RNA

Jaime Ferrando Huertas

January 2021

Contents

1 Introduction

This report contains the individual work part for RNA course. We choose to train an automatic speech recognition system (ASR) based on neural networks.

The current ASR state of the art is a very intense fight between what is called "hybrid" models and end-to-end models. Hybrid models are an iteration of traditional ASR models that consists of two parts, acoustic model and language model. Current hybrid approaches have changed the statistics used on these traditional models by neuronal networks, so the acoustic model is an HMM where transition probabilities are calculated by an RNN and the language model are transformers. End-to-end models try to encapsulate these two parts into one single neural network.

Our chosen architecture for this work has been DeepSpeech[1], a rather old (by Deep learning standards) end-to-end model but very famous. It was originally developed by Baidu[1] and there are APIs serving it but we choose to implement it ourselves. We wrote a DeepSpeech implementation with some minor upgrades in Pytorch and run the experimentation locally in our RTX 3090. Regarding data, we will be using the most common ASR corpus, Librispeech[4].

2 Dataset

LibriSpeech is a corpus of approximately 1000 hours of read English speech with a sampling rate of 16 kHz, prepared by Vassil Panayotov with the assistance of Daniel Povey. The data is derived from read audiobooks from the LibriVox project and has been carefully segmented and aligned. The dataset contains multiple sets:

- Training: train-clean-100,train-clean-360,train-other-500
- Development: dev-clean, dev-other
- Test: test-clean, test-other

We will be using all training sets during training, a development set for early patience policy, and test sets for evaluating metrics on the final model.

To process our raw audio we will transform them into Mel-frequency cepstral coefficients[2]. Mel-frequency cepstral coefficients (MFCC) are a combination of mathematical operations to create an uncorrelated multidimensional representation of an audio signal, capturing human audible frequencies.

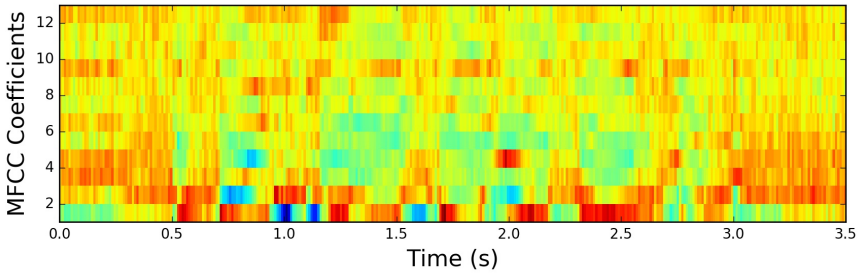

Figure 1: MFCC representation

3 DeepSpeech

3.1 Architecture

DeepSpeech is based on two main neural network modules, simulating the hybrid architecture but all in one single model. The original DeepSpeech consisted of

- 2 Convolutional Neural Networks to learn the relevant audio features.
- 3 Recurrent Neural Networks.
- A final fully connected layer is used to classify characters per time step.

We made some changes that preserve the core architecture but replace some layers with more advanced versions of the layer itself, these changes are not our invention but rather a collection of improvements seen in the ASR community. The implemented DeepSpeech consists of:

- 2 Residual Convolutional Neural Networks to learn the relevant audio features.
- 3 Bidirectional Recurrent Neural Networks (BiRNN).
- A final fully connected layer is used to classify characters per time step.

With residual connections in our CNN, we will help the model overcome the problem of gradient vanishing while BiRNN layers are a stronger RNN layer more suitable for context sequence modeling like audio signals. This architecture reminds us of traditional encoder-decoder architectures, the encoder would be our Residual CNN layers and the encoder will be our BiRNN. The model has a total of 14233053 trainable parameters and you can find a detailed version of the architecture in our appendix.

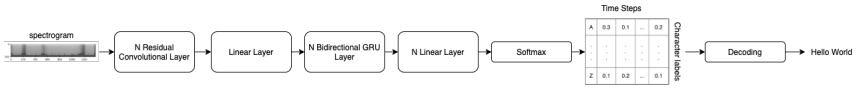

Figure 2: DeepSpeech representation

3.2 Training

For our training configuration, we choose to go with AdamW as an optimizer. AdamW is a version of Adam with an alternative weight decay algorithm. For our learning rate scheduler, we picked the OneCycleLR scheduler[6] with an initial value of 0.0005. OneCycleLR has a warm-up period that spikes up and then exponentially decreases over time. We trained with a batch size of 32 and run an evaluation on the development set at the end of every epoch. We applied what is called "patience policy" with patience 5, we waited for 5 epochs without a significant increase in our development CER results before stopping training. This policy leads us to 40 epochs of training.

