LSTM for Classifying News Articles through Headlines and Short Descriptions

Andreas T. Damgaard, 201906422 Casper M. Haurum, 201906479

Deep Learning F21
Data Science, Aarhus University

December 17, 2021

1 Problem Statement

In this project, we are interested in classifying news articles to a specific genre[1] through their headlines and short descriptions. This is a Natural Language Processing (NLP)[2] problem, and we will therefore use a Recurrent Neural Network (RNN) to solve the problem. More precisely, we have decided to use a Long-Short Term Memory (LSTM) architecture, which will be described more in-depth.

2 Presentation of Data

To solve this categorization problem, we have used a dataset which contains information of 202,372 news articles published in The Huffington Post (HuffPost) from 2012 to 2018[3]. A data point contains the following information of an article:

- Category: a string representation of the genre of the article
- Headline: a string representation of the headline of the article
- Authors: a string representation of the names of the authors of the article
- Link: a string representation of the website link to the article
- Short Description: a string representation of a short description of the article
- Date: a string representation of the date that the article was published at the website

2.1 Exploratory Data Analysis

First we inspect the data that we are interested in. We create a distribution plot of the categories, and quickly realize we are dealing with an imbalanced dataset. This is something we need to deal with. First, we group some categories – either because they directly overlap such as "Culture & Arts" and "Arts & Culture", or because they share semantic space such as "Science" and "Tech". Hereafter, we create a weighted distribution of our labels based on the proposed formula in scikit-learn's compute class weight function[4], which we will implement in our optimizer through class weights.

[1] List of genres chosen by Huffington Post, see Appendix A for overview.
[2] https://www.ibm.com/cloud/learn/natural-language-processing
[3] https://www.kaggle.com/rmisra/news-category-dataset
[4] https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html

We inspect the length of the headlines and the short descriptions.[5] The lengths of the headlines appear approximately normally distributed. We decide to create a cutoff such that we only keep articles for which the length of the headline is between 1 and 120 characters. This enhances the normally distributed appearance, removes missing data, and ensures that the few outliers with very long headlines does impact our performance when we start padding. The short descriptions do not have this clean distribution. There is an extreme jump in occurrences from length 120 characters to 131 characters. There appears to be an error in the dataset. Upon further investigation, we find that when a short description of the article has not been available, the web scraping script has been programmed to use the first 120 characters or until the first space after 120 characters appears. Therefore, these data points do not represent what we are actually interested in, and hence have been removed from our dataset. We end up with around 124.000 data points.

2.2 Data Transformation

The input to our model will be a vector representation of a concatenation of the headline and the short description. First, we create a vocabulary of the words that exist in the headlines and the short descriptions such that each word can be represented by a number. Furthermore, we extend each vector representation by adding our number representation of "<pad>", which is 0, until the length of the vector is equal to the maximum length of vector representations. Afterwards, we convert the label, category, into a numerical value. Then our data is in the correct format for being inputted into our model. We randomly split the data into 80% training set, 10% validation set, and 10% test set. Afterwards, we create dataloaders, put the data transformation into the dataloader, and then we are ready to use the data in our model.

3 Description of Network Architecture

3.1 Recurrent Neural Network in General

Recurrent Neural Networks (RNNs) are networks taking sequential data as input.[6] This means that they are commonly used for tasks such as speech recognition and language translation. Unlike many other types of networks, RNNs incorporate information about prior inputs into its current output, thus not assuming independent inputs. This is the important memory ability. RNNs can be either uni- or bidirectional, the difference being whether or not they are able to use future events in determining its current output. We use a bidirectional variant.

[5] See Appendix B for visualizations.

[6] https://www.ibm.com/topics/recurrent-neural-networks

3.2 Long Short-Term Memory

The long short-term memory architecture (LSTM) introduced in 1997 by Hochreiter & Schmidhuber[7] is (together with the gated recurrent units (GRUs) architecture) one of the best performing RNN architectures. It was introduced to combat the issue of vanishing gradients, which can be encountered when training traditional RNNs. The LSTM have so-called cells in its hidden layers. These cells contain an input gate, an output gate, and a forget gate. The cells control which information is needed for a prediction.

