Deep RNN Speech Recognition with Sub-Labels

Jiayu Wu  Yangxin Zhong  Qixiang Zhang
Stanford University  Stanford University  Stanford University
jiayuwu@stanford.edu  yangxin@stanford.edu  qixiang@stanford.edu

Abstract

In this paper, we introduce a top-performing speech recognition system using both end-to-end deep learning and sub-labeling (i.e., beginning, medium, and ending stages) of basic language units. Our system achieves higher accuracy and robustness than traditional speech recognition systems and other systems applying deep learning techniques. Moreover, this system does not require lexicon, thus enabling more flexibility in learning process. The main architecture of our system is a multi-layer bidirectional recurrent neural network (BiRNN), and our major breakthrough is breaking down language labels into sub-labels, thus fitting each sub-label into more specific context and boosting recognition performance. Our system outperforms the advanced Deep Speech by 1.5% in character error rate (CER) and 1.5% in word error rate (WER) on the famous TIMIT corpus.

1. Introduction

Speech recognition has long been a field of promise and challenge. Over the past years, speech recognition has been widely applied in media live subtitling, offline speech-to-text conversion and other realms. To build a top-performing recognition system, pipelines with advanced models are necessary. In this paper, we describe an end-to-end system that combines deep learning algorithms with a sub-label model. The approach makes use of a large bi-directional recurrent neural network (Bi-RNN) with the help of GPU computing resources and a large dataset. This approach achieves higher performance than both traditional speech recognition algorithms and other algorithms based on deep learning on TIMIT corpus, attaining a PER of 24.2%, a CER of 29.8%, and a WER of 74.4%, and showing a smooth training process.

Traditional speech recognition systems use fixed algorithms and focus on making progresses in multiple areas such as acoustic models and Hidden Markov Models (HMMs) [11][10]. Efforts have been made to improve different parts of the training processes [7][5][9]. These previous efforts have significantly improved performance of speech recognition systems, but they only come into use in traditional speech recognition realm, and they appear relatively less robust in environments such as noisy conditions.

Contrary to traditional speech recognition systems, our system makes two major breakthroughs. Firstly, making use of sufficient training data and computing power, we are able to train a promising deep-learning system in an end-to-end way so that it can achieve high performance and robustness to noises and speaker variations. Specifically, we use a 5-layer neural network model with one layer being Bi-RNN, with architectural details explained in section 4. What’s more, our system applies 3 sub-labels (namely beginning, medium, and ending stages) to basic language units (phonemes and characters). This helps to specify different contexts that a basic language unit is in, and therefore can boost the system performance.

In the rest of this paper, we will first introduce previous work and the data preprocessing techniques we use. Then we will introduce the detailed configuration of our Bi-RNN model and illustrate our sub-label model. After that, experiment settings and results will be presented, together with the final conclusion.

2. Related Work

Traditional speech recognition systems tend to chain together a pipeline of fine-tuned sub-systems. One of the most important phases of such a pipeline is HMM alignment, which helps to pick the most likely frame segmentation for labels. [11] proposes CTC loss function that sums probabilities over all possible alignments, which suggests a new way to build a speech recognition system without using a HMM to align frames beforehand. Towards a simpler pipeline of speech recognition systems, [6] and [2] show that end-to-end deep RNN models are able to achieve state-of-art performance. However, the absence of HMM alignment raises a new question: how can we leverage the power of language models if we do not use a HMM system that is built from a lexicon? [3] investigates a lexicon-free approach to build speech recognition systems. Although ex-
periment in [3] shows that a fined-tuned traditional HMM-GMM model achieves lower CER and WER, it also shows that an end-to-end deep RNN model, attached with CTC loss along with a decoding process that integrates language models, can have comparable performance. [3] suggests it is promising to build a top-performing speech recognition system that is significantly simpler than a HMM-GMM pipeline.

3. Data Preprocessing

For dataset, we use TIMIT corpus for training and testing. TIMIT is a corpus of read speech designed to provide speech data for acoustic and phonetic studies and for the development and evaluation of automatic speech recognition systems. It contains broadband recordings of 630 speakers of American English, each reading ten phonetically rich sentences. The corpus also includes time-aligned orthographic, phonetic and word transcriptions as well as a 16-bit, 16 kHz speech waveform file for each utterance [4].

