Speech recognition with DNN-LAS

Jack Jin
Stanford University
Stanford, CA 94305
jackjin@stanford.edu

Pengda Liu
Stanford University
Stanford, CA 94305
pengdaliu@stanford.edu

Geng Zhao
Stanford University
Stanford, CA 94305
gengz@stanford.edu

Abstract

In this project, we build a speech recognition system by trying to improve the original Listen, Spell and Attend model. We evaluated our performance with two metrics: character error rate (CER) and word error rate (WER). We were able to generate a result only 3.8% below Google with only one twentieth of their data set.

1 Introduction

Speech recognition is an interesting topic in deep learning because of its difficulty and its great applications in life: Amazon Echo, Apple Siri, speech typer, etc. The most common architecture used in speech recognition is recurrent neural network (RNN) and its variants. In this paper, we implemented the Listen, Attend, and Spell (LAS) model [1] with an additional Deep Neural Network (DNN) feature extractor. The body of our model consists of an attention-based two multilayer Long Sort Term Memory (LSTMs) networks. This model is very efficient for end-to-end speech recognition tasks, and produces reasonably good results even without any language model rescoring or hidden Markov model (HMM).

2 Related work

There are many good models for doing end-to-end speech recognition tasks. We are particularly interested in the LAS model which was first introduced in Google's paper [1] in 2015, compared to other models such as Connectionist Temporal Classification (CTC) [2] and standard Deep RNN [3]. Though the paper shows that this model is decently good, but we disagree with the method in that it feeds the audio features directly into the pyramidal bidirectional LSTM network, and that it used one-hot format for samples from the previous time step. In our project, we attempted to improve this model on these two points.

3 Data and preprocessing

We combined the TIMIT speech corpus [4] with VoxForge speech database [5] as our data set. The total number of samples is over 90,000. Audio files that are longer than 8 sec were excluded, leaving us with 80,000 samples that totaled about 100 hours of audio.

For each audio file, 40-dimensional MFCC frames were computed for every 10 ms, and then zero-padded to 800 frames (8 sec). Figure 1 shows the distribution of the length of audio signals in frames. Every 10 consecutive frames were then concatenated into a 400-dimensional feature, which spans roughly the duration of a phoneme.

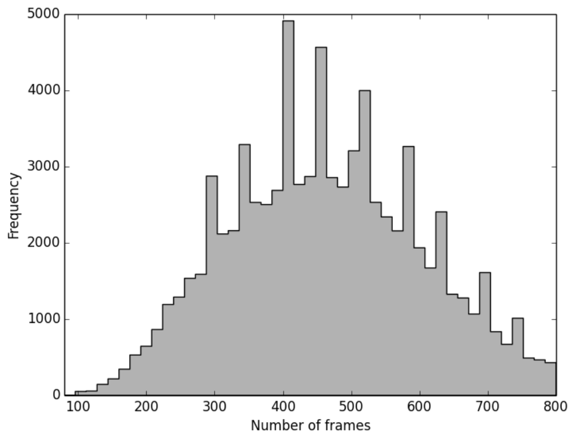

Figure 1: Distribution of length of audio signals.

All non-alphabetical characters except `<space>` were removed from the text labels. Two special characters `<sos>` and `<eos>` were attached to the start and end of each sentence. We then padded the entire sentence to the length of 100 characters. The character set of all letters, `<space>`, `<sos>`, and `<eos>` was encoded with integers $0, \ldots, 28$.

4 Model

The LAS model is subdivided into two components: Listen, and Attend-and-Spell (see Figure 1). The overall structure of the network is shown in Figure 2. The Listen component takes as input MFCC feature blocks from standard waveform audio files, and feeds its output as the input to the Attend-and-Spell component. The main structure of the Listener is a three-layer pyramid Bidirectional LSTM (pBLSTM).

Listen:

$$h_i^j = \text{pBLSTM}(h_{i-1}^j, \left[h_{2i}^{j-1}, h_{2i}^{j-1}\right]), \tag{1}$$

$$h_i^1 = \text{DNN}(x_i), \tag{2}$$

where DNN is a 3-layer fully connected neural network with ReLU activation. This architecture both reduces the dimension of input and extract the content for the attend and spell part.

