Language Identification (Sentence Classification)

Co-authors: Jordi Vaccher[1]*, Daniel Navarrete*[1]

[1]Master in Advanced Telecommunication Technologies
ETSETB - Universitat Politècnica de Catalunya
jordi.vaccher@estudiantat.upc.edu, daniel.navarrete.jimenez@estudiantat.upc.edu

1. Introduction

This report shows the study and development of different models for language identification. We are comparing the performance of a RNN baseline system with other models for the same task. We will study modifications of the basic RNN model, as well as other DNN architectures

2. Hyper parameter optimization

In this section we have studied some parameters of the model in order to outperform the baseline. We have played with the embedding size, RNN hidden size, the batch size and the optimizer. Regarding the optimizer we have tried with an optimizer called Adabound which tries to performs in the half way between SGD optimizer and the Adam (default) optimizer. The modified hyper parameters are showed in the following table:

Table 1: *Hyper parameter optimization*

Value
Batch size = 1024
Hidden size=512
Embedding size =256

We have obtained good results modifying the batch size, both in training time and in performance. The embedding size and the hidden size improve the result slightly but not significantly. Regarding the optimizer, after testing several we conclude that the ADAM optimizer is the best

In the following tables we present the obtained results with each optimizer.

Table 2: *Adam Optimizer*

Value
Train Accuracy = 0.996
Dev Accuracy =0.937
Train Final Model Accuracy = 0.99719

Finally we have tried to add more layers to our LSTM architecture and to change LSTM for GRU.

Table 3: *Adabound Optimizer (Learning Rate1: initial=1e-3, final=0.1 - Learning Rate2: initial=1e-2, final=0.1)*

Value
Train Accuracy (Learning Rate1)= 0.944
Train Accuracy (Learning Rate2) = 0.948
Dev Accuracy (Learning Rate1)= 0.921
Dev Accuracy (Learning Rate2) = 0.914
Train Final Model Accuracy (Learning Rate1) = 0.94709
Train Final Model Accuracy (Learning Rate2) = 0.94560

Table 4: *SGD Optimizer*

Value
Train Accuracy = 0.993
Dev Accuracy =0.930
Train Final Model Accuracy = 0.99154

The following table shows the obtained results with both (LSTM 2 Layers and GRU) experiments):

Table 5: *LSTM 2 Layers - GRU 1 Layer*

Value
Train Accuracy LSTM 2 = 0.995
Train Accuracy GRU = 0.98.2
Dev Accuracy LSTM 2 =0.930
Dev Accuracy GRU =0.926
Train Final Model Accuracy LSTM 2 = 0.99547
Train Final Model Accuracy GRU = 0.98161

3. Combination of max-pool layer with a mean-pool layer

We have concatenated the outputs of the two layers. We have achieved a slightly improvement compared with the previous model
We have initialized the pooling layers parameters as are showed in the code of the Figure 1:

We have achieved the following results:

```
        self.h2o = torch.nn.Linear(hidden_size, output_size)
        self.avgpool = torch.nn.AvgPool1d(2, stride=2, padding=0, ceil_mode=False, count_inc
lude_pad=True)
        self.maxpool = torch.nn.MaxPool1d(2, stride=2, padding=0, dilation=1, return_indices
=False, ceil_mode=False)
```

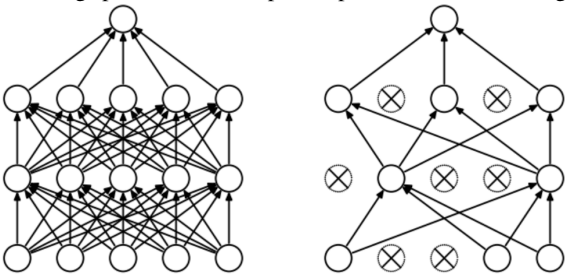
Figure 1: *Pooling Layer*

Table 6: *Train and validation accuracy results*

Accuracy
Train accuracy = 0.999
Dev accuracy = 0.940
Train Final Model Accuracy = 0.99867

4. Dropout layer

In this section, we study the addition of a dropout layer after the pooling layer implemented in the previous experiment. We used a dropout probability of 20 per cent. Dropout layer freeze some weights in the training process and helps to prevent the overfitting.



