Assignment 1: Neural Classification with Bag of Words

Filip Stefaniuk
filipste@student.matnat.uio.no

September 13, 2018

1 Introduction

In this assignment I had to build a neural network calssifier, that based on text of article represended as bag of words, predicts its source. I was working with "*The Signal Media One-Million News Articles Dataset*". To create and train models, I used Keras library with tensorflow backend. Most of the examples, I trained on my local machine since it wasn't computationaly heavy task. However, the code also runs on Abel. One of the main goals of this assignment was to test different hyperparameters. To make it easier, my program parameters (including model architecture) are parameterizable by configuration json files.
In this report I present only partial statistics for each experiment, but all required metrics for each of the experiments are available in the repository. Code that creates plots is in jupyter notebooks. The code, notebooks, results of experiments, dataset, instructions how to run program and latex source code for this report are available in github repository[1].

2 Data analysis

The dataset was provided as a gzipped tab-separated text file, with one line corresponding to one news article. Texts were already lemmatized and POS-tagged, and the stop words were removed. I have implemented module that is responsible for all data preprocessing. It splits dataset, infers a common dictionary, extracts BOW and label for each document. Size of BOW vector is parameterizable and it uses n most common words. Labels are one hot encoded.

2.1 Splitting the dataset

Dataset needed to be split into three parts: training, developement and test. To do that, I used popular scikit-learn function *train_test_split()*, that shuffles dataset based on privided seed for random generator and then splits it into two parts. I use it two times: to split dataset into training and test part and to take part from training part for developement. Both percentages (developement and test) as well as the seed are parameterizable with default values accordingly 0.1, 0.1 and 123. So by default dataset is split roughly in the proportion (80/10/10). Further experiments were prformed with default values for those parameters.

It is important to check distibution of labels since it should be similar in all parts of the dataset. Morover it is useful to check if the classes are balanced. It turns out that while most of them have similar number of examples, one (*myinforms*) has significantly more examples than the others. To deal with that problem, during training I used weighted loss function.

[1] https://github.uio.no/filipste/INF5820-Assignment1

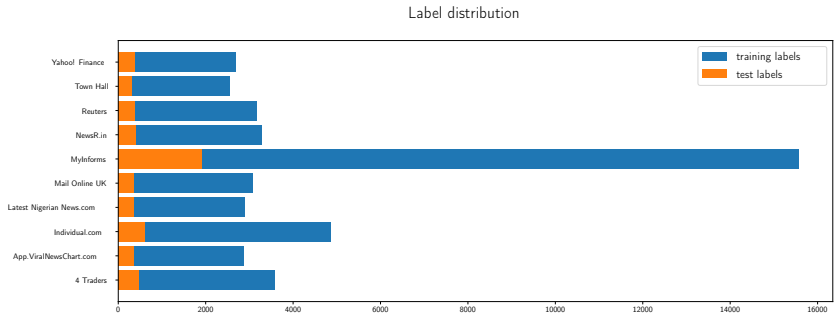

Figure 1: Distribution of labels in all three parts of the dataset using seed 123 for random number generator.

2.2 Word distribution

I run tokenizer on training part of the dataset and infered a common dictionary for all documents. Then I checked the word distributions. I have tested two cases: words with and without POS tag concatenated to them. Here are the numbers of occurances for the 10 most common words in the dataset. It is interesting to look at the differences, for example word report is very common but when with pos tag is split into two tokens: report as noun and verb and does not appear in the top 10 words.

	word	count
0	say	65933
1	company	49392
2	new	41332
3	year	37954
4	market	31059
5	also	29244
6	make	28596
7	include	27673
8	share	26650
9	report	24885

Figure 2: No POS tags.

	word	count
0	say_vb	65864
1	company_nn	49392
2	year_nn	37954
3	market_nn	30480
4	also_rb	29064
5	make_vb	28501
6	include_vb	27651
7	new_jj	27539
8	time_nn	22642
9	share_nn	21308

Figure 3: With POS tags.

