataFrame. Here is an example of the first entry of the English subtitles shown in Figure 7.

However, after counting the possible character size that the model needs to predict, we find the vocabulary size in Kimi No Na Wa transcripts is 83 which contains the capital, lower case of the letter as well as special characters. In this case, solely based on the audio inputs, the model will not have precision predictions since special characters do not have specific pronunciation. We have to shrink the size of the vocabulary and modify the transcripts for better model predictions. First, we transfer all capital letters into lowercase letters using the .lower() command while retaining other special characters. Then, we create a symbol list that we want to remove from the transcript. Here is the symbol list which contains Arabic numbers and special characters. Those letters are not shown in the actual spectrogram data.

We remove more than half of the characters, the resultant vocabulary size equals to 34 which is the same as the HW4P2 dataset. Finally, we convert the character to indexes using the position indexes of the letter list. Below Figure 9 is the sample of the transformation:

First and the last letter represents the start of the sentence and the end of the sentence symbol. Then, the spectrogram and its transcripts are passed into the dataset for alignment. We use the start and end time values to help us find the corresponding spectrogram data. The relationship between the start time and the frame is shown below:

$$starttimeindex = floor(starttime * \frac{totaltimeduration}{totalframes}) \tag{1}$$

$$stoptimeindex = ceil(stoptime * \frac{totaltimeduration}{totalframes}) \tag{2}$$

$$audiosegment = spectrogram[starttimeindex, stoptimeindex] \tag{3}$$

The floor of the scalar x returns the largest integer i, such that i <= x. The ceil of the scalar x is the smallest integer i, such that i >= x. For speaker data, we also find that the number of speakers is 98.

```
LETTER_LIST = ['<sos>', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', \
        'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '-', "'", '.', '_', '+', ' ', '<eos>']
```

Figure 5: Letter list for HW4P2 dataset

Here is the output of the speakers for the first four frames. We can see that some utterances contain more than one speaker and also include some speakers who only show once as shown in Figure 10. This will also significantly affect the ability of the model to make correct predictions. We need to narrow the scope of the speaker type in order to have better performance. One of the ways to shrink the size is to group multiple speakers into one single label. In our dataset, for the example shown above, 'Taki_Mitsuha' and 'Mitsuha+Taki' can be grouped into a 'multiple' label. By doing this operation, we are able to shrink the speaker size down to 46 distinct speakers. Then we tokenize the data by converting speaker data into Arabic number as shown in Figure 11. We use the row index of the transcript to find the corresponding speakers. After processing the speaker data, the data will be passed in a built-in dataset where the data will be padded and transferred to the data loader. The configuration of the data loader is the same as the WSJ dataset. After two datasets are configured, we can pass them to the model for training. We will descript the model in detail.

3.4 Model Detail

3.4.1 Transfer Learning

As we can see from the size of the dataset above, the homework dataset contains much more information than the Kimi No Na Wa (KNNW) dataset. A small dataset also causes the underfitting of the data. So, instead of directly training the Kimi No Na Wa dataset from scratch, we first train the model with the homework dataset and retain the model parameters for KNNW training. Luckily, we can use torch.load() to copy homework parameters to KNNW model. The only caveat is that we need to modify the linear layer in decoder and encoder to predict the vocabulary size we want. Then, we start training with copied model parameters and make prediction.

3.4.2 Working Baseline

To have a more accurate prediction, we use LAS model for training. LAS model contains three major parts: Listener, Attention and Speller. Generally it is a sequence to sequence model. At each timestep listener part takes audio sequences as input and encode the data sequence into high dimensional feature vectors. We use three layer pyramidal BLSTM to process the data sequences and store the output from the final layer. Notice that we need to reshape the input data from (batch-size, length, dim) to (batch-size, length/2, dim*2) for pyramidal BLSTM. The output from encoder will be processed through two independent linear layers to produce key and value for attention. Moreover, we add another linear layer to train the speakers based on the transcript. We use greedy search to find the speakers with highest probability. As a result, listener returns four parts: key, value, encoder length, name prediction. In speller, we use attention mechanism where we generate query of the hidden states from encoder. The attention mechanism will be learned to only focus on the hidden state with highest correlation and it will generate relevant context. Speller part will take previous context and current embedding as input to make more informed prediction. Our baseline model in detail is shown in Figure 12. In baseline model, we don't add any augmentation techniques such as locked dropout and weight tying. In our results section, we will compare the performance of baseline model with final model.

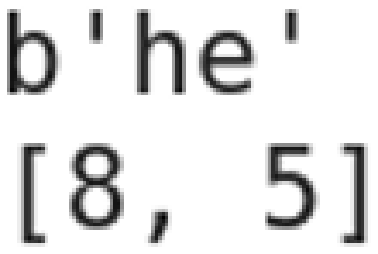


Figure 6: Example of character processing

Number	1
Start time in milliseconds	1650
End time in milliseconds	10800
Text	TOHO CORPORATION

Figure 7: First entry of the English Subtitles

```
SYMBOL_LIST=["!","?",":",'',",","0","1","2","3","4","5",
    "6","7","8","9","[","]",'/','=','é',"(",")",'“','"','”']
```

Figure 8: Sybmbol List for KNNW dataset

```
before transform:
TOHO CORPORATION
after transoform:
[ 0 20 15  8 15 32  3 15 18 16 15 18  1 20  9 15 14 33]
```

Figure 9: Sample character transform for KNNW dataset

```
'Mitsuha', 'Taki', 'Taki+Mitsuha', 'Mitsuha+Taki',
```

Figure 10: Original speakers data from KNNW dataset

```
'Mitsuha', 'Taki', 'multiple',
```

