PROJECT REPORT ON SPEECH RECOGNITION
BY TUNDE AJAYI
AFRICAN INSTITUTE FOR MATHEMATICAL SCIENCES, GHANA

Date: July 3, 2020.

Aim: To build a small Automatic Speech Recognition (character level) model

Tools, Libraries, Framework: Google Colab notebook, Ligaikuma mobile app, Pytorch

Procedure:

- Create dataset (2h recording)
- Pretrain a CPC system
- Finetune CPC on train data using CTC
- Compute CER

Dataset

The dataset is an approximately 2-hour recording of speech read from a Hausa Language corpus. Lig-aikuma, an android mobile app, was the tool used to create the dataset. It had various functionalities such as recording, respeaking, elicitation by text, elicitation by image etc. A text corpus formatted sentences were fed into the app. Using elicitation by text mode and after filling the necessary metadata, the prompted text was read and captured by the app. The entire approximately 2h recordings made up the first part of the dataset. The second part also followed a similar pattern. The source of the recorded speech was an image dataset, in which its events were described and recorded. The elicitation by image mode was adopted in this part, with each prompted image being described and captured.

The 2h recordings were later split into train/val/test sets. The full dataset was initially split into 1h train and 1h test set, but the train set was further split into training set (80% of original train set) and validation set (20% of the original training set).

Link to the github repo containing the dataset:

https://github.com/tunde99/AMMI-2020-SPEECH-COURSE

Pretraining CPC System

A Contrastive Predictive Coding (CPC) system was built to have an encoder and LSTM context recurrent neural network. The CPC model has the input of the encoder, having taken the representation of the input data, being fed to the context network. This in turn is passed into another network that takes care of the prediction. The CPC uses a contrastive loss that distinguishes between a true audio sample from negatives. The CPC loss function was used to obtain the average loss and average accuracy of the model. This model was then trained, with its parameters and weights stored for use in other downstream tasks.

Finetuning CPC using CTC

Connectionist Temporal Classification (CTC) was an algorithm that was used to train in this project. It is an alignment-free algorithm that takes into account the output prediction from the acoustic model to determine the appropriate mapping of input features to true labels. Bi-direction RNN (LSTM) was used to score characters for each observed audio frame, before finally applying softmax, which outputs probability distribution for characters in each time step. This can be seen in the phone classifier function of the project notebook. The CTC loss aided in finding the likely word sequence, searching different path combinations. The sum over all paths that generate the same word sequence was taken.

Computing CER

Character Error Rate (CER) was adopted in this project as an evaluation metric for the project. The goal is such that given a generated sequence and reference sequence, we want to measure how correctly we did on generated sequence. Measurement of minimum edit distance of predicted sequence to target sequence was computed, then normalised by the referenced sequence. The get_per function was used here. The output of the beam search decoder (being the generated sequence) was also used.The ref and target sequences were paired in a list with a library performing a parallelised operation on the inputs that gives a better error rate (the lower the better).

Results/Discussion

The result of training can be found at the *appendix* section of this report. Training took time, a contributing factor being that the get_per function was evaluated at each time step.