TIMIT dataset has 6300 utterances in total, including 4158 training examples and 2142 testing examples. There are 8 dialect types in this corpus and 3 sentence types. The sentence type Dialect (SA) was meant to expose the dialectal variants. The sentence type Compact (SX) was designed to provide a good coverage of phones with emphasis on difficult phonetic contexts. The sentence type Diverse (SI) was selected from an existing text corpus to add diversity.

3.1. Feature Extraction

For each audio frame, we compute 12 mel-frequency cepstral coefficients (MFCC), 12 ΔMFCC, 12 ΔΔ MFCC, 1 energy, 1 Δ energy, and 1 ΔΔ energy, and thus 39 features in total. We use scikits.audiolab to read audio files and compute MFCC and energy.

In our experiments, each audio frame contains 256 samples, which are used to compute features for this frame. The frame shift size we use is 160 samples. Thus each two adjacent frames have 96 overlapping samples. The remaining samples in the end will be cut if (number of samples - 256) is not divisible by 160.

The delta feature for time stamp $t$ is computed as $d(t) = \frac{c(t+w) - c(t-w)}{2w}$, where $w$ is the window size. The larger $w$ is, the longer timespan of the feature variation we are considering. We use $w = 2$ when extracting delta features.

To construct final features for each frame, we introduce a hyperparameter context size. For each frame, we also include 39 features for each of the context size frames on the left and 39 features for each of the context size frames on the right. Thus, we have $39(2^{\text{context size}} + 1)$ final features for each frame.

<table>
<thead>
<tr>
<th>Label Type</th>
<th>#Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phoneme</td>
<td>29</td>
</tr>
<tr>
<td>Sub-Phoneme</td>
<td>83</td>
</tr>
<tr>
<td>Character</td>
<td>62</td>
</tr>
<tr>
<td>Sub-Character</td>
<td>182</td>
</tr>
</tbody>
</table>

3.2. Label Extraction

We extract four types of labels for our project: (1) letters from word annotation; (2) split letters (-beg, -mid, -end) from word annotation; (3) phones from phone annotation; (4) split phones (-beg, -mid, -end) from phone annotation.

For example, for label type (1), if we have [‘y’, ‘o’, ‘u’, ‘ ‘, ‘a’, ‘r’, ‘e’], then for label type (2), we have [‘y-beg’, ‘y-mid’, ‘y-end’, ‘o-beg’, ‘o-mid’, ‘o-end’, ‘u-beg’, ‘u-mid’, ‘u-end’, ‘ ‘, ‘a-beg’, ‘a-mid’, ‘a-end’, ‘r-beg’, ‘r-mid’, ‘r-end’, ‘e-beg’, ‘e-mid’, ‘e-end’]; for label type (3), if we get [‘ow’], then for label type (4), we have [‘ow-beg’, ‘ow-mid’, ‘ow-end’]. Notice that we do not split some special labels, such as $h\#$ (symbol that denotes a start or end of an utterance) and space.

We extract words and phones but ignore time segmentation data in the annotation files since we only work on models with CTC loss.

A special space label is added to letter labels as well as split letter labels between the last letter of a word and the first letter of the following word. Thus, the space label exists in both label type (1) and label type (2). Each label type also has a unique blank label, which is only used in CTC training process and does not exist in the ground truth labels.

To sum up, label type (1) has 29 unique symbols, label type (2) has 83 unique symbols, label type (3) has 62 unique symbols, and label type (4) has 182 unique symbols.

4. Approach

4.1. Bi-RNN Model Architecture

In this section we introduce our Bi-RNN model modified from an advanced speech recognition system called Deep Speech[6][2]. The basic idea here is to achieve higher recognition rate by training a Bi-RNN using multiple GPUs and thousands of data. Apart from high recognition accuracy, this RNN model fits in well with GPUs by using a novel model partition scheme to improve parallelization. In our model, some modification are made from the original Deep Speech model for better performance. These modifications will be elaborated later in this paper.