Figure 11: Speakers data after shrinking size

```
Seq2Seq(
  (encoder): Encoder(
    (lstm_new): LSTM(129, 256, batch_first=True, bidirectional=True)
    (pBLSTMs): Sequential(
      (0): pBLSTM(
        (blstm): LSTM(1024, 256, batch_first=True, bidirectional=True)
      )
      (1): pBLSTM(
        (blstm): LSTM(1024, 256, batch_first=True, bidirectional=True)
      )
      (2): pBLSTM(
        (blstm): LSTM(1024, 256, batch_first=True, bidirectional=True)
      )
    )
    (key_network): Linear(in_features=512, out_features=128, bias=True)
    (value_network): Linear(in_features=512, out_features=128, bias=True)
    (linear): Sequential(
      (0): Linear(in_features=512, out_features=46, bias=True)
      (1): ReLU()
    )
  )
  (decoder): Decoder(
    (embedding): Embedding(34, 256, padding_idx=33)
    (lstm1): LSTMCell(384, 512)
    (lstm2): LSTMCell(512, 128)
    (attention): Attention()
    (character_prob): Linear(in_features=256, out_features=34, bias=True)
  )
)
```

Figure 12: Baseline model overview

3.4.3 Augmentation Techniques

Variable teacher forcing: The team utilizes a variable teacher forcing rate in the decoder part during the training. Teacher forcing is quite essential to this model since the decoder produces predictions based on the previous prediction, at early stage of training, the model is most likely to produce all wrong predictions and un capable of training. Teacher forcing helps this situation by revealing the ground truth as the base for next prediction. The team starts with teacher forcing rate of 0.9, it means there is 90% chance a ground truth character will be sent instead of a prediction will be sent into the prediction for the next character. To extent, the team improves the teacher forcing by making it dynamically changing during the training. The teacher forcing that is utilized in the current model is capable of changing based on the validation distance. It means that, as the validation distance decreases during training, the teacher forcing rate also decreases to allow more previous predictions to pass into the current prediction to make the prediction more challenging and allow further improvement on performance.

Weight tying: the team also utilizes the weight tying between the embedding layer in decoder and output layer of decoder. The embedding layer parameter has a shape of (34, 256) and the output layer of decoder has a shape of (256, 34). These two layers basically are reversing processes of each other, so the team utilizes weight tying on the corresponding parameters of these two layers to make better convergence.

3.5 Loss Calculation

In our model, we decide to use cross entropy loss for this model. We want the distribution P(model) learned by the model on the training data to be as close as possible to the distribution P(real) of the real data, so that we can minimize its relative entropy. However, we do not have the distribution of the real data, so we can only hope that the distribution P(model) learned by the model and the distribution P(train) of the training data are as similar as possible. Assuming that the training data are sampled independently from the overall same distribution, then we can reduce the generalization error of the model by minimizing the empirical error of the training data. The cross entropy for character embedding is given by:

$$CE = \sum_{x \in X} P(x) log \frac{1}{Q(x)} = - \sum_{x \in X} P(x) log Q(x) \tag{4}$$

In our dataset, we use softmax to normalize the network and use this cross entropy to calculate the loss for model training:

$$L(\hat{y}, y) = CE(\hat{y}, y) = - \sum_{i=1}^{K} y_i log \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{5}$$

The gradient of cross entropy loss is given by:

$$\frac{\partial L(\hat{y}, y)}{\partial x_j} = \hat{y_j} - y_j \tag{6}$$

3.6 Evaluation Metric

After obtaining the prediction, we use greedy search to generate predicted text. The greedy search only picks labels with highest probabilities. In order to better understand the difference between two character sequence, we use Levenshtein distance. Levenshtein Distance is a string metric that measures the difference between two character sequences. The Levenshtein Distance between two words is the minimum number of single-character edits (insertions, deletions, or substitutions) required to convert one word to another. The calculation of Levenshtein distance is shown in Figure 13.

Where a and b are two different strings. Tail of x means a string of all but the first character of x. X[n] is the nth character of the string x starting with character 0. In our model, the Levenshtein distance suits our purpose. Those the two strings are from model predictions and transcript labels. For each training epoch we will take average for every batch and store the running average distance for visualization. The performance of the model is calculated by how much distance it reduces after

training. The plot of the distance for various optimizer is shown in the results part. We only show the first several predictions due to limitation of the page size. Some of the sequences are too large to displayed in this report. In Figure 141516 Here is the training output from model with highest to lowest Levenshtein distance:

4 Results

The baseline model chosen for comparison is the original LAS model, which was introduced by Chan, William, et al [7].

4.1 Baseline Model

4.1.1 Loss and Distance for Baseline Model

For the baseline model, the loss of Adam optimizer with learning rate 0.001 is the most ideal as it reaches the lowest cross entropy loss value, below 2.5 after twenty epochs. For the training and validation distances, they are much lower than those of 0.1 and 0.01 learning rates. The validation distance reduces a lot down to around 200 and stabilizes after 25 epochs. For the AdamW optimizer, 0.001 is also the most ideal learning rate as it reaches the lowest loss, 2.2, after twenty epochs. The validation distance is also the lowest among the three. The validation distance stabilizes to around 150 after 20 epochs.

However, it is different for SGD optimizer. With learning rate 0.1, this optimizer performs best in terms of the training loss. However, there is no outstanding difference noticed in training and validation distances during training.

4.1.2 Word Accuracy for Baseline Model

The similar result is also seen in word accuracy vs. epochs graph. For the Adam optimizer, the training and validation accuracies increase only when the learning rate is 0.001 but fluctuates around 19% after initial epochs. This pattern is also seen in the AdamW optimizer. We only see an increasing trend for the learning rate of 0.001. However, the accuracies are lower than the other two learning rates. With SGD optimizer, the accuracies are highest when the learning rate of 0.1. When it reaches the around the twentieth epoch, the accuracy achieves up to 32.5%, which is higher than the results of other two optimizers.

4.2 Proposed Model

4.2.1 Loss and Distance for Proposed Model

In order to explore the differences between different optimizers and learning rates, we ran the same model with different optimizers and learning rate settings and record the corresponding loss values. We first applied Adam optimizer with the learning rate of 0.1, the training loss decreases dramatically as the number of epochs increases. However, there is no noticeable decrease in training and validation distances except for one outlier. After decreasing the learning rate to 0.01, the training loss of the initial epochs is much lower and there shows a clear trend on decrease in training distance. However, the validation distance is still unstable and unable to reach convergence. After changing the learning

$$\operatorname{lev}(a,b) = \begin{cases} |a| & \text{if } |b| = 0, \\ |b| & \text{if } |a| = 0, \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) & \text{if } a[0] = b[0] \\ 1 + \min \begin{cases} \operatorname{lev}(\operatorname{tail}(a), b) \\ \operatorname{lev}(a, \operatorname{tail}(b)) \\ \operatorname{lev}(\operatorname{tail}(a), \operatorname{tail}(b)) \end{cases} & \text{otherwise.} \end{cases}$$

Figure 13: Levenshtein Distance Pseudo Code

rate to 0.001, the training loss starts very low and no dramatical changes happen in 50 epochs and the training distance keeps decreasing until the twentieth epoch when it becomes steady around 50. However, the validation distance fluctuates dramatically in the first twenty epochs and settles down to 150 after the twentieth epoch.

