Introduction to Mozilla Deep Speech

Sep 24, 2018 Reading Time: 9 minutes Tags: [Artificial Intelligence]

Mozilla released version 0.2.0 of their open/free Deep Speech implementation a few days ago. It is the only non-proprietary system out there that reaches human-level word error rates (WER). Let us dive into it.

Contents

Introduction to Mozilla Deep Speech

Mozilla Deep Speech is Mozilla's implementation of Baidu's Deep Speech [1] Neural Network Architecture. It is designed for various reasons:

- open state-of-the-art alternative to proprietary solutions such as Amazon Alex, Google Assistant or Baidu …
- offline support
  - privacy
  - low latency
  - works in "disconnected" settings, e.g. industrial control systems that are detached for good reasons or mobile devices experiencing insufficient network coverage
- scaleable

It certainly does not match the capabilities in terms of understanding of Google Duplex but it has a decent performance. It matches human-level performance on the Librispeech ASR clean test set with a WER of 5.6 % (human performance: WER = 5.83 %). A non-technical talk by Tilman Kamp @ FOSDEM 2018 gives a brief and nice introduction to Mozilla Deep Speech.

A support channel @ Mozilla Discourse is offered for all kinds of questions about Mozilla Deep Speech).

Installing Mozilla Deep Speech

Installing Mozilla Deep Speech is a straight forward procedure using pip:

```
(env) $ pip install deepspeech
```

```
(env) $ pip install deepspeech-gpu
```

Now comes the tricky part: We have to download and unpack the fully trained models which are big (1.84 GiB):

```
(env) $ curl -OLC - https://github.com/mozilla/DeepSpeech/releases/download/v0.2.0/
(env) $ tar -xvzf ./deepspeech-0.2.0-models.tar.gz
```

The model folder contains 7 files:

```
.
├── alphabet.txt
├── lm.binary
├── output_graph.pb
├── output_graph.pbmm
├── output_graph.rounded.pb
├── output_graph.rounded.pbmm
└── trie
```

`alphabet.txt` is self-explanatory. It contains a list of all characters in the English alphabet.

`output_graph.pb` is the model as a frozen tensorflow graph.

`.rounded.` are models that use rounded weights to increase computation.

`.pbmm` are memory mapped versions of the graphs to make them more efficient.

`lm.binary` is the binary language model (1.7 GiB unpacked!!!).

`trie` is the language model as a trie.

First Steps

Let us run Deep Speech for the first time. Therefore, we should get the example files:

```
(env) $ curl -OLC - https://github.com/mozilla/DeepSpeech/releases/download/v0.2.0/
(env) $ tar -xvzf ./audio-0.2.0.tar.gz
```

The audio files are licensed under CC BY 4.0 and are a tiny selection from the LibriSpeech ASR corpus.

Deep Speech offers some input options:

```
(env) $ deepspeech -h
usage: deepspeech [-h] --model MODEL --alphabet ALPHABET [--lm [LM]]
                  [--trie [TRIE]] --audio AUDIO [--version]

Running DeepSpeech inference.

optional arguments:
  -h, --help            show this help message and exit
  --model MODEL         Path to the model (protocol buffer binary file)
  --alphabet ALPHABET   Path to the configuration file specifying the alphabet
                        used by the network
  --lm [LM]             Path to the language model binary file
  --trie [TRIE]         Path to the language model trie file created with
                        native_client/generate_trie
  --audio AUDIO         Path to the audio file to run (WAV format)
  --version             Print version and exits
```

Only 16 kHz mono input WAV files are supported.

Audio clip #1

0:01 / 0:01

```
(env) $ deepspeech --model models/output_graph.pb --alphabet models/alphabet.txt --
```

```
Loading model from file models/output_graph.pb
TensorFlow: v1.6.0-18-g5021473
DeepSpeech: v0.2.0-0-g009f9b6
Warning: reading entire model file into memory. Transform model file into an mmappe
Loaded model in 0.591s.
Running inference.
experience proves this
Inference took 2.084s for 1.975s audio file.
```

Audio clip #2

0:02 / 0:02

```
(env) $ deepspeech --model models/output_graph.pb --alphabet models/alphabet.txt --
```

```
Loading model from file models/output_graph.pb
TensorFlow: v1.6.0-18-g5021473
DeepSpeech: v0.2.0-0-g009f9b6
Warning: reading entire model file into memory. Transform model file into an mmappe
Loaded model in 0.126s.
Running inference.
your power is sufficient i said
Inference took 2.654s for 2.590s audio file.
```

Audio clip #3

0:00 / 0:02

```
deepspeech --model models/output_graph.pb --alphabet models/alphabet.txt --audio au
```

```
Loading model from file models/output_graph.pb
TensorFlow: v1.6.0-18-g5021473
DeepSpeech: v0.2.0-0-g009f9b6
Warning: reading entire model file into memory. Transform model file into an mmappe
Loaded model in 0.128s.
Running inference.
```

```
why should one hall t on the way
Inference took 2.874s for 2.735s audio file.
```

It should be `why should one halt on the way`. Hence, let us see if using language models leads to any improvement:

```
(env) $ deepspeech --model models/output_graph.pb --alphabet models/alphabet.txt --
Loading model from file models/output_graph.pb
TensorFlow: v1.6.0-18-g5021473
DeepSpeech: v0.2.0-0-g009f9b6
Warning: reading entire model file into memory. Transform model file into an mmappe
Loaded model in 0.121s.
Loading language model from files models/lm.binary models/trie
Loaded language model in 16.9s.
Running inference.
why should one halt on the way
Inference took 5.079s for 2.735s audio file.
```

It certainly does ;). However, it almost took double the time to run it. It is possible to run it in real-time on a CPU without using the binary language model. However, with the binary language model it slows down quite a lot. *(Using the trie language model only, it will produce `why should one hallt on the way`).*

Mozilla Deep Speech for Applications

Client-side only

My initial idea was that we can build a pure client-side web application or mobile application that uses Mozilla Deep Speech. The market lacks of a useful speech to text app. Currently, the main problem seems to be that the language model (binary) is of size 1.7 GiB. This is certainly required for useful speech to text conversion. However, it is too much to ship a 2+ GiB mobile app or a website that has to load 2 GiB of data first.