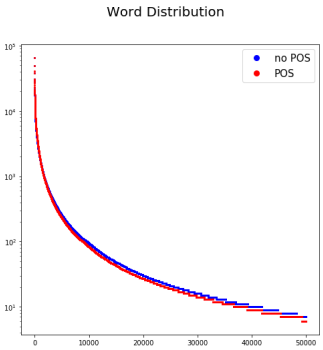

Figure 4: Word occurances.

The word counts in dataset drops extremely quickly. We can see that only 10000 words occurs more than 100 times. I have tested different sizes of input vector, using simple network with two hidden layers of 32 neurons, training for 15 epoch and saving the model that had the best loss on validation set. Increasing the size of input vector up to 2000 significantly improves the results.

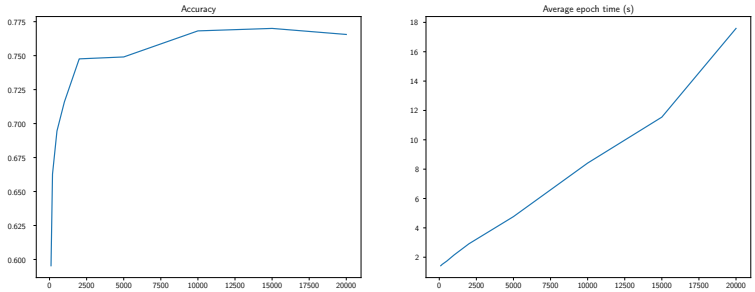

Figure 5: Results of training simple neural network classifier on different input sizes.

2.3 Feature engeenering

I have decided to test different variants of BOW:

- **binary** - 1 if word is in document 0 otherwise.
- **count** - for each word, number of occurances in document
- **tfidf** - term frequency-inverse document frequency

I have tested all three variants with both words only and words with POS tag on my final neural network model. The best performance was achieved with binary BOWs with POS tags.

3 Neural network classifier and hyperparameter tuning

I have tested number of different hyperparameters which i describe in details below. If not stated otherwise, the default architecture is neural network with three hidden layers with sizes 256, 128 and 64, input size of 2000. Trained for 10 epochs. Default loss function is categorical crossentropy, adam optimizer and relu activation function.

3.1 Optimizers

I have tested the popular optimizers: **sgd**, **RMSprop** and **Adam**. As seen below, simple sgd performs really poorly compared to other two that use adaptive learning rate.

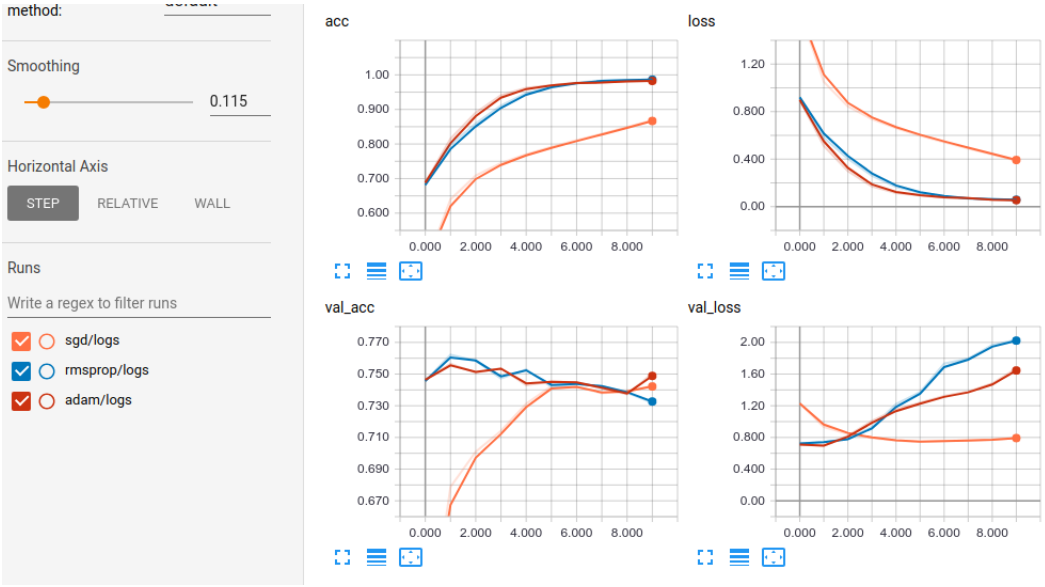

Figure 6: Training neural network with different optimizers.