AdamW optimizer is also applied. With learning rate of 0.1, the result is very likely to what we observed in Adam optimizer with the same learning rate. After changing the learning rate to 0.01, the initial training loss also decreases a lot and the clear trend emerges for the training distance. Except for the clear outlier, the validation distance only takes one step to decrease from 330 to 250. After keeping decreasing the learning rate to 0.001, the performance of model improves a lot. While training loss does not have dramatic changes as it also starts very low, the training distance decreases down from 150 to 50 and the validation distance stabilizes to 150 after twenty epochs.

Another type of optimizer, SGD is also tested. With learning rate of 0.1, despite of the training distance, there is not any noticeable difference in training loss and validation distance during the whole 50 epochs. After decreasing the learning rate to 0.01, there is a huge outlier in the first few epochs. However, there is no significant trend noticed. With learning rate decreased to 0.001, still not any changes emerge on the decreasing trend on training loss, training distance, and validation distance.

4.2.2 Word Accuracy for Proposed Model

The word accuracies corresponding to different optimizers and learning rates are also recorded. For the Adam optimizer, only with a learning rate of 0.001, both training and validation accuracy of speaker prediction increases. With AdamW as the optimizer and 0.001 as the learning rate, the validation accuracy does not improve during the 50 epochs. For the SGD optimizer and 0.01 learning rate, the validation accuracy fluctuates frequently for the beginning 20 epochs. Decreasing the learning rate down to 0.001, both training and validation accuracies increase to 25% immediately but stabilize around that accuracy.

4.2.3 Attention Plots

As the attention plot indicates, though it is changing, the model is not able to produce a accurate attention which should be a diagonal line. The team also compare the proposed model to the plain LAS model. The results from the plain LAS model is worse, it indicates that the improvements(locked dropout, changing teacher forcing rate, weight tying) on model actually helps the learning. And both of the models achieve fair results after the training on hw4p2 dataset. The attention plot is shown in Figure 17. In the other word, the model is able to learn, but not able to produce a good result on this current dataset.

5 Conclusion

In short, for the closest transcript prediction and generation, optimizer AdamW with learning rate 0.001 has the best performance as it has the lowest distance, 150 after 20 epochs. On the other hand, for predicting speaker name, optimizer Adam with learning rate 0.001 has the best performance as it has 27% accuracy for the validation set.

Speaker label prediction is ideal when predicting the speaker who spoke the most of the sentences in the whole dataset, such as Taki and Mitsuha, but not robust enough to correctly predict the speakers who only spoke a few sentences. This is not ideal but reasonable.

Overall, The results are not good according to the output of the transcript, the model seems to be unable to learn the connection between the log spectrogram of the audio to the corresponding subtitles.

```
training_Pred:
<sos>- whill have toe semor she saote <sos>i wan mindaof ofsessed.<sos>incaf<sos>i widn
training_Target:
<sos>i still have the memos she wrote.<sos>i was kind-of obsessed.<sos>a caf<sos>i did
```

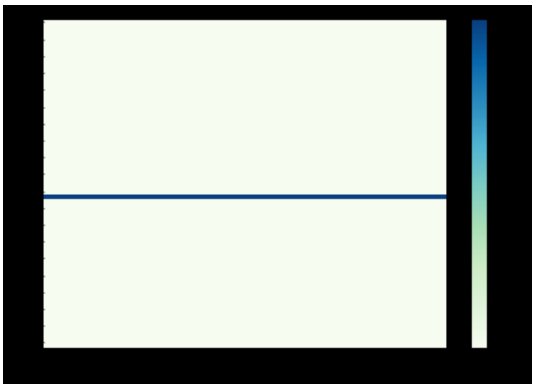
Figure 14: Sample Training Results with Highest Levenshtein Distance

```
training_Pred:
<sos>- wey mirh - uh.yes
training_Target:
<sos>- hey you. - uh yes
```

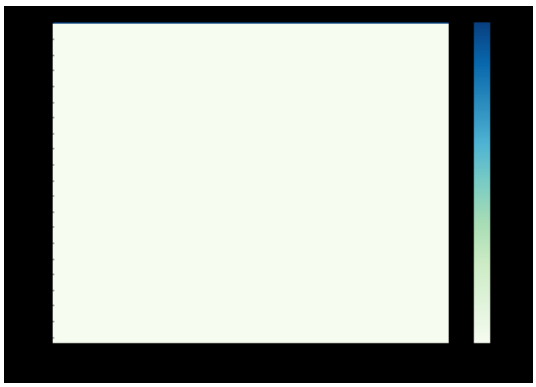
Figure 15: Sample Training Results with Medium Levenshtein Distance

```
training_Pred:
<sos>-ven the cucond hour hands of the toock.
training_Target:
<sos>even the second hour hands of the clock.
```

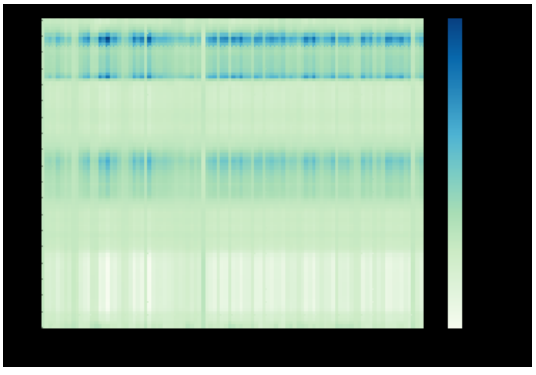
Figure 16: Sample Training Results with Smallest Levenshtein Distance



(a) Attention Plot at Epoch 10



(b) Attention Plot at Epoch 25



(c) Attention Plot at Epoch 50

Figure 17: Attention Plots

The failure of the models on this dataset may be caused by several reasons: the current dataset is too different from the training dataset which cause the transfer learning not ideal; the current dataset is too small, make the model overfit before actually learns the connection between log spectrograms and subtitles.

