Deep Speech Recognition

Abstract

In this project, we attempted with both Kaldi and Deep Speech 2 on efficient speech recognition task. Kaldi is a toolkit for speech recognition written in C++ for use by speech recognition researchers. We ran through 4 tri examples and presented experiment results table in the report. Deep Speech 2 [Amodei et al., 2015] is an End-to-end Deep learning based speech recognition system proposed by Baidu Research. It is round 7x faster than Deep Speech 1, up to 43% more accurate. Possible to deploy the system in online setting. This feature makes it possible for us to implement a real-time demo for online speech recognition. We ported the public github version to Mandarin by feeding the network with news dataset(provided by TA) and THCS30 dataset. Our experiment results show that this system has outperformed Kaidi tutorials with a lower WER score and faster deployment speed. To further improve the prediction accuracy of our system, we equipped our model with a self-designed N-Gram language model and further boosted the WER scores. Our demo is implemented on a distributed level. Front end is implemented based on Web Applications, is able to run at any browsers. Deployment of network is on servers with GPUs. The forward of Deep Speech Network will be largely accelerated. Code of our project will be released.

1 Introduction

Speech recognition can be viewed as generalization of the tagging problem: Given an acoustic output $A_{l,T}$(consisting of a sequence of acoustic events $a_1, ..., a_T$) we want to find a sequence of words $W_{l,R}$ that maximizes the probability $Argmax_w P\ (W_{l,R}|A_{l,T}\ )$.

Using the standard reformulation for tagging by rewriting this by Bayes Rule and then dropping the denominator since it doesn't affect the maximization, we transform the problem into computing $Argmax_w P\ (A_{l,T}\ |W_{l,R})P\ (W_{l,R})$. In the speech recognition task, $P\ (W_{l,R})$ is called the **language model**, and $P\,A_{l,T}\ |W_{l,R}$ is called the **acoustic model**.

We will give a short review on the pipeline of speech recognition.

1.1 Input Transformation

The input to speech recognizer is in the form of a stream of amplitudes, sampled at about 16,000 times per second. But audio in this form is not useful for the recognizer. Hence a feature extractor is needed at this point. Examples of outputs of the feature extraction stage include slices of the raw waveform, spectrograms and the equally popular mel-frequency cepstral coefficients (MFCCs).

1.2 Acoustic Model

An acoustic model is used to represent the relationship between an audio signal and the phonemes or other linguistic units that make up speech. The model is learned from a set of audio recordings and their corresponding transcripts. It is created by taking audio recordings of speech, and their text transcriptions, and using software to create statistical representations of the sounds that make up each word.

1.3 Language Model

A statistical language model is a probability distribution over sequences of words. Given such a sequence, say of length m, it assigns a probability $P\ (W_1, ...W_m)$ to the whole sequence. Those probabilities are estimated from a sample data and automatically have some flexibility.

2 Related Works

We perform two approaches to build the whole speech recognition system here.

2.1 HMM Approach

2.1.1 HMM Acoustic Model

Each base phone q is represented by a continuous density HMM of the form illustrated in 1 with transition probability parameters $\{a_{ij}\}$ and output observation b_j. In operation, an HMM makes a transition from its current state to one of its connected states every time step. This from of process yields the standard conditional independence assumptions for an HMM[Young et al., 2006]:

- states are conditionally independent of all other states given the previous state;
- observations are conditionally dependent of all other observations given the state that generate it.

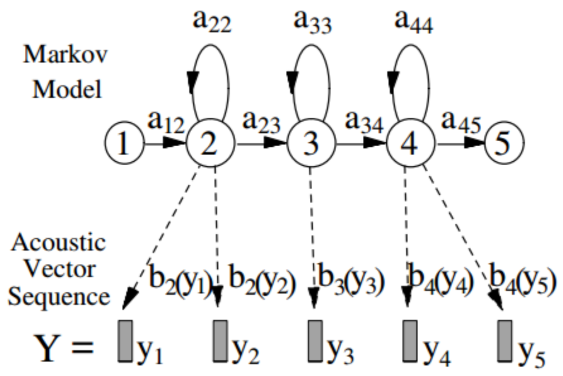

Figure 1: HMM-based phone model

The acoustic model parameters can be efficiently estimated from a corpus of training utterance using the forward-backward algorithm which is an example of expectation-maximization(EM).