3.2 Activation function

The activation functions that I have tested are: **sigmoid**, **tanh** and **relu**. Again, network learns much faster when using relu activation. This is due to the fact that with other two neurons are more prone to saturate.

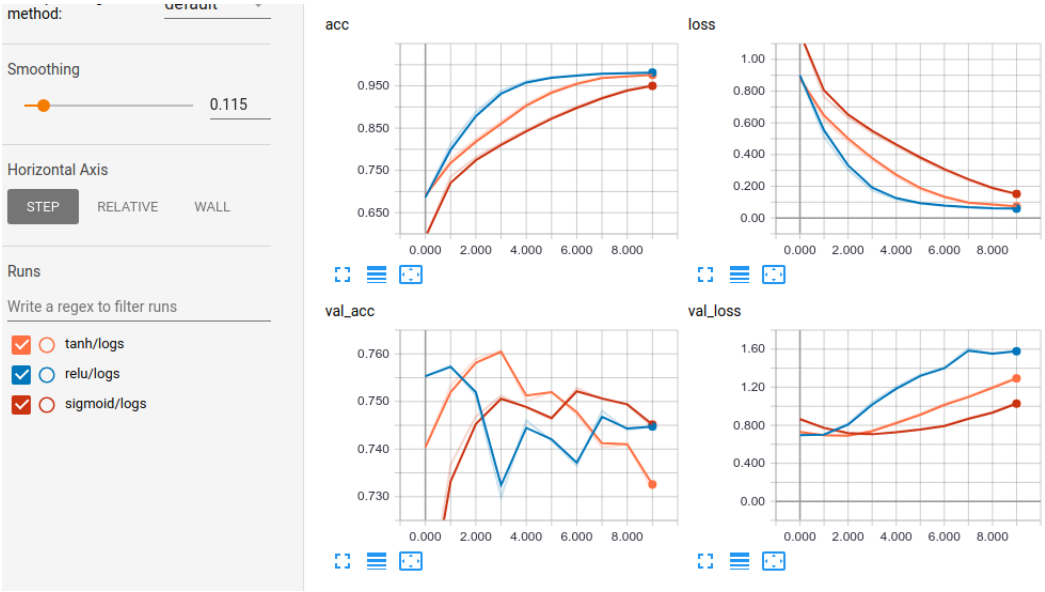

Figure 7: Training neural network with different activation functions.

3.3 Number of layers

To test how network behaves with different number of layers, I trained models with layers of size 512. As input, I also took a vector of size 512 not to bias time measurments with large matrix multiplications. As shown on charts below, accuracy slowly increases with number of layers and then drops. This is probably due to overfitting since it's common behaviour with large networks. Time increases linearly since with each new leayer I add new matrix multiplication of the same size.

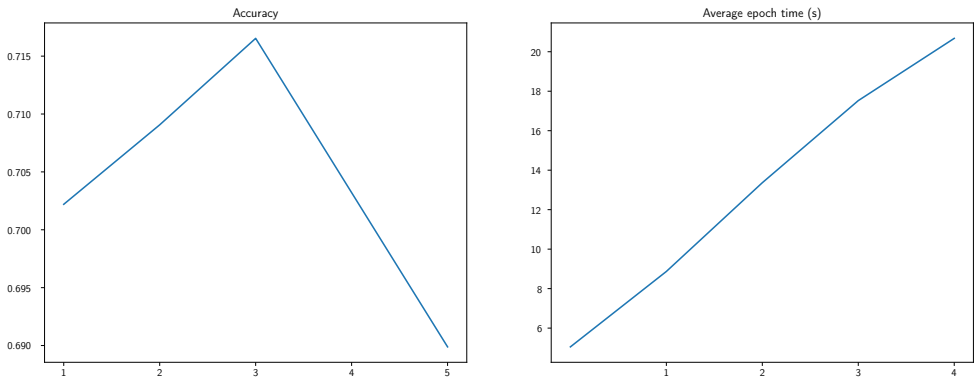

Figure 8: Results of training neural network classifier with different number of layers.