In the future, we can improve the model in several ways. First, we can make a larger input dataset to train. As we mentioned before, the current dataset is too small to train, which cannot train the model perfectly. Second, we can make some improvements to our model structure to suit the task more efficiently. We use LSTM with LAS now, but there are lots of parameters, like teacher forcing rate, gumble noise,dropout rate and layers of the LSTM, we can adjust. With more data and capability of the model, we can definitely improve our model's performance. Third, we can try some more structure, like CNN-MaxPooling, to do the LAS task to get the features of the input, in which we implement pyramidal bLSTM currently.

References

[1] A. Graves, N. Jaitly, and A. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278, 2013.

[2] Chaojun Liu, Yongqiang Wang, Kshitiz Kumar, and Yifan Gong. Investigations on speaker adaptation of lstm rnn models for speech recognition. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5020–5024, 2016.

[3] Anthony Romero. How does automated closed captioning work?, Nov 2018.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[5] Alex Graves, Santiago Fernández, and Faustino Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *In Proceedings of the International Conference on Machine Learning, ICML 2006*, pages 369–376, 2006.

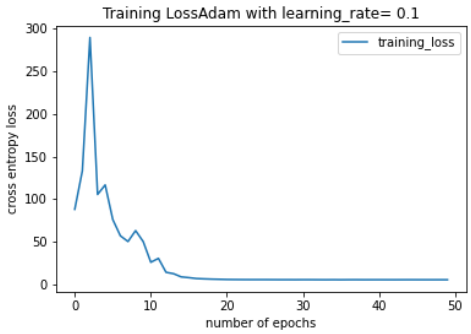
[6] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584, 2015.

[7] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell, 2015.

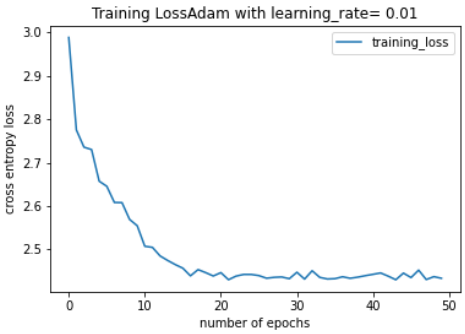
[8] Yuanhang Su and C.-C. Jay Kuo. On extended long short-term memory and dependent bidirectional recurrent neural network. *Neurocomputing*, 356:151–161, Sep 2019.

[9] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks, 2019.

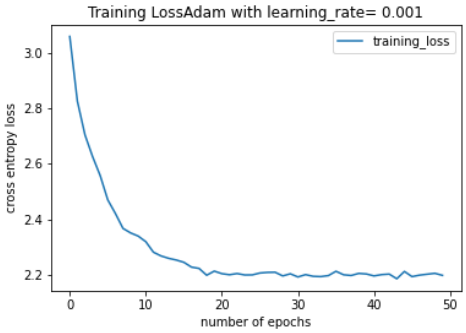
Appendix A Baseline and Proposed Model Plots



(a) Adam trial 1

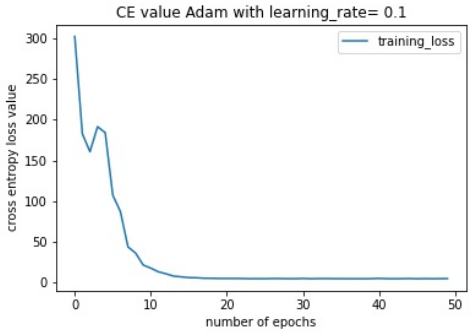


(b) Adam trial 2

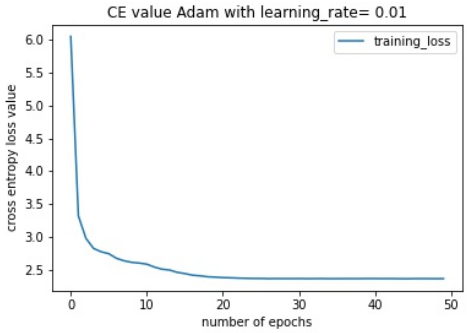


(c) Adam trial 3

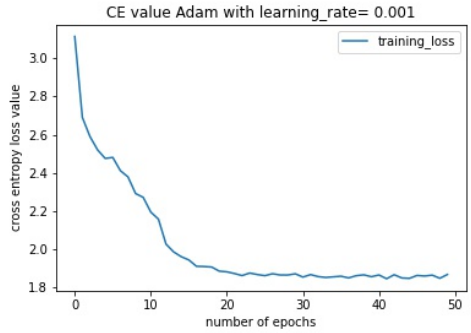
Figure 18: Loss Plots of Baseline Model with Adam



(a) Adam trial 1

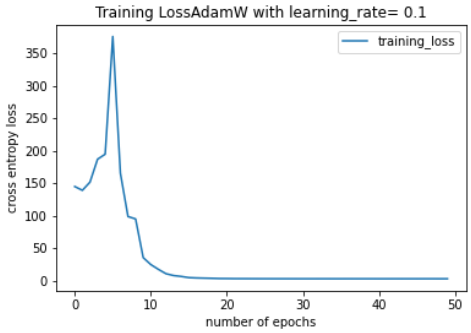


(b) Adam trial 2

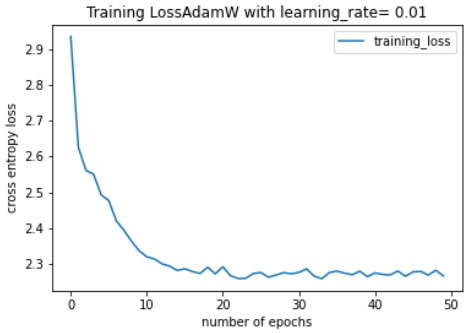


(c) Adam trial 3

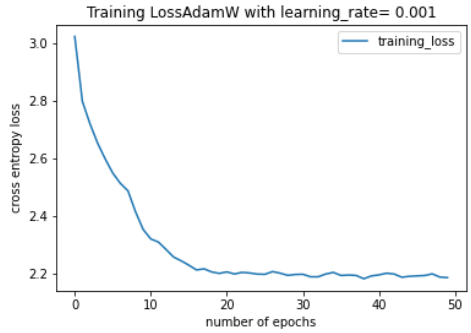
Figure 19: Loss Plots of Proposed Model with Adam



