University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Piotr Ambroszczyk
Student no. 385090

Łukasz Kondraciuk
Student no. 385775

Wojciech Przybyszewski
Student no. 386044

Jan Tabaszewski
Student no. 386319

NVIDIA Deep Speech

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
dr Janina Mincer-Daszkiewicz
Instytut Informatyki

May 2019

Supervisor's statement

Hereby I confirm that the presented thesis was prepared under my supervision and that it fulfils the requirements for the degree of Bachelor of Computer Science.

Date Supervisor's signature

Authors' statements

Hereby I declare that the presented thesis was prepared by me and none of its contents was obtained by means that are against the law.

The thesis has never before been a subject of any procedure of obtaining an academic degree.

Moreover, I declare that the present version of the thesis is identical to the attached electronic version.

Date Authors' signatures

Abstract

The authors of this thesis focus on implementing scripts for training DeepSpeech2 model for Automatic Speech Recognition. We try to reproduce results obtained by Baidu Research in End-to-End Speech Recognition paper [2] using `PyTorch` framework. We also experiment with obtaining dataset for Polish language and trying DeepSpeech2 model for it. Finally, we provide fully trained models for English and Polish together with statistics about how changing hyperparameters and architecture impacts model's performance and accuracy.

Keywords

Deep Speech, ASR, Neural Networks, Machine Learning, Python, PyTorch, NVIDIA, RNN, multi-GPU, FP16

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatyka

Subject classification

D. Software

Tytuł pracy w języku polskim

NVIDIA Deep Speech

Contents

Introduction

Transcription of spoken language is a crucial problem for many areas of modern technology industry. Being able to communicate with electronic devices not only by touching them, but also by talking to them is an important goal for IT companies. Such devices are more user-friendly so it is for sure beneficial for everybody. To achieve this goal various solutions were proposed and many of then use complex algorithms (e.g. Hidden Markov Models) [1]. However it has been shown that the best accuracy can be achieved with Automatic Speech Recognition (ASR) models based on neural networks [2].

Our thesis concentrates on implementing state of the art ASR model DeepSpeech2 described in [2] and we realize it with the support of NVIDIA Corporation. Authors of DeepSpeech2 prepared their model only for recognizing English and Mandarin, so we plan to experiment with applying it to Polish language as well. We think, it is the biggest challenge, since accuracy of the model depends not only on its implementation, but also on the size and diversity of used dataset. Therefore we have to find appropriate one (paying attention to licenses and copyrights) and prepare it adequately. Large size of the dataset creates another problem – we need our model to be able to train on that data in reasonable time and then work in real time. Last but not least, in order to determine the best hyperparameters we have to run many experiments, collect their results, and finally analyze them.

In order to accomplish our goals we are going to implement DeepSpeech2 model using `PyTorch` deep learning framework, which is supported with CUDA, and is considered to be comfortable to work with. To achieve high performance system we plan to use open-source libraries prepared by NVIDIA which makes it possible to train one neural network over multiple GPUs. Another optimization which we expect to speed computations up is using half precision floating point numbers (also known as FP16) instead of single precision. When it comes to collecting datasets we assume it should be relatively easy for English as there are lots of free English utterances with transcriptions. However, for spoken corpus of Polish it may be harder - we are going to search for Polish speech collected for university programs and from audiobooks.

Structure of our thesis is the following. In Chapter 1 we introduce architecture of DeepSpeech and DeepSpeech2 models in terms of, among others, used layers, data flow and functions. After that in Chapter **??** we present applied optimizations which increased network performance. Next in Chapter 2 we describe the results of experiments on model hyperparameters. In Chapter **??** we describe how we modified and trained neural network to detect Polish language and compare obtained results with English model. Finally in Chapter 3 we summarize our experiments, show their results and present final version of the model.

We divided our work on the model into two main parts. The first one consisted of preparing appropriate datasets (for both English and Polish) and processing them to fit the model – Łukasz Kondraciuk and Jan Tabaszewski were in charge of this part. The second one consisted of implementing the model and applying GPU optimisations to it – this was the task for Piotr Ambroszczyk and Wojciech Przybyszewski.

Chapter 1

Basic model description

DeepSpeech 2 (DS2) system is a recurrent neural network trained to ingest speech spectrograms and generate a text transcription. Following description of the model architecture is pretty basic and it is based on [2]. More details can be find there. **Czy coś takiego jak wyżej w pełni wystarcza, żebyśmy mogli cytować zdania z tej pracy bez uprzedzenia?**

1.1. Input and Output specification

Let $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ...\}$ be a training set. $x^{(i)}$ is a time-series of variable length where every time-slice is a spectrogram of power normalized audio clips, so $x^{(i)}_{t,p}$ denotes the power of the p'th frequency bin in the audio frame at time t. $y^{(i)}$ is a transcription of the utterance $x^{(i)}$. One can see that $x^{(i)}$ is in fact a matrix of size depending on number of frequency bins and i'th audio clip length. We use the same number of power bins for all audio clips in dataset.