3.3 Evaluation metrics

The most common metrics for ASR are Character Error Rate (CER) and Word Error Rate(WER). These metrics represent the classification error of our model in two levels, characters, and words. CER is commonly used when dealing with ASR models that don't have an extra language model at the end to fix "non-real" output words like "hoose" that is not a word in the English vocabulary but given the right context could be replaced by "house". On the other hand, WER takes more importance when these language models are present as it also helps evaluate both the performance of the ASR main system and the LM.

We will be presenting results in both CER and WER but expect WER to be worse since we are not adding any type of LM to fix DeepSpeech output.

4 Results

We present now our CER WER results on both test-clean and test-other.

Test set	WER	CER
test-clean	13.67%	4.87%
test-other	32.33%	14.95%

Table 1: Our LibriSpeech DeepSpeech results

There are not public DeepSpeech benchmarks on LibriSpeech from the original authors but we found other implementations [3] where they benchmarked on LibriSpeech. Their results were

Test set	WER	CER
test-clean	10.46%	3.39%
test-other	28.28%	12.03%

Table 2: Public LibriSpeech DeepSpeech results

Their results are better but we do get quite close to them. We know as a fact how hyper-parameter optimization can affect a neuronal network performance (especially with sizes like DeepSpeech) so we believe with proper optimization it should reach the same level of results. Unfortunately, this is out of the scope for this single experiment took three and a half days of training, we will require a GPU cluster or more time to do this hyper-parameter optimization.

5 Conclusion

This has been a good learning experience on how to implement and train big architectures compared to the ones we used in the practical assignments. Our DeepSpeech architecture has proven to work with datasets like LibriSpeech but requires more work in terms of hyper-parameter optimization. It's been great to implement theory seen in lectures like Residual-CNN and Bi-RNN layers to an existing ASR architecture.

6 Future work

As mentioned in the results section we would still have to do a full hyper-parameter optimization of our model to squeeze all possible performance and possibly reach public results. This was a first great start with ASR modeling but is not representative of the current state of the art, modern techniques such as transformers/conformers are leading and we would like to implement those as well and compare results with DeepSpeech. We also think that implementing a language model would be extremely helpful to our model predictions and is worth experimenting with in the future.

In lectures we have seen how crucial the amount of data is for neuronal network models, we have seen techniques as image data augmentation during training and their effect in increasing model performance. We would like to try something similar for our audio samples. We have done a bit of research and found SpecAugment [5] and think is very interesting to evaluate and see if our performance increases as much as seen with other data augmentation techniques in laboratory sessions.

References

[1] Awni Y. Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam

Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[2] S. Molau, M. Pitz, R. Schluter, and H. Ney. Computing mel-frequency cepstral coefficients on the power spectrum. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 1, pages 73–76 vol.1, 2001.

[3] Sean Naren. Deepspeech. https://github.com/SeanNaren/deepspeech.pytorch, 2020.

[4] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[5] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. *Interspeech 2019*, Sep 2019.

[6] Leslie N. Smith and Nicholay Topin. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

7 Appendix

```
DeepSpeech(
  (cnn): Conv2d(1, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (rescnn_layers): Sequential(
    (0): ResidualCNN(
      (cnn1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cnn2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (layer_norm1): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
      (layer_norm2): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
    )
    (1): ResidualCNN(
      (cnn1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (cnn2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (dropout1): Dropout(p=0.1, inplace=False)
      (dropout2): Dropout(p=0.1, inplace=False)
      (layer_norm1): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
      (layer_norm2): CNNLayerNorm(
        (layer_norm): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
      )
    )
  )
  (fully_connected): Linear(in_features=2048, out_features=512, bias=True)
  (birnn_layers): Sequential(
    (0): BidirectionalGRU(
      (BiGRU): GRU(512, 512, batch_first=True, bidirectional=True)
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (1): BidirectionalGRU(
      (BiGRU): GRU(1024, 512, bidirectional=True)
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (2): BidirectionalGRU(
      (BiGRU): GRU(1024, 512, bidirectional=True)
      (layer_norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
```

```
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (classifier): Sequential(
    (0): Linear(in_features=1024, out_features=512, bias=True)
    (1): GELU()
    (2): Dropout(p=0.1, inplace=False)
    (3): Linear(in_features=512, out_features=29, bias=True)
  )
)
Num Model Parameters 14233053
```