rijksuniversiteit groningen

Hyperparameter tuning for Noun-Noun compound classification and Word Embeddings

Ruben Schut s2717166
Sharif Hamed s2562677

October 12, 2020

Abstract

In this research paper we investigate the optimal hyperparameters using an adaption on the multi-layered perceptron. Different settings of the hyperparameters are investigated using a cross-validation scheme. We perform several experiments with different configurations and we elaborate on our choice for these settings. The results show that the Adam optimiser with a learn rate of 0.0005, a dropout rate of 0.4 with a batch size of 8 is optimal for our architecture.

1 Introduction

During this week's assignment, we will be experimenting with Neural Network Architectures (NNA). Neural Networks are powerful models that have the property of being able to approximate any non-linear function, given enough nodes in the hidden layer [0]. The NNAs will be used to classify noun-noun compounds. These compounds are built up from pre-trained word embeddings, as we have seen in previous weeks.

The goal of this assignment is to build a neural network that is able to identify the semantic relation between two nouns. An example of such a relation other than the ones given in the assignment are *NARCOTIC, TRAFFICKER*, *HEALTH, INSPECTOR* and *HOUR, MINUTE*. The first example is a trafficker of narcotics and hence is given the *objective* class. In the second example the subject is an inspector of health and is thus given the class *objective*. The last example are units of measurement and is thus given the class *measure*. Some of the input is open to debate, as stated in the assignment.

The remainder of this research paper is structured as follows. The data we use for our experiments is explained in section 2 as well as the data used to train our word embeddings. In section 3 the methods with which we perform our experiments are discussed. Following up on this section, our experimental setup is discussed in section 4. The results of these experiments are given in section 5. Finally, we discuss the main findings of our results and elaborate on the lessons learned in section 6.

2 Data

The data for the classifier consists of received data in the form of word embeddings and text word data which are described in section 2.2. In this paper also the creation of the word embeddings are researched and the data for this is explaned in the following section

2.1 Word embedding

For the classifier we used pre-trained word embeddings with dimensional length of 300. In order to create/train the word embedings the GoogleNews data set was used. The size of this corpus is 100 billion words creating from which 3 million unique words. The data set is not made public but it is known to use only news articles [0].

Next to the pre-trained word embeddings, self made embeddings where created/trained by using a different corpus. For training word embeddings we used the corpus of the paper [0]. This corpus consists of four genres: blogs, news articles, academic papers and digitised books. The size is significantly smaller then that of GoogleNews, namely about 136.8 million words from which 54 thousand unique words. The trained word embeddings/vectors have a length of 300.

2.2 Classifier data

In this assignment, training data consists of NOUN-NOUN compounds and word embeddings. The word embeddings are vectors of length 300 and are trained by word2vec using the GoogleNews data set. The elements of the vectors are floats to represent a weighted value. The maximum weight in the pre-trained word embeddings is 2.6668 and the minimum weight is -3.0241.

The data is constructed in the `load_data` method, situated in the class `Data` in the `DataService` module. The file training_data.tsv contains rows of three words. The first two words contain a noun and the third word is the corresponding class. For each row, the embedding of the two nouns are concatenated and appended to a list. This list then serves as input to the classifier and is of length 600 for each pattern (input data and corresponding class). There are 37 distinct classes. A few examples of such classes other than the ones given in the assignment are *PURPOSE, OTHER* and *EMPLOYER*. The training data is split into a training, development of sizes 12261, 3066, respectively. The test data is taken from the file test_data_clean.tsv and is of length 3831. The classes of the test set are unknown to the user and will be evaluated by the staff.

3 Method/Approach

In order to train our different architectures, we train our classifiers using K-fold cross validation on the training data. The development set will be used to measure the accuracy of a particular architecture. In section 4 several architectures are described and the design choices are motivated. We follow the conference paper by Verhoeven et al. (2012) [0] and calculate the precision-, recall- and F-score for each validation on the development set in order to quantify performance of our different architectures.