(a) AdamW trial 1

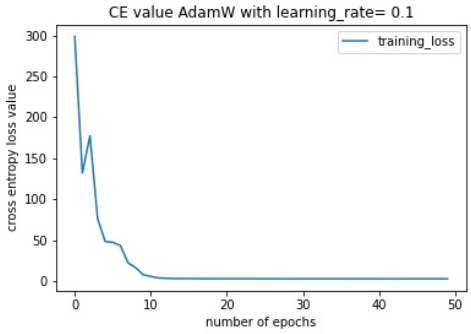


(b) AdamW trial 2

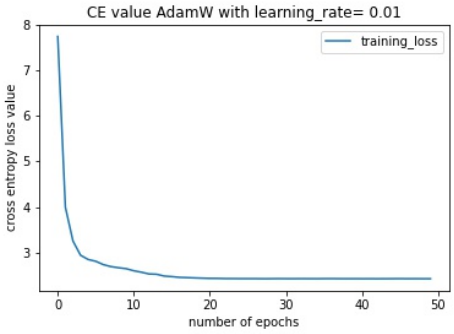


(c) AdamW trial 3

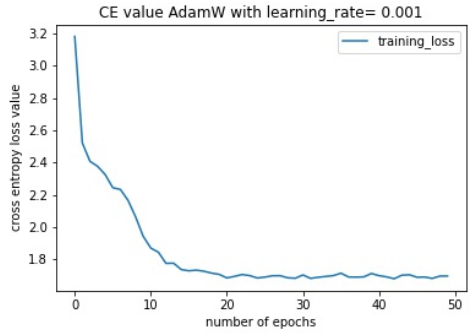
Figure 20: Loss Plots of Baseline Model with AdamW



(a) AdamW trial 1

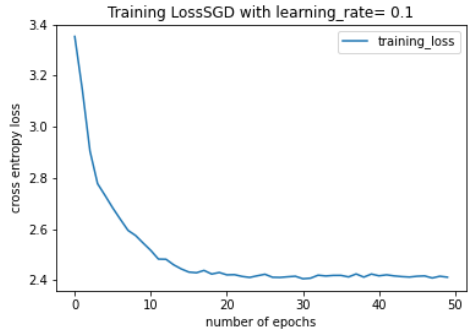


(b) AdamW trial 2

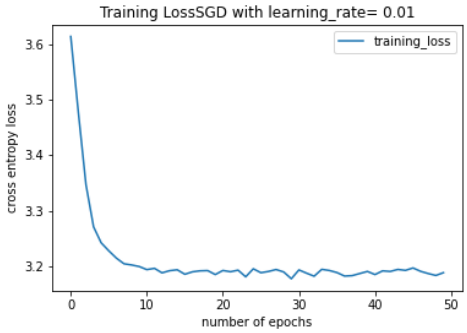


(c) AdamW trial 3

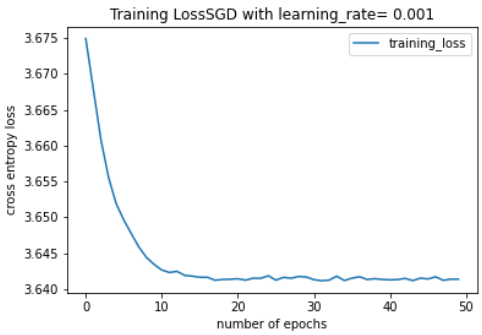
Figure 21: Loss Plots of Proposed Model with AdamW



(a) SGD trial 1

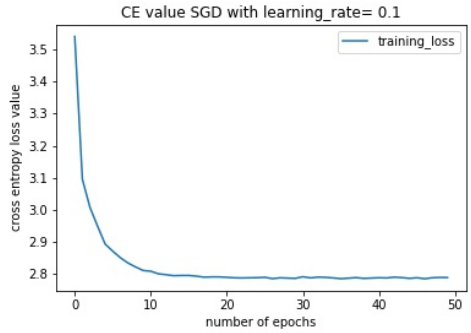


(b) SGD trial 2

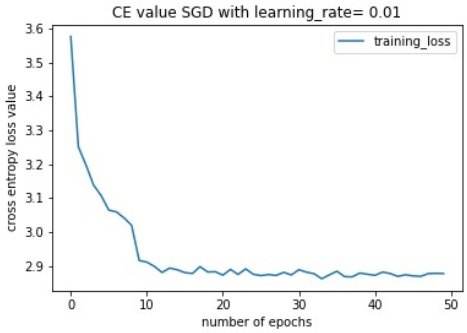


(c) SGD trial 3

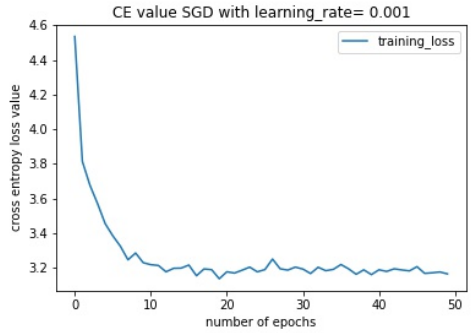
Figure 22: Loss Plots of Baseline Model with SGD