DS2 network's input is a time-series x and the output is a prediction over characters $p(l_t|x)$ for each output time-step. For English language possible values of l_t are:

- letters from 'a' to 'z';
- space;
- apostrophe;
- blank.

Adding non-letter characters allows to find word boundaries. Special symbol blank is outputted each time, when the network is unable to tell which character is most likely to occur for the current input spectrogram.

1.2. Layers

The model of the network is composed of one or more convolutional layers, followed by one or more bidirectional recurrent layers [4], followed by one or more fully connected layers. Activation function used throughout the network is the Clipped Rectified Linear (ReLU) function, given by the formula:

$$\sigma(x) = \begin{cases} 0 & \text{for } x < 0, \\ 20 & \text{for } x > 20, \\ x & \text{otherwise.} \end{cases}$$

The recurrent layers appear in a few different variants – standard recurrent layers or Long Short-Term Memory (LSTM) [5], or Gated Recurrent Units (GRU) [6]. After the recurrent layers and fully connected layers are applied, we count the output layer L as a softmax of the output of the last layer.

Softmax function $f : \mathbb{R}^k \to \mathbb{R}^k$ is defined by the formula:

$$f_i(v) = \frac{e^{v_i}}{\sum_{i=1}^{k} e^{v_j}},$$

and

$$f(v) = \big(f_1(v), f_2(v), \ldots, f_k(v)\big).$$

We basically apply exponential function to each outputed value, and then normalize these values to make sure, that probabilities sum up to 1. In our case $k = 29$, hence there is 29 possible output characters to distribute probability on (26 letters and 3 special symbols, as described in the previous section).

1.3. CTC Loss

To train a neural network we typically need a function that would tell us how good current network's output is. The lesser value this function has, the better results our model achieves. This kind of function is called a **loss function**. Usually, minimizing a value of the loss function is a main goal of the training.

Loss function used in DS2 is Connectionist Temporal Classification (CTC) [3]. To define this loss let us introduce an encoding of a text. Encoding of a given text S is done by replacing every character c in S by any number of characters c and blanks "-". Only restriction is that if there are two adjacent identical letters in S, they must be separated by a blank "-". For instance, possible encodings of *"to"* are *"-tttooo"* and *"-tttoo-o"*, but only the latter could be an encoding of the word *"too"*.
Now we say that the probability of an actual transcription is the sum of probabilities of all possible encodings of the actual transcription, that have the same length as the output of the model. The loss function here is simply the negative logarithm of this probability. Having this we can count derivatives of this loss function with respect to model output and then apply *backpropagation through time* algorithm to train the network, which is just a standard backpropagation algorithm modified for recurrent neural networks.

1.4. Training

During the training data sets are divided into batches of some certain size. Batches are fed one by one into the model and after each one CTC loss is computed and backpropagated through the network, and then weights on the layers are updated. The speed in which changes are applied to the model depends on some chosen constant – **learning rate**. All data sets make up for an epoch of training and training lasts for many epochs – until loss doesn't decrease anymore or decreases very slightly.

Accuracy of the model is measured by Word Error Rate (WER). It is a common metric of the performance of speech recognition systems. Here [8] many commercial ASR systems were compared using WER. To measure Word Error Rate, we compare the recognized word sequence with a reference word sequence by transforming the latter to the former and find:

- S – the number of substitutions,
- D – the number of deletions,
- I – the number of insertions,
- C – the number of correct words,
- N – the number of words in the reference ($N = S + D + C$).

Now the WER can be computed as:

$$WER = \frac{S + D + I}{N} = \frac{S + D + I}{S + D + C}$$

1.5. Generating transcription

In order to find Word Error Rate of the model, one has to generate some final transcription. It is generated based on both output of the model and a n-gram language model [9]. To generate the final transcription we search for transcription y that maximizes *Q(y)*, where Q is given as follows:

$$Q(y) = \log\left(\mathbb{P}_{ctc}(y|x)\right) + \alpha \cdot \log\left(\mathbb{P}_{lm}(y)\right) + \beta \cdot word\_count(y).$$

The term $\mathbb{P}_{ctc}(y|x)$ denotes the probability of y being a transcription of the utterance x as described in section 1.3 and the term $\mathbb{P}_{lm}(y)$ denotes the probability of the sequence y according to the language model. The weight α controls the relative contributions of the language model and the CTC network. The weight β encourages more words in the transcription. Both of those parameters are tunable. We use a beam search similar to one described in [7] to find the optimal transcription y.

Beam search is an algorithm that iterates over each time-step in the output of DS2 model, while remembering some constant number (let this number be *BW* – *Beam Width*) of the most probable transcriptions up to this point. When considering the next time-step, we create $BW \cdot C$ new transcriptions corresponding to appending every possible character (we denote number of them by C) to the end of each one of BW remembered transcriptions. After merging identical transcriptions and recalculating their probabilites we again keep BW best ones. In the end we have *BW most probable* transcriptions, so we just take the best one.
In order to efficiently merge transcriptions and recalculate new probabilities for them we must store probabilities $\mathbb{P}_b$, $\mathbb{P}_{nb}$, $\mathbb{P}_{lm}$ for all kept transcriptions, where:

- $\mathbb{P}_b(y,t)$ – the probability of the given transcription y at time-step t if the encoding ends with *blank*,
- $\mathbb{P}_{nb}(y,t)$ – the probability of the given transcription y at time-step t if the encoding doesn't end with *blank*,
- $\mathbb{P}_{lm}(y)$ – the probability of the given transcription y, according to the language model.