(a) Standard Neural Net (b) After applying dropout.

The results are showed in the Table 7

Table 7: *Train and validation accuracy results*

Accuracy
Train accuracy = 0.999
Dev accuracy = 0.940
Train Final Model Accuracy = 0.99820

5. Convolutional Neural Networks

In this section it is proposed to use other kind of DNN architectures such as convolutional neural networks.
In the Figures 3 and 4 it is showed the code and the architecture of our CNN.
By applying these architecture we are trying to improve the RNN models, decreasing the vanishing problems but also the time consumption. As we found using GRU models, they spent less time training the model but with CNN model we have realized that less time consumption are spent.
By the other hand the obtained results for our architecture is not greater than the best LSTM model in the InClass Kaggle competition test-set.

In the following table you can see the obtained results.

```
        self.conv11 = torch.nn.Conv1d(embedding_size, int(embedding_size*2), 2, stride=2, pa
dding=pad_idx)
        self.conv12 = torch.nn.Conv1d(int(embedding_size*2), int(embedding_size*2), 2, strid
e=2, padding=pad_idx)
        self.conv21 = torch.nn.Conv1d(int(embedding_size*2), int(embedding_size*4), 2, strid
e=2, padding=pad_idx)
        self.conv22 = torch.nn.Conv1d(int(embedding_size*4), int(embedding_size*4), 2, strid
e=2, padding=pad_idx)
```

Figure 2: *Code: Convolutional Neural Network Architecture.*

```
        encoded = self.embed(input)
        # T x B x E
        paddedInputTensor =  encoded.view(encoded.size(0),  encoded.size(1), encoded.size
(2)).transpose(0, 2)
        paddedInputTensor =  paddedInputTensor.transpose(0, 1)
        # T x E x B
        encodedTensorLayer1 = F.relu(self.conv11(paddedInputTensor))
        encodedTensorLayer1 = F.relu(self.conv12(encodedTensorLayer1))
        encodedTensorLayer1 = F.max_pool1d(encodedTensorLayer1, 2, stride=2, ceil_mode=True)

        encodedTensorLayer2 = F.relu(self.conv21(encodedTensorLayer1))
        encodedTensorLayer2 = F.relu(self.conv22(encodedTensorLayer2))
        encodedTensorLayer2 = F.max_pool1d(encodedTensorLayer2, 2, stride=2, ceil_mode=True)

        outputTensor = encodedTensorLayer2.transpose(1, 2)
        outputTensor = outputTensor.contiguous().view(outputTensor.size(0), outputTensor.siz
e(1), outputTensor.size(2))
```

Figure 3: *Code: Convolutional Neural Network Architecture.*

Table 8: *Train and validation accuracy results*

Accuracy
Train accuracy = 0.983
Dev accuracy = 0.827
Train Final Model Accuracy = 0.983347

6. Conclusions

- We have seen that by modifying the hyperparameters we can slightly increase the results of the base system. By modifying the batch size we can speed up the training time obtaining very similar and even better results. Adam is the optimizer that works best, obtaining better results and in less time than other optimizers such as SGD or Adabound.
- Increasing the Layers of the LSTM we did not improve the result. It might makes the vanishing problem greater than before.
  Using the GRU architecture we found that the time consumption decreases a lot, in comparison with the LSTM.
- Adding max and mean - pooling increases the baseline results. We used them to train a model with our best hyper parameters and we have obtained our best result.
- Theoretically using Dropout we should avoid overfitting. We can see that our model does not overfit at all, such as the other models do. But it does not outperform our best model.
- With the CNN model we have obtained very good results in the training dataset but when doing the validation we have only achieved 82 per cent. This clearly indicates an over-fitting of the network. We might could improve the architecture of the CNN but it is already difficult to achieve the LSTM result with it.
- As we can check in our results, the best model is the one that has dropout with max and mean pooling layers. But

it is not reflected with the test set of the InClass Kaggle competition.

7. References

ICLR 2019 — ‘Fast as Adam Good as SGD’ — New Optimizer Has Both
- https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34