(a) SGD trial 1

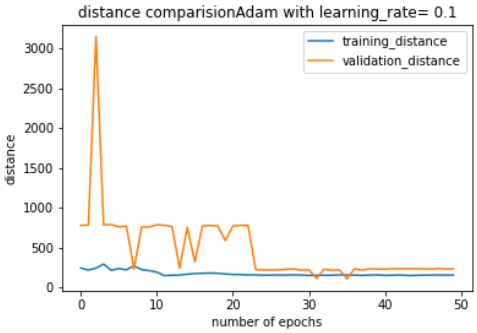


(b) SGD trial 2

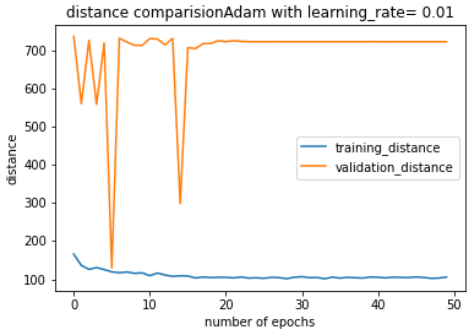


(c) SGD trial 3

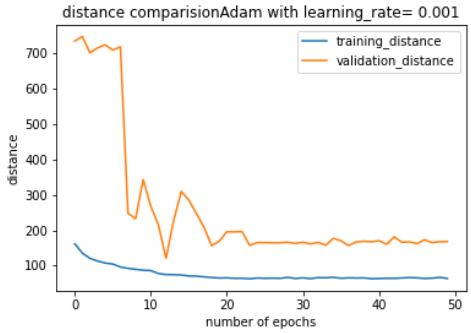
Figure 23: Loss Plots of Proposed Model with SGD



(a) Adam trial 1

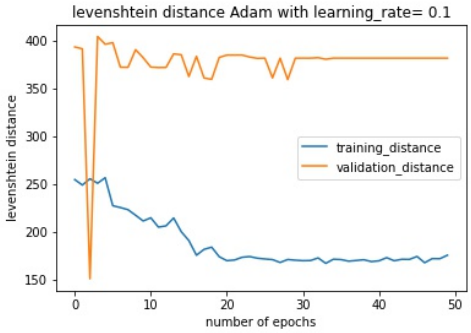


(b) Adam trial 2

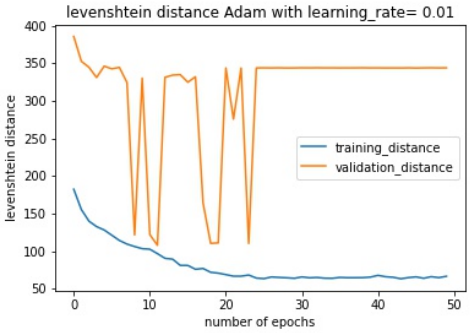


(c) Adam trial 3

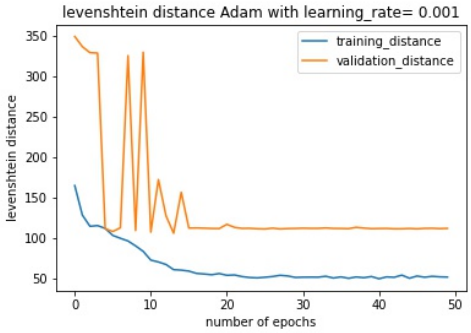
Figure 24: Distance Plots of Baseline Model with Adam



(a) Adam trial 1

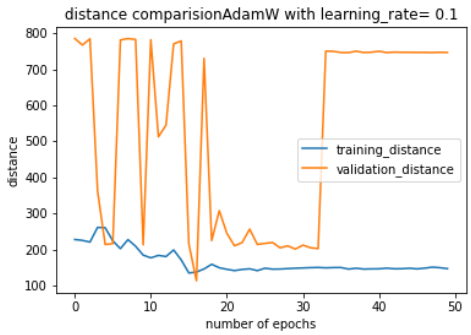


(b) Adam trial 2

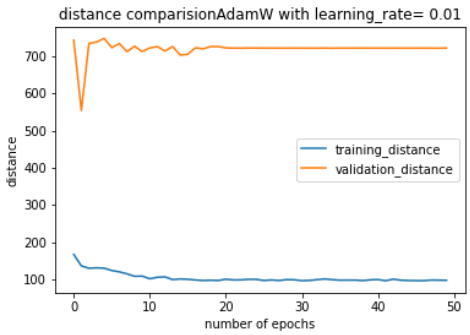


(c) Adam trial 3

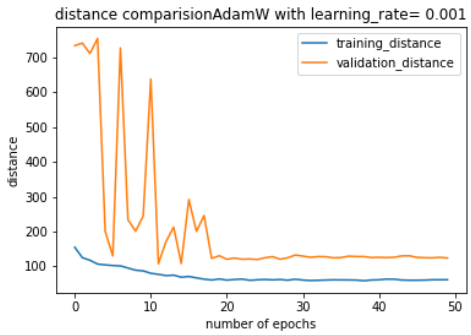
Figure 25: Distance Plots of Proposed Model with Adam



(a) AdamW trial 1

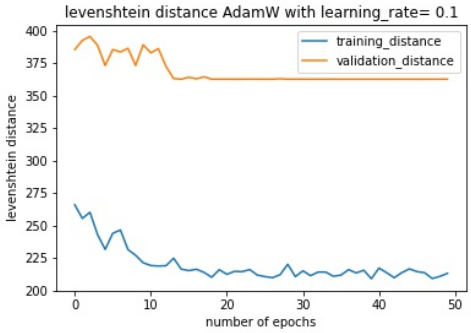


(b) AdamW trial 2

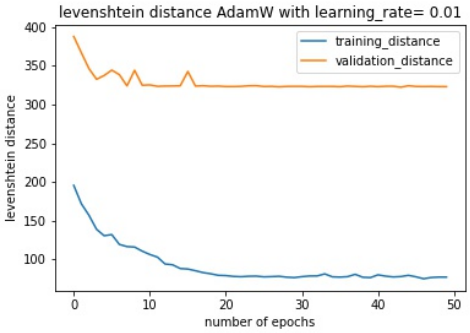


(c) AdamW trial 3

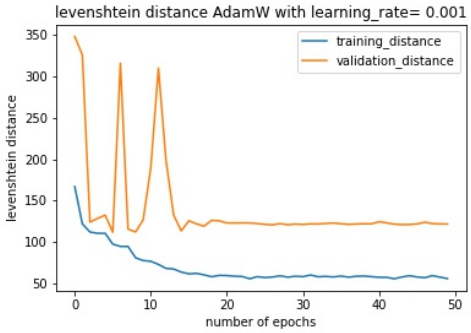
Figure 26: Distance Plots of Baseline Model with AdamW