2.1.2 N-Gram Language Model

The prior probability of a work sequence $\mathbf{W} = W_1, ..., W_k$ required is given by

$$P(W) = \prod_{k=1}^{K} P(W_K | W_{k-1}, ..., W_1) \quad (1)$$

The N-gram probabilities are estimated from training texts by counting N-gram occurrences to form maximum likelihood parameter estimates. For example, let $C(W_{k-2}W_{k-1}W_k)$ represent the number of occurrences of the three words, then

$$P(W_k|W_{k-1}, W_{k-2}) \approx \frac{C(W_{k-2}W_{k-1}W_k)}{C(W_{k-2}W_{k-1})} \tag{2}$$

The most likely work sequence given a sequence of feature is found by searching all possible state sequences arising from all possible word sequences for the sequence which was most likely to generate the observed data. An efficient way to solve this problem is to use dynamic programming.

2.1.3 CTC

Connectionist Temporal Classification [Graves et al., 2006] is a loss function useful for performing supervised learning on sequence data, without needing an alignment between input data and labels. It can be used to train end-to-end systems for speech recognition.

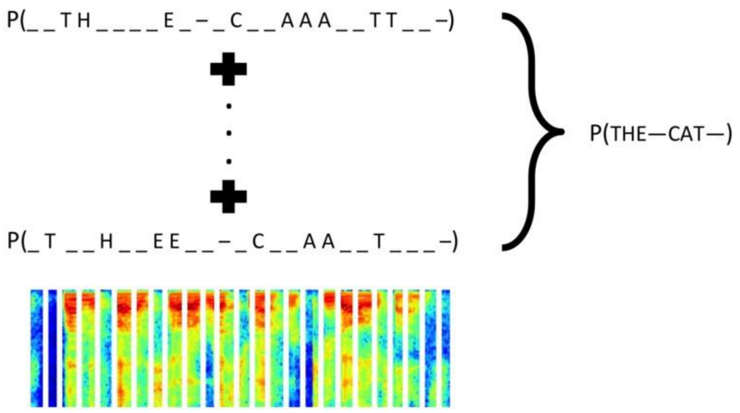

Figure 2: CTC

The illustration above shows CTC computing the probability of an output sequence "THE CAT ", as a sum over all possible alignments of input sequences that could map to "THE CAT ", taking into account that labels may be duplicated because they may stretch over several time steps of the input data (represented by the spectrogram at the bottom of the image). Computing the sum of all such probabilities explicitly would be prohibitively costly due to the combinatorics involved, but CTC uses dynamic programming to dramatically reduce the complexity of the computation. Because CTC is a differentiable function, it can be used during standard SGD training of deep neural networks.

2.2 DNN Approach

In its barest outline, an end-to-end speech recognition pipeline consists of three main components:

- A feature extraction stage which takes raw audio signals (e.g. from a wav file) as inputs and generates a sequence of feature vectors, with one feature vector for a given frame of audio input. Examples of outputs of the feature extraction stage include slices of the raw waveform, spectrograms and the equally popular mel-frequency cepstral coefficients (MFCCs).
- An acoustic model which takes sequences of feature vectors as inputs and generates probabilities of either character or phoneme sequences conditioned on the feature vector input.

- A decoder which takes two inputs –the acoustic model's outputs as well as a language model –and searches for the most likely transcript given the sequences generated by the acoustic model constrained by the linguistic rules encoded in the language model.