Nie podoba mi się używanie mathbb P na oznaczenie tych prawdopodobieństw, bo to sugeruje, że ono jest jakąś miarą, a jest tylko zwykłą funkcją.

1.6. Summary

To sum up workflow in DS2 is quite standard as for neural network. We preapre some data in the form described in 1.1. Next we forward propagate the data through the layers described in 1.2. After that we count CTC Loss (1.3.) and update model weights with specific learning rate (1.4.). To evaluate model we generate transcription from predictions given by the output layer using Beam search (1.5.) and count Word Error Rate metrics (1.4.). **TODO Hiperłącza**

Chapter 2

Experiments on architecture and hyperparameters

2.1. Regularization

2.1.1. Batch normalization

2.1.2. Dropout

2.1.3. L2 regularization

2.2. Automatic mixed precision

2.3. Language model and decoding predictions

2.4. Recurrent unit type

In the section 1.2 we mentioned bidirectional recurrent layers. Three most commonly used ones are vanilla Recurrent Neural Networks (RNN) [10], Long Short Term Memory Unit (LSTM) [11] and Gated Recurrent Unit (GRU) [12]. We will deliver a brief description of these units based on [2].
Let's denote by l the layer number. A bidirectional recurrent layer h^l consist of a forward recurrent layer $\overrightarrow{h^l}$ and a backward recurrent layer $\overleftarrow{h^l}$. The forward and backward recurrent layer activations for time t are computed as $\overrightarrow{h_t^l} = g(\overrightarrow{h_{t-1}^l}, h_t^{l-1})$ and $\overleftarrow{h_t^l} = g(\overleftarrow{h_{t+1}^l}, h_t^{l-1})$. In the end, we add both partial layers to get $h^l = \overrightarrow{h^l} + \overleftarrow{h^l}$.
Function g mentioned before depends on specific recurrent unit type. For vanilla RNN it is just

$$\overrightarrow{h_t^l} = f(W^l h_t^{l-1} + \overrightarrow{U^l}\overrightarrow{h_{t-1}^l} + b^l)$$

and analogically for the backward recurrent layer. Here W^l and $\overrightarrow{U^l}$ are just weight matrices, b^l is a bias vector and f is an activation function (e.g. ReLU or tanh). For LSTM and GRU function g is much more complex to simulate some kind of *memory*.
This is a short summary of our experiments:

2.5. Multi GPU scaling

2.6. Sortagrad and dataset sorting

2.7. Initialization

2.7.1. Xavier initialization

2.7.2. Random seed

2.8. Training dataset

2.9. Hyperparameters

Chapter 3

Conclusions

To sum up, we present `PyTorch` scripts for training DeepSpeech2 model for ASR. We also present already trained models for English and Polish as well as the results of our experiments justifying using specific hyperparameters and architecture solutions.

Bibliography

[1] Hannun et al. *Deep Speech: Scaling up end-to-end speech recognition*, Silicon Valley AI Lab 2014, https://arxiv.org/abs/1412.5567

[2] Baidu Research *Deep Speech 2: End-to-End Speech Recognition in English and Mandarin*, Silicon Valley AI Lab 2015, https://arxiv.org/abs/1512.02595

[3] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber. *Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In ICML, pages 369-376. ACM, 2006.*

[4] M. Schuster and K. K. Paliwal. *Bidirectional recurrent neural networks.* IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997.

[5] S. Hochreiter and J. Schmidhuber. *Long short–term memory.* Neural Computation, 9(8):1735—1780, 1997.

[6] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. *Learning phrase representations using rnn encoder-decoder for statistical machine translation.* In EMNLP, 2014. https://arxiv.org/pdf/1406.1078.pdf

[7] A. Y. Hannun, A. L. Maas, D. Jurafsky, and A. Y. Ng. *First-pass large vocabulary continuous speech recognition using bi-directional recurrent DNNs.* abs/1408.2873, 2014. http://arxiv.org/abs/1408.2873

[8] Bohouta, Gamal & Këpuska, Veton. (2017). *Comparing Speech Recognition Systems (Microsoft API, Google API And CMU Sphinx).* Int. Journal of Engineering Research and Application. 2248-9622. 20-24. 10.9790/9622-0703022024.

[9] https://web.stanford.edu/ jurafsky/slp3/3.pdf

[10] M. Schuster & K. K. Paliwal. *Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997* https://pdfs.semanticscholar.org/4b80/89bc9b49f84de43acc2eb8900035f7d492b2.pdf

[11] S. Hochreiter & J. Schmidhuber. *Long short-term memory. Neural Computation, 9(8):1735—1780, 1997* https://www.bioinf.jku.at/publications/older/2604.pdf

[12] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches* https://arxiv.org/pdf/1409.1259.pdf

All the files were downloaded on May 14, 2019