(a) AdamW trial 1

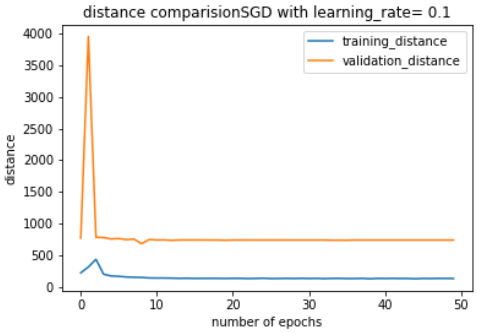


(b) AdamW trial 2

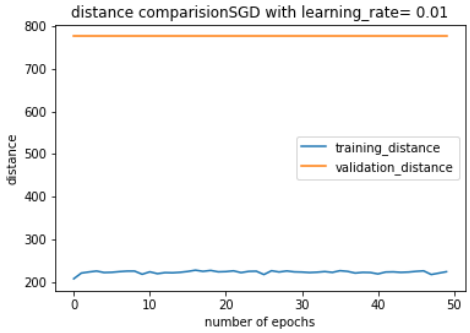


(c) AdamW trial 3

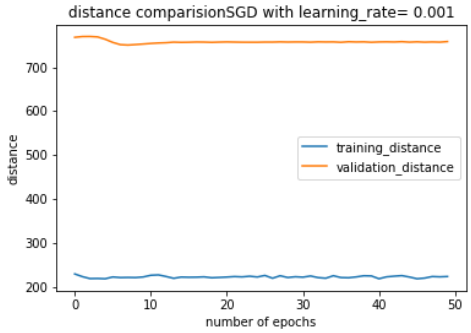
Figure 27: Distance Plots of Proposed Model with AdamW



(a) SGD trial 1

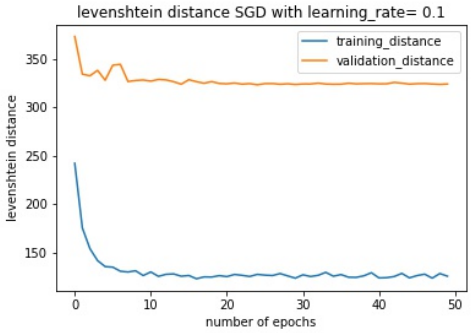


(b) SGD trial 2

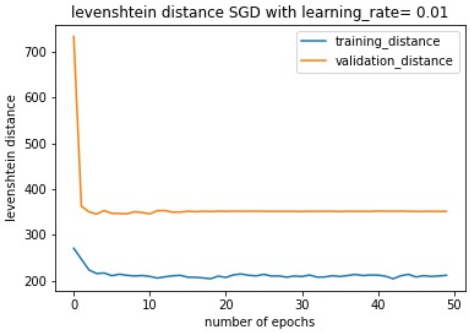


(c) SGD trial 3

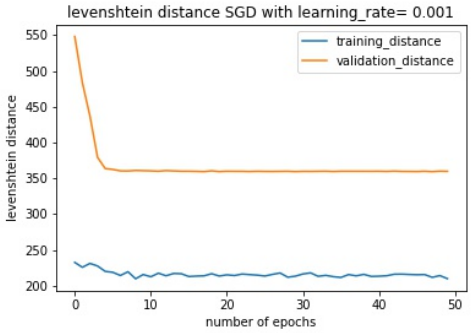
Figure 28: Distance Plots of Baseline Model with SGD



(a) SGD trial 1

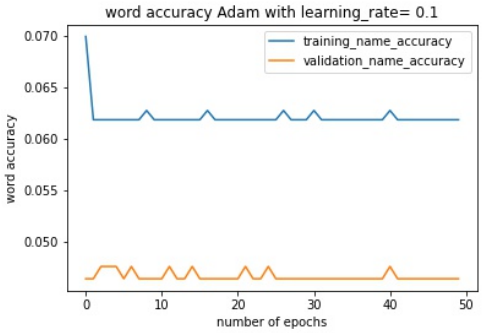


(b) SGD trial 2

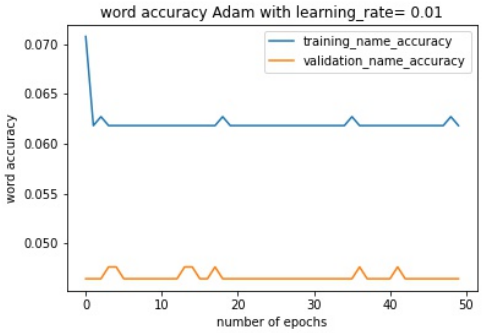


(c) SGD trial 3

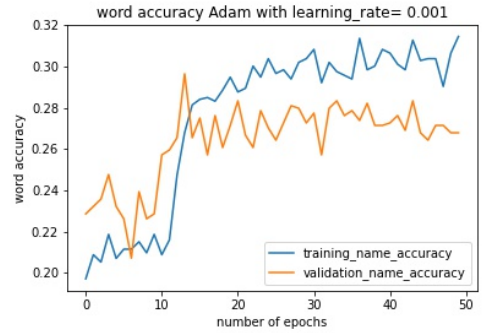
Figure 29: Distance Plots of Proposed Model with SGD



(a) Adam trial 1

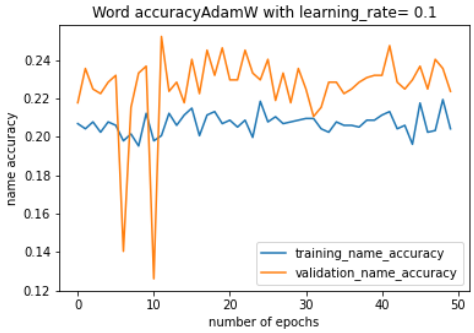


(b) Adam trial 2

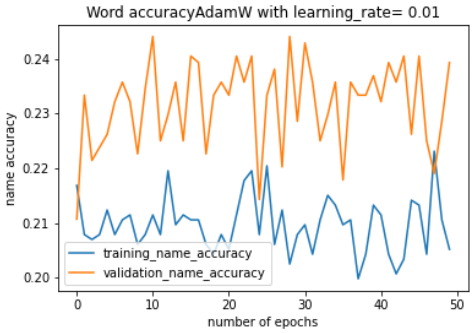


(c) Adam trial 3

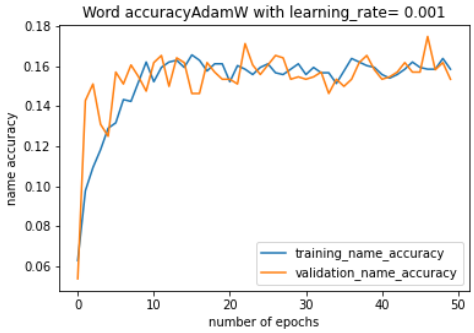
Figure 30: word Prediction Accuracy Plots of Proposed Model with Adam