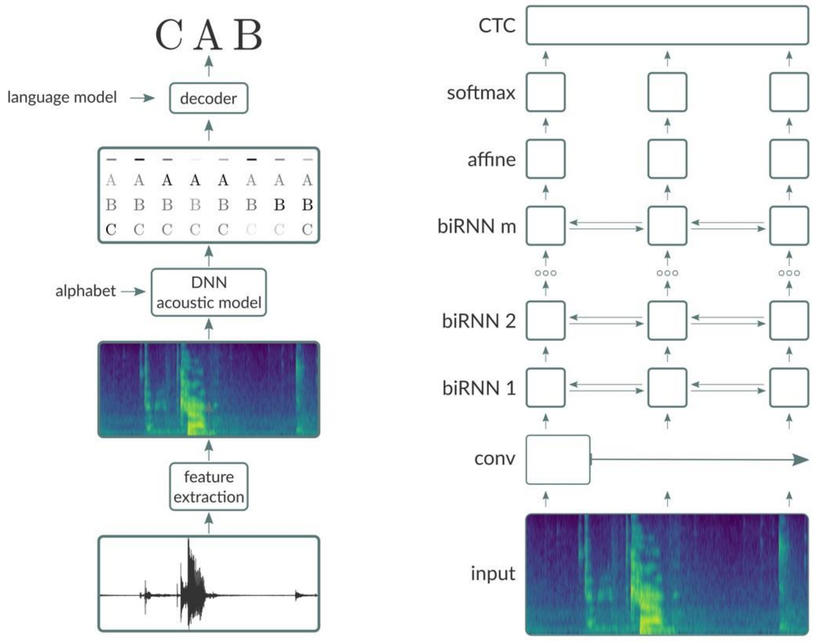

Figure 3: DNN Apporach Pipeline

3 Detailed Approach

3.1 Deep Speech 2 Structure Description

We build our models by training a deep neural network comprised of convolutional (Conv) layers, bi-directional recurrent (BiRNN) layers and fully connected (FC) layers (essentially following "Deep Speech 2"[Amodei et al., 2015]) as shown schematically in the above figure.

This network takes spectral feature vectors as input. The spectral inputs are fed into a Conv layer. In general, one could consider architectures with multiple Conv layers employing either 1D or 2D convolutions. There could also be a cascade of several Conv layers. We found that one Conv layer is strong enough for our relatively small dataset, so we'll employ only one Conv layer in the final version.

The outputs from the Conv layer(s) will be fed into a stack of BiRNN layers. Each BiRNN layer is comprised of a pair of RNNs running in tandem, with the input sequence presented in opposite directions as indicated in the figure.

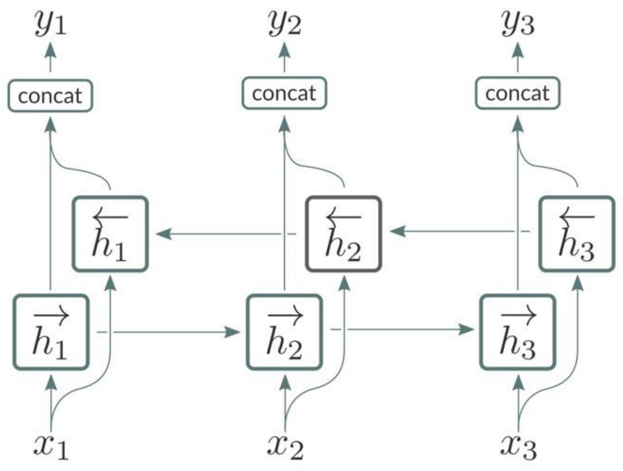

Figure 4: BiRNN Layer

The outputs from the pair of RNNs are then concatenated as shown in the figure. BiRNN layers are particularly suited to processing speech signals as they allow the network access to both future and past contexts at every given point of the input sequence. Though Amodei et al. [2015] stated that applying batch normalization to RNNs will produce better prediction results, our RNNs does not work well with batch normalization so that we discarded it in the final version. We set the output dimension of each RNN layer as 1024, because we find this feature dimension is large enough to encode the information it needs to forward.

The outputs from the BiRNN layers at each timestep are then fed into a fully connected layer. Each unit corresponds to a single character in the alphabet characterizing the target vocabulary. For example, since our data is comprised of Chinese corpora, the alphabet we use include 3645 characters in common use, as well as a blank symbol and an unknown token. The blank character allows the CTC model to reliably deal with predicting consecutive repeated symbols, as well as artifacts in speech signals, e.g. pauses, background noise and other “non-speech”events.

Thus, given a sequence of frames corresponding to an utterance, the model is required to produce, for each frame, a probability distribution over the alphabet.

3.2 Training Strategy

During the training phase, the softmax outputs are fed into a CTC cost function (more on this shortly) which uses the actual transcripts to (i) score the model’s predictions, and (ii) generate an error signal quantifying the accuracy of the model’s predictions. The overall goal is to train the model to increase the overall score of its predictions relative to the actual transcripts.

We use "Xavier" initialization for all layers in our models. To update our model, nesterov momentum method is applied along with annealed learning rate. Most of the model’s hyperparameters, e.g. the depth of the network, the number of units in a given layer, the learning rate, the annealing rate, the momentum, etc., are chosen empirically based on our test on validation dataset. We train the model with a batch size of 10 on 2 NVIDIA TITAN X.

3.3 Add Language Model

With large networks like the ones used by Baidu Research and given enough training data, the network learns an implicit language model. However, since the labeled training data is still small, the use of explicit Language Model still improves WER by a large margin

$$Q(y) = \log(p_{ctc}(y|x)) + \alpha \log(p_{lm}(y)) + \beta word\_count(y) \quad (3)$$

The Goal is to find y that maximized $Q(y)$. Here α and β are tunable parameters found using development set. The optimal y is found using Beam Search. A complete search to the whole possible space gives the best result but is computational expensive, we combined with some greedy tricks to ensure its real-time performance.