The RNN Architecture we use is a 5-layer bidirectional RNN network. The details of feature and label extractions are elaborated in section 1.2 and section 1.3. After extracting the features, we get a training set denoted by $X = \{x_i\}$, where each $x_i$ is a feature vector concatenated by features
extracted from the \( t \)-th speech time frame in the \( i \)-th utterance as well as its \( 2 \times \text{context}\_\text{size} \) neighboring frames on both sides. Then we feed each \( X^t = \{x^t_1, \ldots, x^t_T\} \) into our model. The first 3 layers of our neural network are fully connected layers

\[
h^l_t = g(W^l h^{l-1}_t + b^l)
\]

with the activation function being:

\[
g(z) = \min(\max(0, z), \text{clip}),
\]

and \( \text{clip} \) being a hyper-parameter. The hidden units at layer \( l \) and time step \( t \) is denoted by \( h^l_t \) with the convention that \( h^0_t \) is the input feature vector \( x_t \). The activation function serves as a band-pass filter to restrict output to an allowed range \([0, \text{clip}]\). In our experiments, we set \( \text{clip} = 20 \). We also use batch normalization after each layer to avoid killing gradients in blocked range. The fourth layer is a bi-directional RNN with two hidden state units: forward hidden state \( h^f \) and backward hidden state \( h^b \):

\[
h^l_t = LSTM(W^f h^l_t + W^f h^l_{t-1} + b^f)
\]

\[
h^l_t = LSTM(W^b h^l_t + W^b h^b_{t+1} + b^f)
\]

We apply standard LSTM cells to our RNN network for its ability to reduce vanishing/exploding gradient issue. Then \( h^l_t \) and \( h^b_t \) are concatenated as the input of the fifth layer, which is a fully connected layer without activation as the final output layer. The output is then used to calculate the softmax probabilities for each label at each time step. With these probabilities and phoneme/letter labels of each utterance, we are able to use a standard CTC loss function \([1] \) to optimize our model. For optimization, we use Adam Optimizer to mitigate the over-shoot problems. Architecture of the whole model is shown in Figure 1.

For the decoding process, we use a beam search decoder with a beam width of 100. Language model is not used in our decoding process, and thus the decoder will only consider CTC likelihoods output by the model.

### 4.2. Deep Speech with Sub-Labels

In this section we introduce a model with sub-labels (see Section 3.2). The key idea here is to split one phoneme or letter label into three sub-labels (i.e. -beg, -mid, -end) and use the sub-label sequence as target of CTC loss. Correspondingly, we will have ~3x prediction classes compared to original ones (we don’t split space or blank). In training time, we will use the same network (expect for the output layer) and the same loss function to retrain the model with these sub-labels. In testing time, in order to convert the sub-label sequence back to sequence of original labels, we will apply a post-processor to the output of decoder. Then PER or WER will be calculated using the converted sequence as prediction.

The idea behind using sub-label is that different stages of a phoneme may have very different features. With labels of a more detailed category, the classifier may be able to identify these stages more accurately and thus increase the recognition precision of each frame. Moreover, with a post-processor, we can introduce some smoothing methods to utilize the power of majority voting of sub-label predictions on consecutive frames. Hopefully, with these two benefits, we could lower PER and WER of the model.

Here we illustrate the design of our post-processor, which is applied to the sub-label sequence output by the beam search decoder. The goal of the post-processor is to convert a sequence of sub-labels to a sequence of original labels. For example, for phoneme, we want to convert a sequence of \([\text{ah-beg, ah-mid, ah-end}]\) to \([\text{ah}]\); for character, we want to convert a sequence of \([\text{[h-beg, h-mid, h-end, i-beg, i-mid, i-end]}]\) to \([\text{[h, i]}]\).