Node.js

Since there is native node.js support, this might be a feasible intermediate solution.

IBus Plug-in

Michael Sheldon build a lBus plug-in that uses gstreamer. Certainly, this is a useful approach. However, it looks and feels still very “pre-alpha”.

Model Architecture and Hyperparameters

Earlier this year, I wrote about model architectures and RNN units for speech recognition.

If I understand the model correctly, then it consists of 3 dense layers a bi-directional RNN cell (2 layers) and an output layer (dense). Each of these layers uses 2048 hidden units.

The architecture looks roughly like this:

26 MFCC features -> $\rightarrow$ Dense Layer (2048 units) $\rightarrow$ Dropout 0.05 $\rightarrow$ Dense Layer (2048 units) $\rightarrow$ Dropout 0.05 $\rightarrow$ Dense Layer (2048 units) $\rightarrow$ Dropout 0.05 $\rightarrow$ Bi-Directional LSTM (2048 units per direction) $\rightarrow$ Dense Layer (2048 units) $\rightarrow$ Output

All dense layers except the outlayer use a clipped ReLU activation function which is clipped at 20 ($min(ReLU(x), 20)$). Dropout is used for feed-forward only. *There is an additional Dropout of 0.05 defined but apparently not used.*

LSTM (Long Short-Term Memory) cells

As a brief reminder: A LSTM consists of 3 gates: input gate, forget gate and output gate. It is possible to use individual activation functions (σ) for each gate in theory. However, sigmoid activation functions are used for all three gates usually.

They are calculate as follows:

Forget gate f:

$$f_t = \sigma_f(W_f x_t + U_f h_{t-1} + b_f)$$

Input gate i:

$$i_t = \sigma_i(W_i x_t + U_i h_{t-1} + b_i)$$

Output gate o:

$$o_t = \sigma_o(W_o x_t + U_o h_{t-1} + b_o)$$

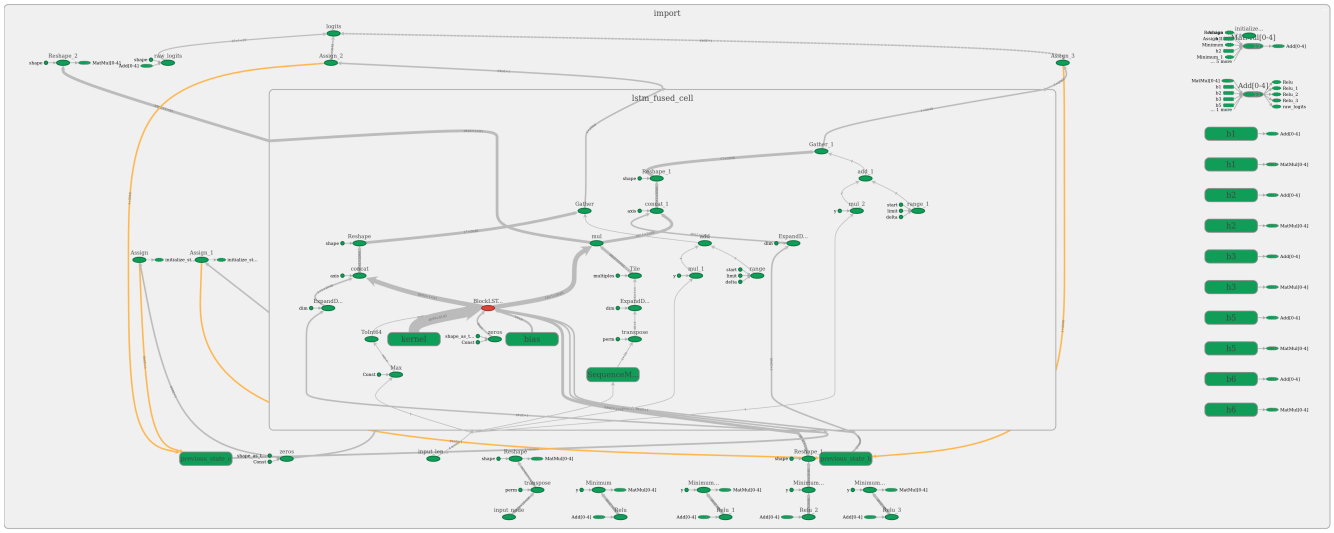
What makes the LSTM special: it uses future as well as past data in the previous/next step. Therefore, we have to calculate h_t:

$$c_t = f_t \ .* \ c_{t-1} + i_t \ .* \ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$h_t = o_t \ .* \ \sigma_h(c_t)$$

. $*$ is the element-wise product (aka Hadamard product ($\circ$)); $tanh$ are used for σ_c and σ_h

BTW, the `output_graph.pb` looks as follows:



CommonVoice Corpus

The Mozilla Deep Speech project is accompanied by the CommonVoice project. It is an approach to democratize the building of a large speech to text corpus by enabling everyone to participate by either donating voice or verify translations. It certainly has the advantage that it covers many accents. A nice introduction to this is the FOSSASIA 2018 talk by Sandeep Kapalawai at the Lifelong Learning Institute, Singapore.

References

[1] Hannun et al. (2014): Deep Speech: Scaling up end-to-end speech recognition. arXiv: 1412.5567