Attend and spell:

$$s_i = \text{2layer-LSTM}(s_{i-1}, [c_{i-1}, A_{i-1}]), \tag{3}$$

$$A_i = \text{Attention}(s_i, h), \tag{4}$$

$$c_i = \text{MLP}([s_i, A_i]), \tag{5}$$

where Attention is defined as follows:

$$\text{Attention}(s_i, h) = \sum_t \frac{\exp(\phi(s_i) \cdot \lambda(h_t)))}{\sum_u \exp(\phi(s_i) \cdot \lambda(h_u))} h_t.$$

At each time step, the state vector s_i is calculated with a two-layer stacked LSTM taking the concatenation of the character distribution (c_{i-1}) and attention context (A_{i-1}) from previous step.

Instead of taking the direct one-hot prediction from the previous time step as the original paper does, we thought it is better to feed the distribution vector as it encodes more information about prediction from previous step and could reduce bias.

The attention context vector could be understood as a weighted sum of the output from the Listener. ϕ and λ are two one hidden layer neural network with the same final dimension. Finally, at time step i, the character distribution is calculated by a one hidden layer neural network taking the concatenation of s_i and A_i.

Note that we tried to only feed prediction from previous time step to the two-layer LSTM but the model is performing very poorly. So during training we use a sampling rate of 0.9 on the input to LSTM for equation (3):

$$c = (1 - r)c_{i-1} + rg_{i-1},$$

where r is a Bernoulli random variable with expectation 0.9, and g_{i-1} is a one-hot vector representing the ground truth of the character at time step i. This enables our model to do both good training and be robust against testing compared to just feed the LSTM with the ground truth.

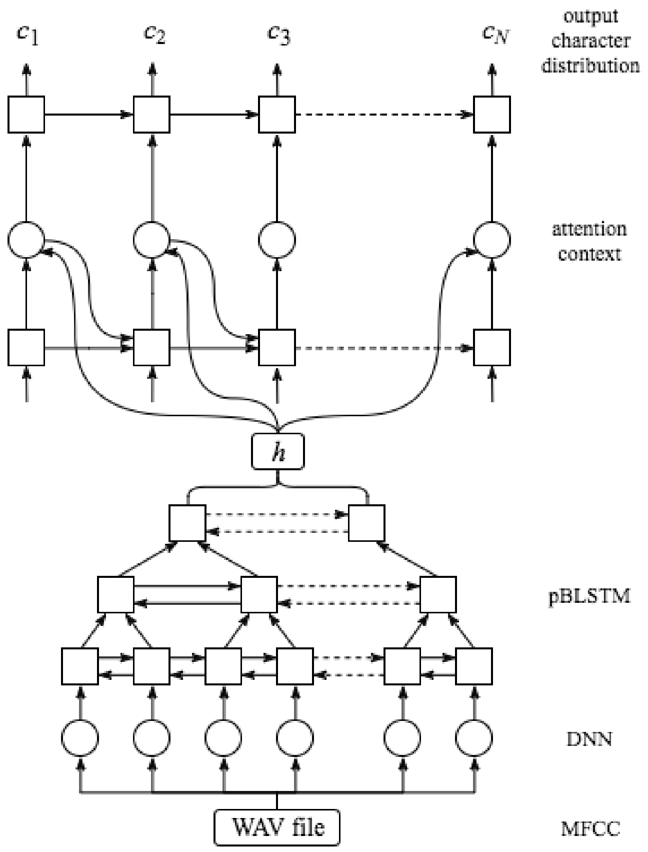

Figure 2: LAS model with a DNN layer.

5 Experiments

5.1 Hyper-parameter tuning

We applied a randomized searching strategy for hyper-parameters tuning. The data we used here is a subset of data that we set aside from the data we used in the actual training. The hyper-parameters include the size of the hidden layers in out network, the batch size, dropout rate, and the learning rate. Following are the range of our search:

Learning rate: $[10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}]$

Hidden state size of pBLSTM: $[5, 10, 20, 50, 100, 100, 100, 150, 150, 200, 200, 300, 400, 500]$[1]

Other hidden layers state size: $[50, 100, 200, 300, 400, 500]$

Batch size: $[20, 40, 50, 75, 100]$ (for minibatch training update)

Drop-out rate: $[0.5, 0.6, 0.7, 0.8, 0.9, 1.0]$

We sampled 30 combinations of hyper-parameters, training for 50 epochs under each configuration. Figure 3 shows the terminal training and development losses for the 30 sets of hyper-parameters. We found that the most influential parameter was the learning rate, with the optimal value around 10^{-2} to 10^{-3}. The second most influential parameter was the drop-out rate, which was optimal at 0.9. Finally, we observed that the hidden layer size for the pBLSTM had little effect on the loss when it is over 20, but its multitude did greatly affect the run time of one epoch.