Since a decoded sequence may not necessarily have the three stages of a label in a row (e.g., we may have \([\text{[h-beg, h-mid, i-mid, i-end]}]\)), we need to handle the case of missing sub-labels of certain stages. Furthermore, there may be some irrelevant sub-labels amid a group of relevant sub-labels, for example, we may find that in \([\text{[h-beg, x-end, h-mid, h-end, i-beg, i-mid, i-end]}]\), the sub-label \(x\)-\text{end} is irrelevant and it is very likely just a noise. Thus, we should penalize sub-labels that appear without a relevant group but also be cautious about not missing out correct sub-labels that just happen to miss certain stages. To handle both cases, we introduce two hyper-parameters \(\text{at least include num} \) and \(\text{at least include} \), to control how selective the post-processor is. Parameter \(\text{at least include num} \) specifies the number of sub-label types we want to include when considering a block of contiguous sub-labels to one original label. For example, if \(\text{at least include num} = 2 \), then we only collapse a block of sub-labels when the block contains at least two sub-label types, which means we want the block to have \([\text{beg, mid}], [\text{beg, end}], [\text{mid, end}], \) or \([\text{beg, mid, end}].\)
thermore, if we only want to collapse a block of sub-labels when both -beg and -mid appear in this block, then we set \( \text{at least include} = \{\text{beg, mid}\} \).

To compute WER, we have an additional step to collapse a sequence of letters to a sequence of words by just joining this sequence of letters because space is also a letter label. Then for each pair of predicted word sequence and target word sequence, we tokenize the words that appear in this pair and finally compute normalized edit distance of these two sequences.

5. Experiments

<table>
<thead>
<tr>
<th>Method</th>
<th>PER</th>
<th>CER</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bi-RNN on Phoneme Labels</td>
<td>25.6</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Bi-RNN on Phoneme Sub-Labels</td>
<td>24.2</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Bi-RNN on Character Labels</td>
<td>–</td>
<td>31.3</td>
<td>75.9</td>
</tr>
<tr>
<td>Bi-RNN on Character Sub-Labels</td>
<td>–</td>
<td>29.8</td>
<td>74.4</td>
</tr>
</tbody>
</table>

Table 1. Test Errors on TIMIT dataset. All test errors are evaluated on the complete test set of TIMIT. PER/CER/WER stand for phoneme/character/word error rate.

We used TensorFlow as training framework, and set a simple single-layer, single-direction RNN network with standard GRU cells as our experiment baseline.

To prevent overfitting, we used dropout after each of the first three layers with a keep probability in \([0.4, 0.8]\). We had tried but did not use weight decay for regularization in our final model. Adam optimizer was used in training with \( \beta_1 = 0.9 \) and \( \beta_2 = 0.999 \). The learning rate was set to 1e-3 and the batch size we use was 64. We ran the training process for 50 epochs, and each epoch took around 13 minutes using one NVIDIA Tesla K80 GPU.

5.1. TIDIGITS

We first compare our model with the baseline in TIDIGITS dataset and achieve a maximum validation WER of 6.0% for our model. In comparison, the baseline validation WER is 6.1%. The results prove the viability of our model. The WERs of both models are close because the training set is too small and easy to fit. Moreover, the prediction problem in TIDIGITS is relatively simple because we only have 12 label classes.

5.2. Phoneme Recognition on TIMIT

We then tried using our model to fit the full TIMIT corpus with phoneme labels. More concretely, we split the training data (4158 examples) into a training set (90%) and validation set (10%). Figure 2 shows the performance of baseline and our improved Deep Speech model. Our model outperforms baseline a lot in terms of phoneme error rate (PER). However, in this experiment our model experiences an issue of overfitting. While the training loss of our model keeps decreasing all the way, the validation loss plateaus after 10 epochs and it even increases after 25 epochs. After 50 epochs, our model obtains a 8.8% PER on training set but gets a 29.0% PER on validation set. The huge gap indicates that our model overfits the training set. To solve this issue, we add a dropout layer after each fully connected layer except for the output layer with a keep probability in \([0.4, 0.8]\). After further fine-tuning, our model with original labels is able to achieve a test PER of 25.6%.

Then we train our model with the phoneme sub-labels and use a post-processor for prediction as discussed in section 4.2. As shown in Table 1, our model with sub-labels achieves a test PER of 24.2%, which is 1.4% lower than our model with original labels.