4 Experimental Setup

In the following subsections several different architectures for feed forward neural networks are outlined and motivated. Our experiments have the same basic architecture, where we use k-fold cross validation to split our data set into six parts of roughly the same length. We use one fold to validate our data on and another part to use in our early stopping criterion, which we use in all experiments. This way, we make sure to use all folds in training data, but simultaneously do not over fit on the early stopping criterion. The loss function we use is categorical_cross_entropy.

Early stopping is a method to generalise properly. The way it works is that you provide a metric to the model to evaluate after every epoch. We use an accuracy metric on a set held out of the training data called early_stopping_set. If this metric decreases, the model will stop fitting on the training data. However, it may happen that a local minimum is found and that the accuracy on the early_stopping_set decreases. To mitigate the chance of finding a local optimum, we implement a patience of 3. This means that the model will continue to fit on the training data until the evaluation metric decreases three times in a row or the pre-specified number of epochs is reached. The optimal number of epochs depends on the optimiser and its learn rate and therefore varies per experiment.

4.1 Experiment I - Dropout

In the first experiment, we iterate over a dropout rate of 0.1 to 1 with a step of 0.1. This way, we analyse the optimal dropout rate for our architecture. The architecture we use contains two hidden dropout layers with ReLu activation and a dense layer with softmax activation for the hidden-to-output layer. The number of nodes in the hidden layers are equal to the number of nodes in the previous layer divided by the dropout rate, according to Jason Brownlee [0].

4.2 Experiment II - Optimiser

In this experiment, we try out different optimisers for the network outlined in experiment 4.1 and optimise the learn rate. The optimisers we try are Stochastic Gradient Descent (SGD) with a momentum of 0.9, Adamax optimiser with a predefined learn_rate and Adam optimiser with a predefined learn_rate.

First we experiment with a lot of different learn_rates and have found the optimum for Adam is somewhere lower between 0.00005 and 0.0005. Then, we iterate from 0.0001 to 0.0025 with a step size of 0.0005 to find the optimal learn_rates. Because the learn_rate is so small, we increase the default number of epochs to 100 in order to minimise the probability of not finding an optimum within the pre-defined epochs. We calculate accuracy-, precision-, recall- and f1-scores for each learnrate based on the early_stopping_set. Using the adam optimiser, the optimal learnrate for all scores is 0.0005.

Second, we experiment a little bit with SGD with momentum and find that the optimum for this configuration is somewhere between 0.01 and 0.1. Therefore, we iterate from 0.01 to 0.12 with a step size of 0.02. We set the default number of epochs a little lower, because this learn rate is quite high and this optimiser converges to an optimum faster. We calculate accuracy-, precision-, recall-

and f1-scores for each learnrate based on the early_stopping_set. Using SGD with momentum, the optimal learnrate for all scores is 0.03.

Third, we experiment with the adamax optimiser and find that the optimum is somewhere between 0.0005 and 0.0025. Then, we iterate from 0.0005 to 0.0025 with a step size of 0.0005 and find that the optimal learn_rate for the adamax optimiser is 0.0015 based on precision- recall and f1-score. The best learn_rate in terms of accuracy was 0.002. However, we think that the other scores are better for generalisation and choose that.

4.3 Experiment III - Batch size

With our network parameters, we want to know what the best batch size is 8. Preliminary research has shown that a small batch size is favourable. In the literature, its known thats its best to set the batch_size as a polynomial of 2 for best performance run times. We have performed an experiment where we iterated over [8, 16, 24, 32] as the batch size.

5 Results

5.1 Experiment results

5.1.1 Experiment I - Dropout Rate

We calculate accuracy-, precision-, recall- and f1-scores for each dropout rate based on the early_stopping_set. The results are shown in table 1. As can be seen from the table, the optimal dropout rate for all scores is 0.4.

Table 1: Dropout Rate Scores

dropout rate	accuracy	precision	recall	f1
0.4	0.7260	0.6461	0.6780	0.6483
0.5	0.7161	0.6389	0.6630	0.6388
0.6	0.6947	0.6247	0.6370	0.6146
0.7	0.6475	0.5805	0.5922	0.5637
0.6	0.5737	0.5036	0.5552	0.4918

5.1.2 Experiment II - Optimiser

To determine the best optimiser, we calculate the accuracy-, precision-, recall- and f1-scores for the best learn_rate for all of them for this network. The networks are trained on the training data and evaluated on the development data. The results are presented in table 2. As can be observed from the table, the optimisers all perform pretty well, but the Adam optimiser performs slightly better than the others. Therefore, the Adam optimiser with a learn rate of 0.0005 is our choice for the best model.