(a) AdamW trial 1

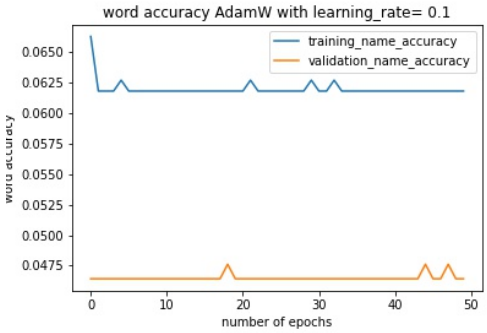


(b) AdamW trial 2

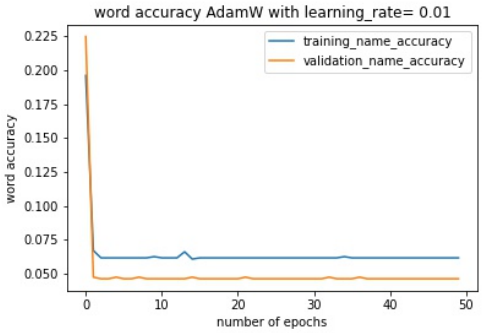


(c) AdamW trial 3

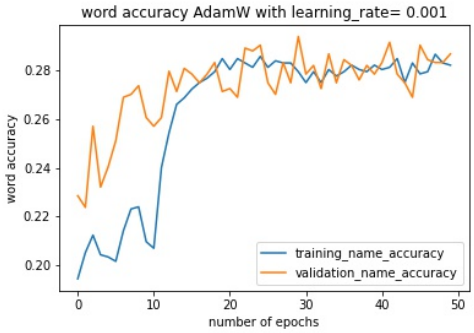
Figure 31: Word Prediction Accuracy Plots of Baseline Model with AdamW



(a) AdamW trial 1

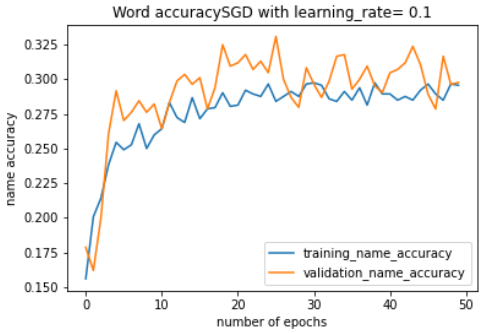


(b) AdamW trial 2

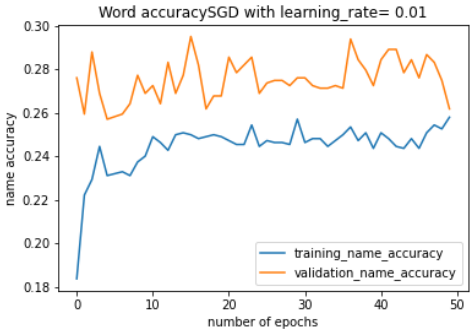


(c) AdamW trial 3

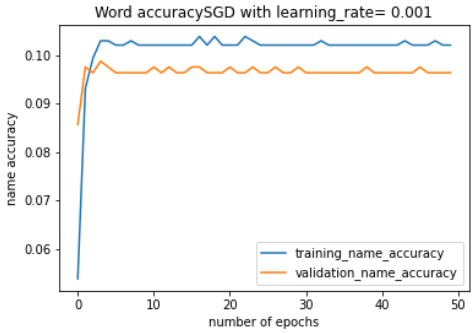
Figure 32: Word Prediction Accuracy Plots of Proposed Model with AdamW



(a) SGD trial 1

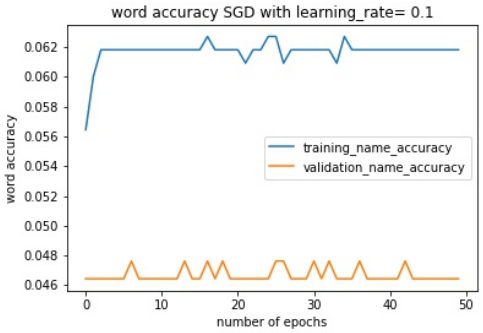


(b) SGD trial 2

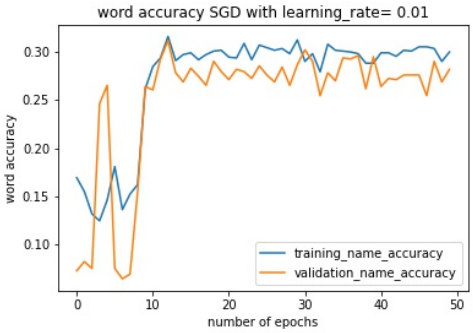


(c) SGD trial 3

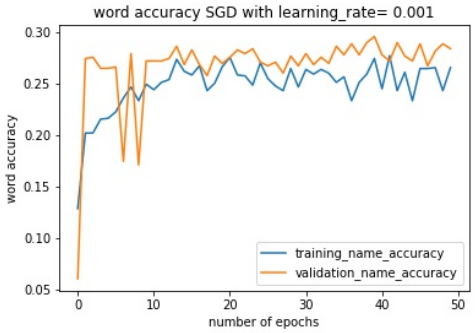
Figure 33: Word Prediction Accuracy Plots of Baseline Model with SGD



(a) SGD trial 1



(b) SGD trial 2



(c) SGD trial 3

Figure 34: Word Prediction Accuracy Plots of Proposed Model with SGD