As Figure 3 shows, the validation PER of the model with
sub-labels drops faster and plateaus at a lower level. We also notice that the model with sub-labels shows a sign of overfitting after 30 epochs. The results support our hypothesis that the model with sub-labels is able to be more discriminative and able to learn subtle features of different stages of a phoneme/character. We found that our model with sub-labels has a more severe overfitting issue than the model with original labels. So far we have tried several normalization techniques such as dropout and \( L^2 \) norm, but the problem still exists. This may indicate the dataset is too small for the sub-label model since it’s more discriminative and thus needs more data to learn the common features for different stages of an original label. Furthermore, the faster declining PER also indicates it is easier to train the model with sub-labels.

5.3. Speech-to-Text on TIMIT

We also trained our bi-directional RNN model on TIMIT dataset with character labels and sub-character labels. As shown in Table 1 the model with sub-labels achieves 1.5% lower CER and WER than the model with original labels.

From Figure 4 we see that training WER of sub-labels drops faster than training WER of original labels before 10 epochs, and then they drop at a similar pace. The validation WER of sub-labels drops to around 0.73 and then plateaus while the validation WER of original-labels decreases at a slower but smoother pace. The widening gap between the train and validation WER after 30 epochs of sub-labels shows a sign of overfitting. Similar to the previous experiment, this experiment with character labels also indicates that the model with sub-labels may be easier to train but also more likely to overfit since it is more discriminative and needs more data for generalization.

Figure 5. CER vs. WER. Dashed lines are validation CERs; solid lines are validation WERs. Blue for original labels and orange for sub-labels.

We also notice the significant gap between CER and WER for both models. Figure 5 shows that although the validation CER of both original labels and sub-labels quickly drops to around 0.4 after 10 epochs, both models are only making a small progress towards 0.7 in terms of validation WER. Since WER evaluates a model’s ability to predict correct words, the model needs to be correct on every letter that makes up the correct word. For example, if a word has 5 letters and even the model can achieve 0.2 CER, which means the model is likely to get 4 correct letters, there is still 1 incorrect letter, which will result in the entire word being incorrect. Especially in the case that we did not integrate any language models into the decoding process, one minor misspelling can make the entire predicted word incorrect. Therefore, Figure 5 shows that, without a proper language model, WER is not improving as fast as CER during the training process.

Figure 6. CER vs. WER. Dashed lines are validation CERs; solid lines are validation WERs. Blue for original labels and orange for sub-labels.

Here we show an example of how the model with sub-labels is gradually making progress during the training process (the sequence shown below is converted from a sub-label sequence using our post processing method):
Ground Truth:  
*she had your dark suit in greasy wash water all year*

After 8 epochs:  
*she had yur dar si in greasy wash waa-t er al year*

After 45 epochs:  
*she had your dark sit in greasy wash water al year*

We see that after 8 epochs, the model is able to have the right number of words but fails to get the correct words. After 45 epochs, the model is able to predict most characters correctly but still, some words are incorrect because of minor misspellings.

5.4. Impact of Post-Processor

<table>
<thead>
<tr>
<th>(at_least_include_num)</th>
<th>(at_least_include)</th>
<th>PER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[]</td>
<td>27.9</td>
</tr>
<tr>
<td>1</td>
<td>[beg]</td>
<td>26.7</td>
</tr>
<tr>
<td>1</td>
<td>[mid]</td>
<td>24.6</td>
</tr>
<tr>
<td>1</td>
<td>[end]</td>
<td>26.5</td>
</tr>
<tr>
<td>2</td>
<td>[]</td>
<td>24.2</td>
</tr>
<tr>
<td>2</td>
<td>[beg, mid]</td>
<td>25.7</td>
</tr>
<tr>
<td>2</td>
<td>[mid, end]</td>
<td>25.4</td>
</tr>
<tr>
<td>2</td>
<td>[beg, end]</td>
<td>26.0</td>
</tr>
<tr>
<td>2</td>
<td>[mid, end]</td>
<td>24.2</td>
</tr>
<tr>
<td>3</td>
<td>[]</td>
<td>26.2</td>
</tr>
</tbody>
</table>

Table 2. PER of our model with phoneme sub-labels on TIMIT dataset with different post-processor hyperparameters.