Table 2: Optimiser Scores

optimiser	accuracy	precision	recall	f1
Adamax(lr=0.0015)	0.7537	0.6886	0.7386	0.70114
Adam(lr=0.0005)	0.7586	0.6979	0.7384	0.7087
SGD(lr=0.03, momentum=0.9)	0.7423	0.6933	0.7304	0.7065

5.1.3 Experiment III - Batch size

Table 3 shows the results of the batch_size.

Table 3: Batchsize Scores

batch size	accuracy	precision	recall	f1
8	0.7125	0.6285	0.6657	0.6314
16	0.7038	0.6147	0.6615	0.0.6215
24	0.7080	0.6249	0.6674	0.6300
32	0.7067	0.6191	0.6697	0.6263

5.2 Word Embeddings and Generalization

It is already shown that data sets other than the Google News data set give higher performance. Particularly when the data is less general and more task specific then the Google News data set, the performance increases [0]. The google embeddings can be seen as universal word vectors/embeddings [0]. This is due to the fact that the data set is so large. However, since the sources of the corpus are all news articles, we would not say that the word embeddings of GoogleNews are fully generalized. The corpus of [0] has more diversity in that is also uses blogs, academic papers and books.

Table 4: Interesting results similarities

word	Self made embeddings	GoogleNews embeddings
Seventeen	Seventy, Twelve, Eighteen	Roughly, Altogether, Some
Pole	Star, Downs, outh	IceCube_Neutrino_Association Ernest_Shackleton Manaslu
Bear	bears, grizzly, moose	Anna_Kozyreva Malayan_sun Cabinet_Yaak_ecosystem
orange	red, pink, tangerine	participant_LOGIN creamsicle fleshed_sweet_potato
threshold	criterion, limit, maximum	Tax_Credit_IETC Cleft_palate_Down_syndrome above

6 Discussion/ Conclusion

The network architecture that we chose is a feed-forward network with two hidden layers. The input-to-hidden layer is a dropout layer with an optimal dropout rate of 0.4. The input-dimension is 600 nodes, whereas the first hidden layer has 1500 nodes (600 / 0.4). The first hidden layer has a ReLu activation function. Further, the hidden-to-hidden layer is a Dropout layer with optimal dropout rate of 0.4. The second hidden layer also has 1500 nodes. The last layer is a dense layer with softmax output. The number of output nodes are the same as the number of classes (37).

The parameters we have tested are based on a k-fold validation scheme and are therefore properly regularized and are the best for a generalisation based on these parameters. Furthermore, we could have experimented a little more with different architectures such as additional hidden layers. Additionally, we could have experimented with improved feature selection to classify the nouns.

References

Jason Brownlee. A gentle introduction to dropout for regularizing deep neural netowkrs, 2018.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

Shibamouli Lahiri. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 96–105, Gothenburg, Sweden, April 2014. Association for Computational Linguistics.

Quanzhi Li, Sameena Shah, Xiaomo Liu, and Armineh Nourbakhsh. Data sets: Word embeddings learned from tweets and general data. *arXiv preprint arXiv:1708.03994*, 2017.

Ben Verhoeven, Walter Daelemans, and Gerhard Van Huyssteen. Classification of noun-noun compound semantics in dutch and afrikaans. pages 121–125, 09 2012.

Xiaocong Wei, Hongfei Lin, Liang Yang, and Yuhai Yu. A convolution-lstm-based deep neural network for cross-domain mooc forum post classification. *Information*, 8:92, 07 2017.

Xiaocong Wei, Hongfei Lin, Yuhai Yu, and Liang Yang. Low-resource cross-domain product review sentiment classification based on a cnn with an auxiliary large-scale corpus. *Algorithms*, 10:81, 07 2017.