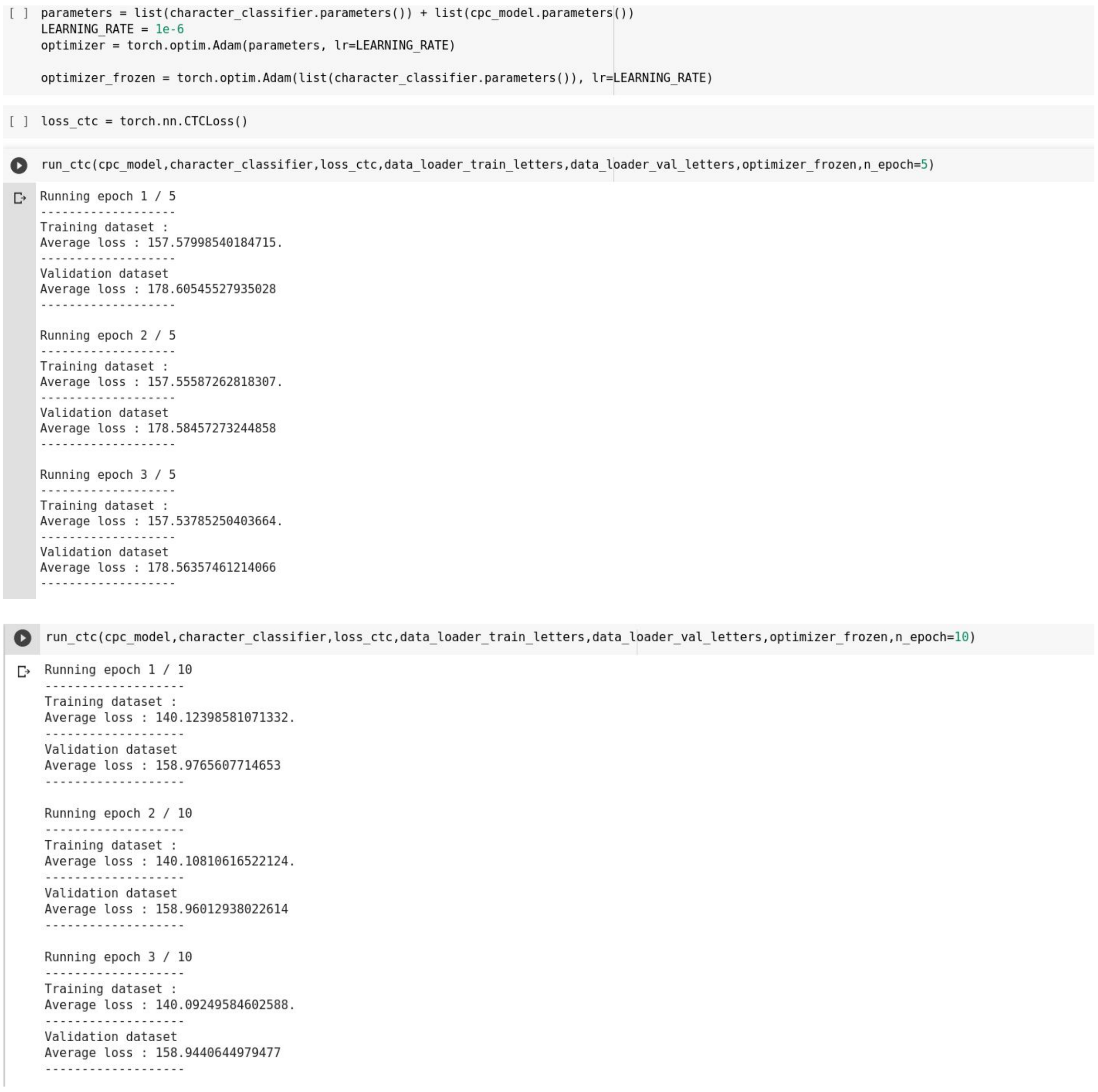
Summary/Conclusion/Future work

The project has been an eye-opener. It has helped me to fully grasp the concepts taught in the course. I understand how I can better capture raw speech from text and build a model that would evaluate the dataset. Given time, I would consider integration of a Language model and better beam search implementation, which is a good requirement for reducing the search space.

Reference

Jacob Eisenstein (October 15, 2018): Natural Language Processing

Appendix



```
[ ] parameters = list(character_classifier.parameters()) + list(cpc_model.parameters())
    LEARNING_RATE = 1e-6
    optimizer = torch.optim.Adam(parameters, lr=LEARNING_RATE)

    optimizer_frozen = torch.optim.Adam(list(character_classifier.parameters()), lr=LEARNING_RATE)
```

```
[ ] loss_ctc = torch.nn.CTCLoss()
```

```
run_ctc(cpc_model,character_classifier,loss_ctc,data_loader_train_letters,data_loader_val_letters,optimizer_frozen,n_epoch=5)
```

```
Running epoch 1 / 5
-------------------
Training dataset :
Average loss : 157.57998540184715.
-------------------
Validation dataset
Average loss : 178.60545527935028
-------------------

Running epoch 2 / 5
-------------------
Training dataset :
Average loss : 157.55587262818307.
-------------------
Validation dataset
Average loss : 178.58457273244858
-------------------

Running epoch 3 / 5
-------------------
Training dataset :
Average loss : 157.53785250403664.
-------------------
Validation dataset
Average loss : 178.56357461214066
-------------------
```

```
run_ctc(cpc_model,character_classifier,loss_ctc,data_loader_train_letters,data_loader_val_letters,optimizer_frozen,n_epoch=10)
```

```
Running epoch 1 / 10
-------------------
Training dataset :
Average loss : 140.12398581071332.
-------------------
Validation dataset
Average loss : 158.9765607714653
-------------------

Running epoch 2 / 10
-------------------
Training dataset :
Average loss : 140.10810616522124.
-------------------
Validation dataset
Average loss : 158.96012938022614
-------------------

Running epoch 3 / 10
-------------------
Training dataset :
Average loss : 140.09249584602588.
-------------------
Validation dataset
Average loss : 158.9440644979477
-------------------
```