Advanced Deep Learning Assignment 2 pwn274

1 The Implementation

1.1 Data Generation

After having tried different generation-techniques i ended up with the following: I picked out all the possible true words and as many random false words in the same category of lengths.

I choose to make completely separate training, test and validation. Given that we had so little data in the main test-set (lengths below 20), I choose to make a validations set on data-points beyond even the test-set (that is words with lengths between 50 and 70). This allowed me to get a continuous estimate for how well the models were generalizing to larger input-sizes.

1.2 The models

Both models were initially implemented using their respective PyTorch-layers. The final hidden layer of each were then fed into a linear layer of two neurons for classification. Neither converged to any meaningful classifier. After having spent far too much time on this assignment, in a moment of pure apathy, I copy-pasted the simplest Keras LSTM model into a new notebook. This converged instantly. All further work were therefore done on this.

1.3 Hyper-parameter search

I have spent multiple days worth of compute-hours on hyper-parameter searching for both the RNN and LSTM. As it was only the LSTM i could manage to get working, the model was trained using multiple learning rates and epoch numbers. A learning rate of 0.01 and 200 epochs seemed to give the best trade-off between convergence-time and final F1-accuracy on the validation set.

2 The Results

Given that we are doing 'Deep Learning' on a dataset of 10 positively labeled data points, there were some hiccups in making the models converge.[1] Therefore only results from the trained LSTM is shown.

All models were run thrice times to make for some semi-valid statistical conclusions to be made.

As the F1-score is undefined for samples without the ability of getting true positives, the x-axis is only every 2nd or every 3rd word-length. I have only plottet from length 20+, since the accuracies for short members of the language of course are impeccable, as the models have seen them countless times. The final results can be seen here, where it is clear that the 2-letter words are easier to classify than their more complex counterparts. This might have something to do with the even lower number of true-labeled data-points:

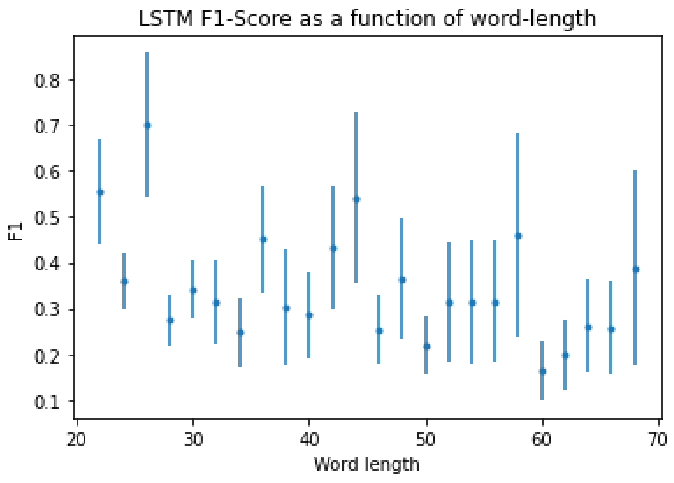

Figure 1: For members of $a^n b^n$

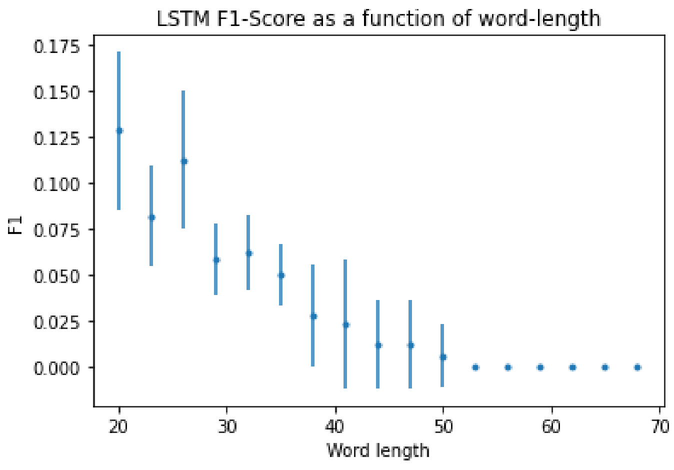

Figure 2: For members of $a^n b^n c^n$

[1] I mean, this is barely enough data for a good linear fit