We decided to utilize beam search to combine our language model and CTC predictions. Considering the time efficiency we can endure, we've designed a pipeline that is able to equip our real-time requirements. One thing to add is that, since we follow the setting of CTC loss, the algorithm will skip on time t when the top prediction is blank or same as the previous one.

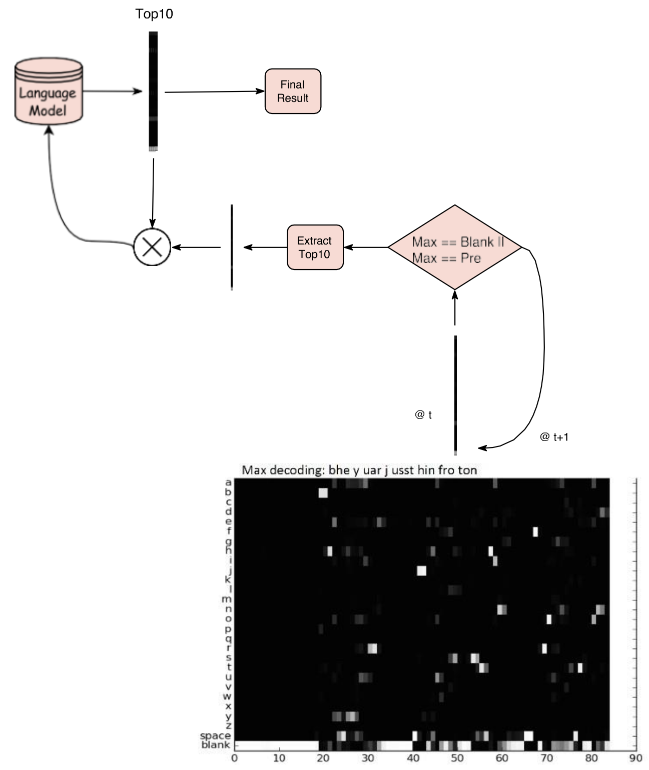

Figure 5: Language Model

We've tried 3 strategies based on different size s of beam. We'll list them as follows

1. if $s == 1$:
   Each time t we apply Cartesian Product of current 6000 predictions with previous calculated sentences and feed them into language model to get scores estimated by language model. We pick the prediction using weighted scores as shown in equation 3.
2. if $1 < s < 200$:
   For each time t, we select s top ranked prediction, apply Cartesian Product with previous results, and send them into language model to give predictions. We resort the weighted results given by language model and CTC, and pick the top s. In this pipeline, no matter how long the sequence is, the max num of items being calculated by our language model is no longer than s^2. This makes the computational cost tolerable.
3. if $s > 200$:

Every moment *t* we apply Cartesian Product with previous results. Top *s* sentences will be selected only by estimated CTC probabilities. We will only feed surviving *s* sentences into language model when the whole clip has been handled. We select the best sentence by weighted estimation of CTC and language model.

3.4 Deployment Strategy

Our demo framework is separated into Front End and Backend. Backend is deployed at servers with GPUs. The forward of Deep Speech Network will be largely accelerated. We load the model into GPU once, and server will listen on port to requests to conduct forwarding computation. Language model will also be ready at servers to calculate scores of top queries given by DNN. Front end is implemented based on Web Applications, which is able to run at any browsers. It is responsible to conduct necessary preprocessing which will be explained later.

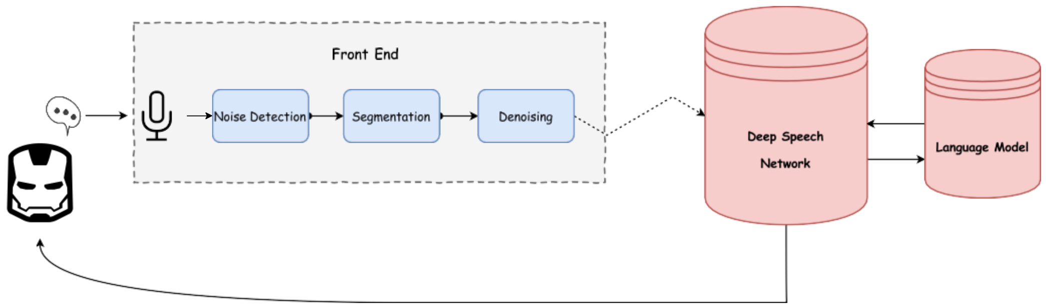

Figure 6: Demo Pipeline

3.4.1 Front End

Noise Detection There are particular features for human's pronunciation (e.g. Voiced and Unvoiced) compared with noise. We detect such cases using handcrafted energy based methods.