3.4 Loss function

For the loss function I have tested only two values: **categorical crossentropy** and **mse**. Categorical crossentropy which is a goto loss function when training multiclass classification behaves really well. Using mse only slowed training process, as seen below accuracy on training set achieved after one epoch using crossentropy is achieved after four using mse.

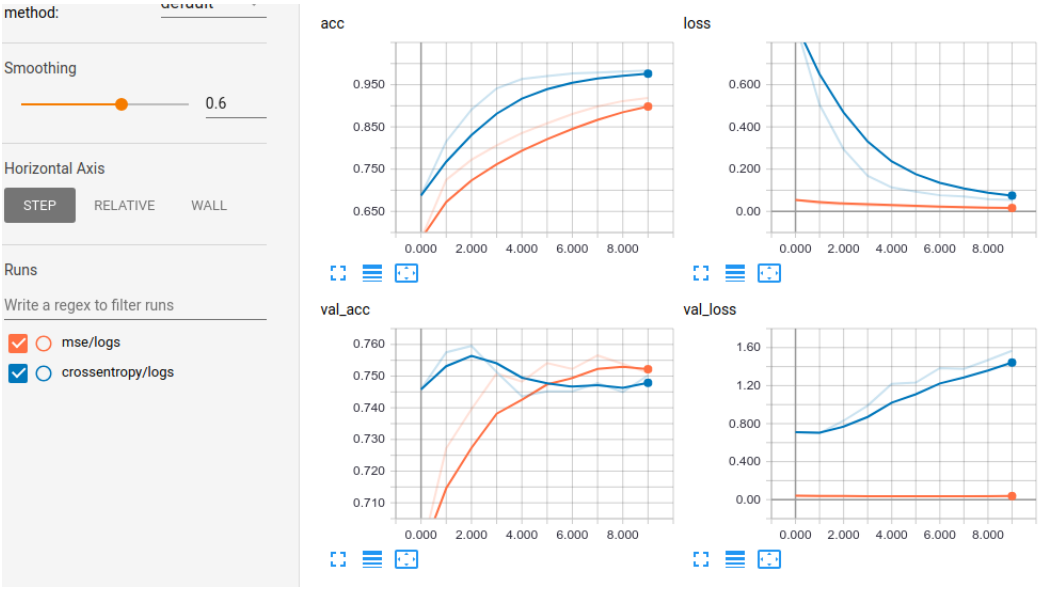

Figure 9: Results of training neural network classifier with different loss functions.

3.5 Regularizations

Since in all previous experiments it can be clearly seen that models overfit the data rally quickly, it is important to test different regularization terms. I have tested: **l1**, **l2** and **dropout**. While when using l1 and l2 improvement wasn't that significant, applying dropout (with dropout rate 0.5) helped.