Since the output of our deep speech RNN model is a sub-label sequence from the beam search decoder, we need to use a post-processor to convert it back to a sequence of original labels as the final prediction result. As discussed in section 4.2, we use two hyperparameters \(at\_least\_include\_num\) and \(at\_least\_include\) to control how selective the post-processor is. Table 2 shows PER of our model on TIMIT dataset with different hyperparameter settings.

As shown in the table, there is a huge gap between performance of our model using a naive post-processor and a well-designed one (27.9% vs. 24.2%). With the setting in the first row, we just simply collapse any continuous sub-label block with the same original label into that original label. This can make the model suffer from noise such as \(g\)-mid in the \([h\text{-}beg, g\text{-}mid, h\text{-}end]\) sequence (\([h]\) is the correct original label but predicts \([h, g, h]\) instead), which causes a high PER (27.9%). On the contrary, with the setting in the last row, we are extremely strict that only continuous sub-label blocks of the same original label that contains all three types of sub-labels will be collapse into original label. For example, \([h\text{-}beg, h\text{-}mid, h\text{-}end]\) will be collapsed into \([h]\), but \([h\text{-}beg, g\text{-}mid, h\text{-}end]\) will be collapsed into \([\text{z}d]\). This can help to filter out some noisy or uncertain predictions and thus get a lower PER (26.2%) than the naive method. However, this setting may lose the chance to recover from noise.

For example, if we have \([s\text{-}beg, z\text{-}mid, z\text{-}end]\), it will be a good guess to predict \([z]\) instead of \([\text{z}d]\) since we have two different types of sub-labels of \(z\).

Our model achieves the lowest PER (24.2%) when we set \(at\_least\_include\_num = 2\) and \(at\_least\_include = []\), which means we only collapse those continuous sub-label blocks of the same original label with at least two different types of sub-labels into that original label. For example, \([z\text{-}beg, z\text{-}mid, s\text{-}end]\), \([s\text{-}beg, z\text{-}mid, z\text{-}end]\), \([z\text{-}beg, z\text{-}end]\) and \([z\text{-}beg, z\text{-}mid, z\text{-}end]\) will all be collapsed into \([z]\), while \([z\text{-}beg, s\text{-}mid, z\text{-}end]\) will be collapsed into \([\text{z}d]\) since \(z\text{-}beg\) and \(z\text{-}end\) are not continuous. By doing this, we can leverage the strength of majority voting to give a good guess and smooth out some noise in the predicted sequence. This hyperparameter setting also helps getting the lowest CER and WER in the experiment with character sub-labels.

Another interesting observation is that the sub-label type \(mid\) is the most helpful one to post-processing. When we require the eligible sub-label blocks to include at least the mid sub-label (the 3rd, 6th and 7th rows in Table 2), it can always achieve a relatively good performance. This indicates that generally speaking, the medium stage of a phoneme contains the most important information of that phoneme and our model often yields a higher prediction accuracy for it compared to the beginning and ending stages.

6. Conclusion

To conclude, we propose a speech recognition system using both end-to-end deep learning architecture and sub-labeling techniques for basic language units. Our system with sub-labels outperforms the Deep Speech model using original labels by 1.4%~1.5% on TIMIT dataset in terms of PER, CER and WER. Our experiments show that the sub-labeling method can make the Deep Speech model more discriminative and help to identify different stages of a phoneme or character. Furthermore, experiments show that by using a post-processor, our sub-label model can leverage the power of majority voting and smooth out some prediction noise to get a higher performance.

For now, our sub-label model suffers from the issue of overfitting, regardless of the using of several normalization methods. This may indicate that we need more data for generalization. In the future, we plan to train our model on a larger dataset such as WSJ to see if we can get an even better performance compared to traditional models. Moreover, we will incorporate a language model into our model to further lower the current WER.

References