Snippet Cutting Here we calculate energy for each frame. Continuous low-energy frames are considered to be silence scene thus we discard them using fixed threshold. Besides, it also requires a minimum time lapse between two sentences.

Denoising The procedure goes as follows,

- When we start the front end, this program will automatically record the environment noise and calculated its spectrum for future reference.
- We first apply FFT to snippet of previously detected noise and regard it as background noise.
- Subtract background noise spectrum from audio spectrum.
- Then, we apply IFFT to audio spectrum and get audio snippet with background suppression.

3.4.2 Communication Between Front End and Back End

After recording, we apply methods described in 3.4.1 and ended up with a locally saved audio file. We upload this file to remote server using secure copy and its filename through TCP socket. Once the remote server has received this file and its filename, it will automatically start forwarding and generate prediction sentence. Once it has finished computing, results will be sent back to front end by TCP socket. Front end will then display the result on the screen.

4 Experiments

4.1 Kaidi Result

We conducted several training on Kaidi. Below is the WER testing result on thchs30 dataset [Wang and Zhang, 2015] containing 30 hour Chinese recording. We use the formal training-testing split proposed by

Table 1: Experiment Results on Kaidi

test			model							
task	dataset		mono	tri1	tri2b	tri3b	tri4b	tri4b_dnn	tri4b_dnn_mpe	tri4b_dnn_dae
phone	test		32.26	20.37	17.24	15.07	13.61	10.2	10.02	
	noise	café	-						73.22	30.78
		car	-						12.93	11.32
		white	-						87.47	30.78
word	test		50.57	36.2	32.4	29.54	28.02	23.65	23.27	
	noise	café	-						87.78	52.4
		car	-						27.36	24.87
		white	-						99.4	66.57

From the table above we see the tri4b_dnnś performance is the best among all. It has been shown that by adding noise data into training procedure testing result in noisy environment will also be improved.

4.2 Deep Speech 2 Result

When training Deep Speech 2 model, we employ both news dataset provided along with thchs30 dataset. We split the news dataset using given annotations. Notice that we drop a snippet if it is too long or too short. We conducted several experiments to examine a reasonable parameter setting.

Table 2: Ablation Study on Parameter Setting

architecture				**train set**	**test wer(cer)**				
conv layers	**rnn type**	**rnn hidden size**	**rnn layer**		**thchs30**		**thchs30 + news**		
					test	**noise**	**test**	**noise**	**lm**
1	gruBN	768	7	thchs30	33.14	*	*	*	*
1	lstm	1024	3	thchs30	30.97	*	*	*	*
1	lstmBN	1024	5	thchs30 + noise	*	42.14	*	*	*
2	lstm	960	5	thchs30 + noise	*	42.31	*	*	*
2	lstm	1300	3	thchs30 + noise	*	42.23	*	*	*
2	lstmBN	600	5	thchs30 + news	4.41	4.41	24.45	*	*
2	lstmBN	768	7	thchs30 + news	4.02	4.02	24.61	*	*
1	lstm	1024	5	thchs30 + news	3.75	3.75	23.56	23.53	19.71
2	**lstm**	**1024**	**5**	**thchs30 + news + noise**	**3.84**	**3.84**	**23.45**	**23.46**	**21.82**

Above is the table for ablation study. Though Amodei et al. [2015] claims that a deeper model with less hidden size will work well, our result demonstrates an opposition. This is probably due to the fact that our training data is much more smaller than the one used by Baidu. We employ the last line shown in the table as our final setting.

4.3 Language Model Selection

We also sampled several parameter settings to test on language model selection.

Table 3: Parameter Setting for Language Model

LM parameters		**WER metric**
α	**beam size**	
0	0	23.56
3	1	23.73
3	**10**	**19.71**
5	200	22.07

From the above table we see that language model is able to lower down the WER metric score based on DNN models. Parameter setting of the second line is applied in our final submitted demo.

5 Acknowledgement

It's been a great journey exploring the state-of-the-art speech recognition techniques and deploying it into a practical setting. We've devoted plenty of time into learning related topics as well as debugging codes and environment setting.

Here we give special appreciation to teachers and TA of this course. We couldn't reach such level without their attention and help.

References

Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al. The htk book (for htk version 3.4). *Cambridge university engineering department*, 2(2):2–3, 2006.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Dong Wang and Xuewei Zhang. Thchs-30: A free chinese speech corpus. *arXiv preprint arXiv:1512.01882*, 20