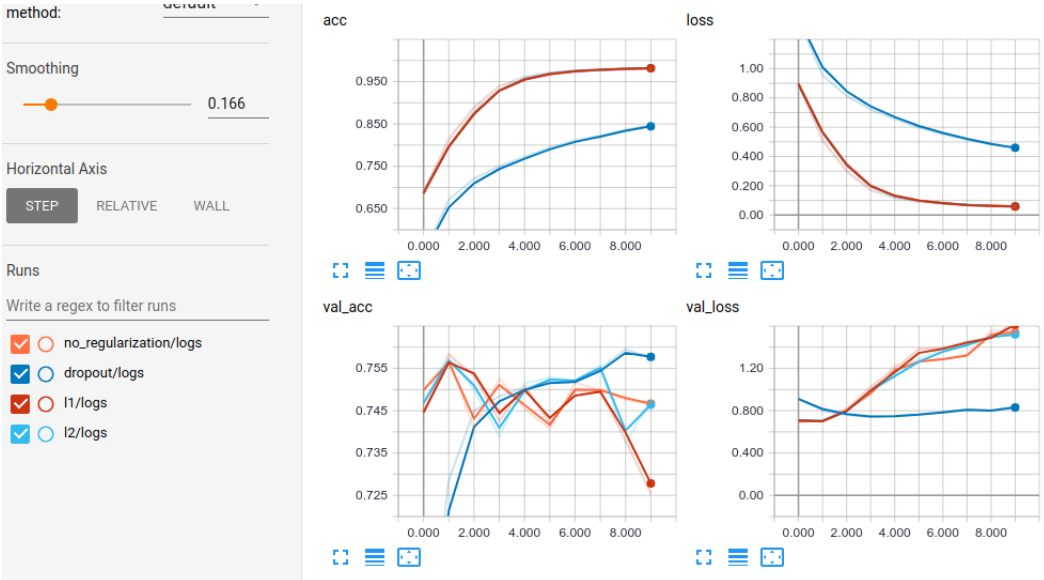

Figure 10: Training neural network with different regularizations.

4 Results

For each of the previous experiments, as well as for the final model I have collected required statistics: **accuracy**, **precision**, **recall** and **f1 score**. Precision, recall and f1 are computed both per class and weighted average (using scikit_learn function *precision_recall_fscore_support()*). Additionally I computed the confusion matrix. All the metrics are saved in json files.

4.1 Best Model

During the previous experiments I have noticed two things. First that having larger input significantly improves quality of classification. Second almost every model overfit really quickly. As my final model I have decided to build very simple neural network, only two hidden layers with 64 and 32 neurons. One dropout layer in between with rate set to 0.25. Having small network helps in both cases, it is less likely to overfit the data, and I can feed it a large input while still having reasonable training time. I used very large input of size 10000. Input BOWs were in binary mode with POS tags. Cross entropy loss function was weighted to deal with the unbalanced classes. I used early stopping and model was trained after just 4 epochs.

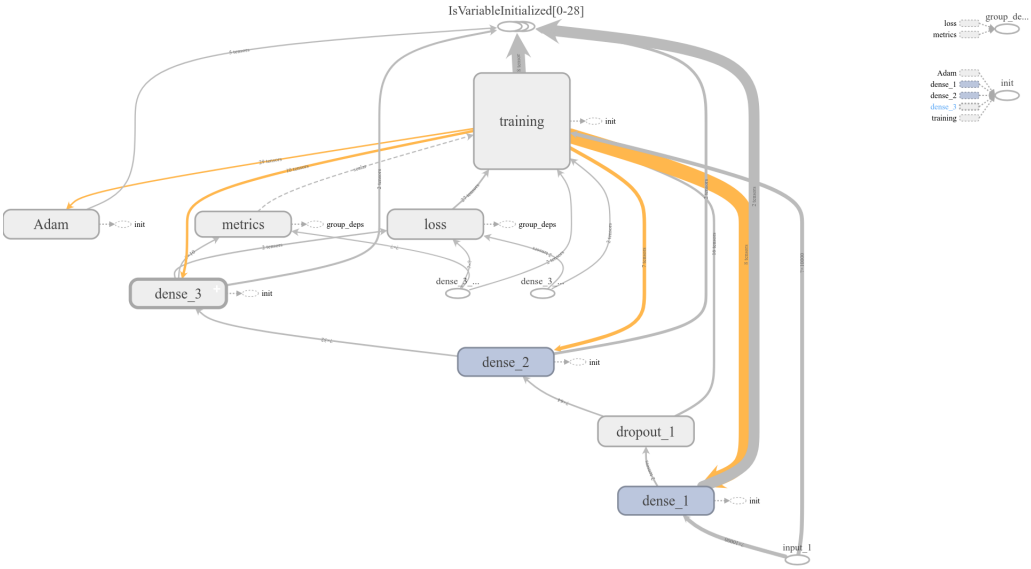

Figure 11: Neural network model.

