Deep Learning Course

Native Language Identification using RNN

Gilad Eini and Liat Nativ

**Introduction**

Native language identification is the task of determining the native language of the author, given only a text in a foreign language. The task was introduced by Koppel et al. (2005) has gained much popularity, typically aimed to the language of learners. However, this task is relevant in the much more challenging context of advanced non-native speakers, especially since while English language dominates the internet, there are far more non-native English speakers than native speakers.   
This Project focuses on the task of native language identification of highly fluent speakers, using a corpus of *Reddit* posts in which the native language of the author has been accurately annotated (Rabinovich et al. 2018). Based upon this corpus our task is to identify the native language of non-native authors from 5 different counties of origin.  
We obtained these results:  
Accuracy XXXXXXXXXXXXX  
Recall XXXXXXXXXXXX  
…

DATA

**Dataset**

We used the Reddit dataset released by Rabinovich et al. (2018)   
Reddit is a popular online community consisting of thousands of forums in a wide range of topics. The dataset includes Reddit posts whose content is generated by users specifying their country as a flair (metadata attribute). We selected 5 native languages – represent here as the countries of origin - with a large amount of data for our dataset: USA, Germany, Turkey, France, Russia. We sampled an equal portion of sentences for each class (100K sentences), dividing each to (90%) training set and 10% test set. We further divided the training set to (90%) training set and 10% dev (validation) set. Our resulting dataset is 81000, 10000 and 9000 sentences for the training test and dev sets respectively.

**Pre-processing** & data representation  
We performed some clean-ups and pre-processing aimed at getting better classification results. We removed any non-alphanumeric characters from the text, except for punctuation(dots and commas) that might be meaningful for determining the author's native language.   
We also restricted the length of the sentences, getting best results with sentences from 25 to 150 tokens length. Also, to avoid sentences like: “d d d d d d d d d … d”, we ignored sentences that contain the same token 20 time or more.  
We shuffled the sentences in all 3 groups to prevent any bias.  
To create a suitable machine-learning representation we converted each word to a vector of real numbers using word embeddings. We used GloVe pre-trained word embeddings, based on Wikipedia (2014) containing 6B tokens, 400k vocabulary[[1]](#footnote-1). We got best results when using 300-dimension word vectors.

**Model if we can add drawing**  
We used bi-directional RNN. We have 2 multi Rnn Cell that contain 3 GRU cells for each directions, forward and backward. Each of the multi rnn cells is wrapped with a dropout layer. Those 2 multi cell, togher with the embbeding of the data, were given as arguments to the bi-directional RNN. As a loss function we used the mean of softmax cross entropy. As an optimizer we used Adam optimizer with default parameters as they gived the best results.

Evaluation methodology

To avoid overfitting, we evaluated the training op(weight updates) based only on train data, and evaluated in each epoch twice(middle of the epoch and at the end) on VALIDATION data. After training was done, we ran a Test on the model using clean 1 time test data. As the results will show, our model generalize well and we will show very close results for best model saved and the testing accuracy.   
**Experiments and improvements**   
Loss function  
LSTM cell (basic, gru)  
# of hidden units  
Optimizers  
Regularization  
Learning Rate  
Batch Size  
Embedding dim size  
dataset size  
sentence length

To get the ball rolling, we looked online for an Rnn network that solves a similar problem. We found a project that classified crime description to a category([link](https://github.com/jiegzhan/multi-class-text-classification-cnn-rnn)). As a model it’s used static one direction rnn. After doing all the modification for out task, we started testing the model with different parameters. After several experiment, and glass ceiling of 53% accuaricy, we decided to build a new bi-directional model from scratch.

After building the new bi-directional model, we conducted approximately 40 experiments.

The starting paramets:

Usually 10 epochs, batch size 100, hidden cells in a GRU cell 64, embedding dimension 50, data set contains 50K lines for each class (250K total)

* The static rnn previous model achieved ~53%, this became our new baseline(maybe mention the original baseline is 20%).
* The new model built was with LSTM cells. We didn’t budge from the baseline no matter what we tried until we replaced the cells with GRU cell type. The change was dramatic.
* To start with, we fiddeled the learning rate(we will call it ‘lr’) of the adam optimizer. We tried the dynamic approach. We initialize the lr to 0.01 and decreaced it by epochs or by achievements. We didn’t made any progress. We changed the initial lr to be 0.001 from 0.01. That change improved our results to ~53.50 on train and test. Later on we figured out there is a mechanism that changes the learning rate. It’s called tf train exponential decay. <insert here the results of using it>
* Next, we fiddled with the batch size. We found the best results are with batch size of 200 sentences. We tested 50,100,200,300.
* We proceeded with finding the best number of hidden units to each cell. We doubled the hidden units to 128 and achieved 1% improvement. Increasing more to 200 and 300 units, didn’t neccerily improved our results.
* In the embedding file we had 4 embedding dims: 50,100,200,300. We decided to test the 100 dim and a big 2% improvement achieved. We reached a milestone of ~56%. So we took it a step further and used the biggest embbeding dim we had which was 300. This did improve our results.
* We tried changing adam optimizer with adaGrad and rmsProp. adaGrad failed completely and rmsprop got clost to adam’s results but still came short.
* Then we added An initializer for our weight matrix . we used Xavier initialize ([link](https://www.tensorflow.org/api_docs/python/tf/contrib/layers/xavier_initializer)). Results were improved a bit.
* We added another layer to each direction, meaning each direction had a multi cell that was compiled from 2 GRU cells with 300 hidden units. We achieved 57.67% on train and 57.2% on test. Since the improvement was great, we added a third layer to each direction and achieved very close results. We test as much as 7 layers in each direction and didn’t reach better results under this setting.
* We thought to enlarger our data file. So we created a new one containing 100K lines per class bringing to a total of 500K lines. This didn’t improve our results but reassured us our model is legit.
* We then want to remove some of the restriction on the data and dropped the boundaries to 5-100 (from 25-150) tokens. The accuracy dropped significantly.
* We tried to regulization see results in excel #33-35
* We tried 1 more loss functions. We changed our mean cross entropy with minimum squared loss. The results were very close
* After discussing with Alaa, he advised us to change the dropout cell wrapper from being applied on the multi cell to being applied on each cell separately. The results slightly improved/
* We added attention cell wrappers to each cell. <insert here the results>. We also tried to apply the attention with the dropout. <insert here the results>.

**Summary**

We started with a baseline of 20%. After experimenting with the static model we reached 53% which became our new standard. Then we built a new dynamic bi-directional model and reach as high as 57.457% in training and 57.252% in testing. The parameters for that model were <insert here>

**Results**Accuracy & other measures   
Learning Graph

1. https://nlp.stanford.edu/projects/glove/ [↑](#footnote-ref-1)