We found no clear correlation in the test loss with respect to the other hyper-parameters (the other hidden layer sizes and the batch size), so we simply averaged their values in our top five runs and rounded them up to a multiple of 100.

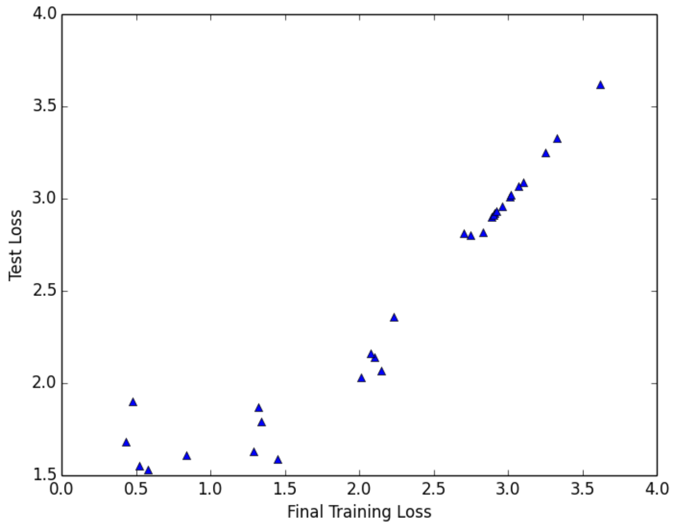

Figure 3: Losses after 30 epochs for 50 sets of hyper-parameters.

5.2 Training

We trained our model using Adam optimizer and cross entropy loss with minibatch update. In the final training, we used most of the optimal hyper-parameters found in our tuning process with one exception of the batch size since we had a larger data set in the training process. We partitioned the data into a training set, a development set, and a test data set, with ratio 7:2:1. Finally, after 500 epochs, we used the result with the lowest loss on development set to generate a final test result.

6 Results

Our original decoding on test set was done by the direct feed forward. However, when analyzing test outputs, we noticed that lots of the predictions differ from the ground truth by 1 character, which motivated us to add an auto-spell-corrector on the test set using the python package `autocorrect` and the improvement is huge.

[1]We set the same value multiple times to increase the chance of it being selected.

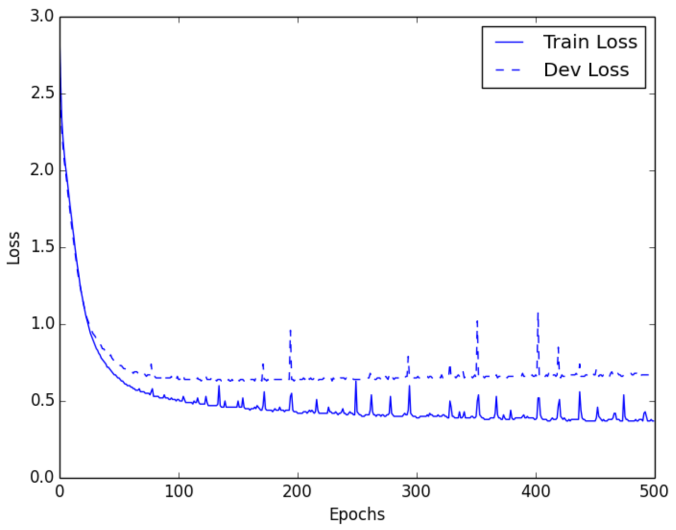

Figure 4: Training and development losses of the DNN-LAS model.

Table 1 shows results from different experiments as well as the cutting edge on the LAS model.

Table 1: Comparison of performance between DNN-LAS and the original LAS implementation from Google.