4.2 Evaluation

I have trained model three times and computed the mean and standard deviation of all metrics. Results are available as json files in *experiments/best/* for both validation and test set. Here I present resoults as tables. Model achieved average accuracy on validation set of **0.7765** with std **0.0051**.

	classes	precision	recall	f1
0	4 Traders	0.6918	0.6223	0.6531
1	App.ViralNewsChart.com	0.9193	0.9091	0.9141
2	Individual.com	0.7843	0.7567	0.7701
3	Latest Nigerian News.com	0.7675	0.6709	0.7158
4	Mail Online UK	0.7871	0.7885	0.7874
5	MyInforms	0.8135	0.8918	0.8507
6	NewsR.in	0.9798	0.9073	0.9421
7	Reuters	0.5948	0.6369	0.6147
8	Town Hall	0.5726	0.4116	0.4787
9	Yahoo! Finance	0.6265	0.6387	0.6305

Figure 12: Average values of precision, recall and fscore for each class on dev.

	classes	precision	recall	f1
0	4 Traders	0.0327	0.0419	0.0078
1	App.ViralNewsChart.com	0.0056	0.0151	0.0082
2	Individual.com	0.0122	0.0135	0.0071
3	Latest Nigerian News.com	0.0019	0.0201	0.0121
4	Mail Online UK	0.0362	0.0093	0.0185
5	MyInforms	0.0141	0.0115	0.0087
6	NewsR.in	0.0036	0.0069	0.0021
7	Reuters	0.0174	0.0175	0.0044
8	Town Hall	0.005	0.0223	0.0168
9	Yahoo! Finance	0.0244	0.0476	0.0133

Figure 13: Standard diviation of precision, recall and fscore for each class on dev.

On the test set, model achieved average accuracy **0.7585** with std **0.0009**, which is slightly worse but that is understandable since hyperparameters were tuned for validation set. Nevertheless model is stable, since standard deviation is small.

	classes	precision	recall	f1
0	4 Traders	0.6921	0.6046	0.6435
1	App.ViralNewsChart.com	0.9157	0.9107	0.9132
2	Individual.com	0.7659	0.7662	0.7659
3	Latest Nigerian News.com	0.7727	0.6225	0.6895
4	Mail Online UK	0.7746	0.7686	0.7713
5	MyInforms	0.7986	0.884	0.8391
6	NewsR.in	0.9776	0.8922	0.9329
7	Reuters	0.53	0.5874	0.556
8	Town Hall	0.5467	0.4202	0.4739
9	Yahoo! Finance	0.6031	0.5757	0.5855

Figure 14: Average values of precision, recall and fscore for each class on test.

	classes	precision	recall	f1
0	4 Traders	0.0421	0.0301	0.0076
1	App.ViralNewsChart.com	0.0071	0.0128	0.0088
2	Individual.com	0.0155	0.0079	0.0043
3	Latest Nigerian News.com	0.0042	0.0083	0.0066
4	Mail Online UK	0.0213	0.0104	0.0056
5	MyInforms	0.006	0.0097	0.0041
6	NewsR.in	0.0087	0.0042	0.0025
7	Reuters	0.0146	0.0402	0.0107
8	Town Hall	0.0323	0.0189	0.0007
9	Yahoo! Finance	0.0324	0.0584	0.0178

Figure 15: Standard diviation of precision, recall and fscore for each class on test.

Looking at metrics for individual classes, it can be observed that model has high success rate of properly classifying some of them, when the others mostly the ones that had less samples in the dataset are difficult. Weighting loss function helped but didn't solve the problem. Performance could be probably improved on those classes, by increasing number of samples in the dataset.

5 Conclusion

I have implemented flexible framework, that allowed me to test different models for given classification problem. I have tested different sets of features and hyperparameters such as input sizes, number of layers, activations, loss functions and optimizers. Finally I have built model that achieves stable 0.75 accuracy on test set.