Model	WER
Google LAS	16.2%
DNN-LAS with auto-corrector	20%
DNN-LAS	25%

Our CER is 16% without auto-corrector. With the additional spell auto-corrector, our result still cannot beat the state-of-the-art performance. However, note that the original LAS model by Google was trained on nearly three million Google voice search utterances (representing 2000 hours of speech), yet our data set is relatively tiny with only 80,000 utterances (representing roughly 100 hours of audio). We believe that our result can be further improved if trained on larger data set. Figure 5 shows a sample prediction made by our model. The predicted sequence is the same as the ground truth, except at the letter `A(ND)` and `W(IDTH)`.

7 Discussion

Through the figure below, we can see that our model is doing extremely well in the middle of the sentence, while our model is not doing very well on the beginning of sentences. Indeed, by examining our model , we can see that to make the prediction about the first character, it takes the padded `<sos>` token as the input from the first time step, thus making the prediction for the beginning of the sentence very difficult, our future plan is to use a Bidirectional LSTM for the Attend and Spell part.

Furthermore, we noticed that although our WER is higher than previous results, we notice that quite a lot of the errors we made are because of character misspelling while the other characters in the same word are correct. Thus, adding a language model could potentially improve our correctness a lot, so this is also among our next steps.

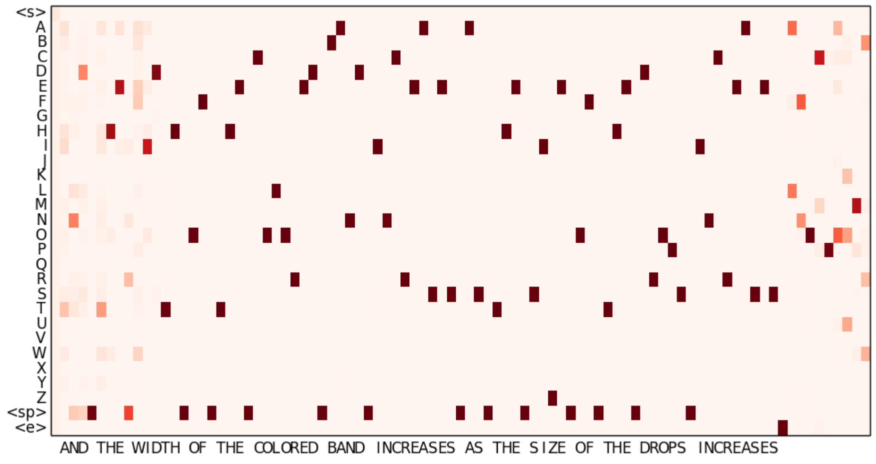

Figure 5: Predicted character distribution for sentence "`AND THE WIDTH OF THE COLORED BAND INCREASES AS THE SIZE OF THE DROPS INCREASES`".

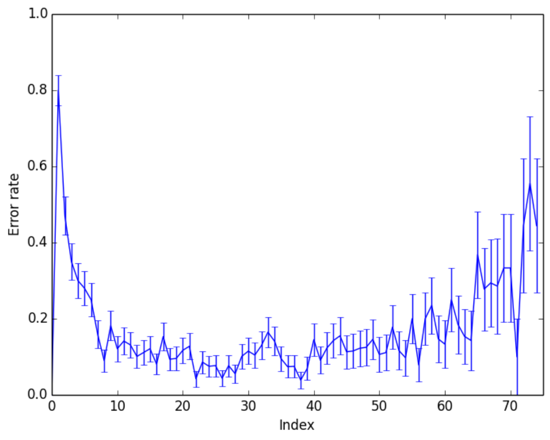

Figure 6: Error rates at different position into the sentence.

Acknowledgments

Special thanks to Dr. Socher and Pr. Chris's instruction and our mentor Verma Prateek's guidance.

References

[1] Chan, W., Jaitly, N., Le, Q. V., and Vinyals, O. (2015). *Listen,attend and spell. arXiv:1508.01211 [cs, stat].*

[2] Alex Graves, et al. (2006) *Connectionist Temporal Classification: Labelling UnsegmentedSequence Data with Recurrent Neural Networks. ICML*

[3] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton. (2013). *Speech Recognition with Deep Recurrent Neural Networks. arXiv:1303.5778 [cs.stat]*

[4] Garofolo, John, et al. TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1. Web Download. Philadelphia: Linguistic Data Consortium, 1993.

[5] Voxforge.org. Free Speech... Recognition (Linux, Windows and Mac) - voxforge.org. *http://www.repository.voxforge1.org/downloads/SpeechCorpus/Trunk/Audio/Main/*