3.3 Our Network Structure

For our network we start with an embedding, so that our network gets more information about the context that each word is used in. Then we use dropout before forwarding the data to our LSTM hidden layer. We are using a stacked, bidirectional LSTM with dropout between each layer. Afterwards, we calculate attention based on the last hidden state and the output from the LSTM layer. With attention, we hope that the network will be able to learn which part of the output of the LSTM layer that it should focus on. After using attention, we apply log softmax such that the output of our model is compliant with our loss function, Cross Entropy Loss. This loss may be weighted through the previously mentioned class weights. Defining the size of a network is a challenge in itself and therefore, we have decided to use hyperoptimization.

4 Hyperoptimization

To choose the hyperparameters of our model, we have decided to implement a Bayesian optimization method proposed by Bergstra et al[8]. We use the Hyperopt package[9] in which we define distributions for our hyperparameters - for example we have defined the distribution for the amount of stacked LSTM layers to be a random integer between 1 and 8. Meaning that we still need to define boundaries for our hyperparameters. Hyperopt then uses Tree-structured Parzen Estimator (TPE)[10] to replace our configuration with non-parametric distributions. Essentially, Hyperopt builds a surrogate function, which is defined as $P(\text{objective}|\text{hyperparameters})$, but uses Bayes' rule, and instead estimates $\frac{P(\text{hyperparameters}|\text{objective})\cdot P(\text{objective})}{P(\text{hyperparameters})}$. TPE calculates this such that there is a balance between exploration and exploitation.

We run 30 trials with 30 different suggestions from TPE, and Hyperopt then returns the suggested set of hyperparameters. Here, 6 epochs are chosen after running multiple hyperoptimizations and then realizing that some of our models are already extremely influenced by overfitting after 2-3 epochs. Therefore, after 30 trials of hyperoptimization and 6 epochs, we expect to choose the set of hyperparameters that minimize the loss while also being the least influenced by overfitting.

[7] https://www.bioinf.jku.at/publications/older/2604.pdf
[8] http://proceedings.mlr.press/v28/bergstra13.pdf
[9] https://github.com/hyperopt/hyperopt
[10] https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

There are some hyperparameters that we have chosen not to include in our hyperoptimization. We have for example chosen Adam as our optimizer, and have decided not to include weight decay. This is done after reading an article from 2003 by Bengio et al. in which they argue that this form of regularization can "limit the model to single point attractor at the origin, where any information inserted in the model dies out exponentially fast".[11]
Furthermore, we could have run hyperoptimization in which we also included switching between Stochastic Gradient Descent and Adam as our optimizer, or even switched between LSTM and GRU. We have also chosen to run our model with a batch size of 300. This is relatively arbitrarily chosen, but the main factors behind the decision are memory and training speed. As we have just described, we have decided to put some limitation on our hyperoptimization – the more hyperparameters we wish to include in the hyperoptimization the more trials we will have to run.
After running the hyperoptimization we use the resulting values as our hyperparameters for training the network.[12]

5 Performance

Both hyperparameter optimization and training was carried out for several different setups.[13] The most successful model achieved a test accuracy of 65.48%.

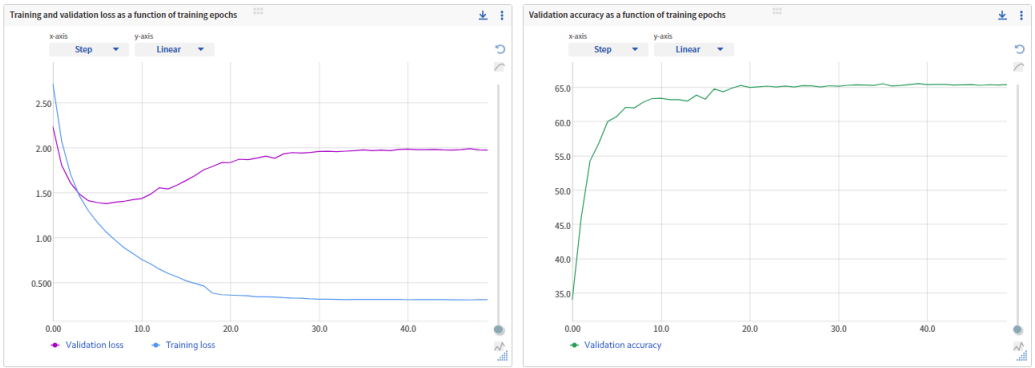

Fig. 1 The left plot contains loss graphs of training data (blue) and validation data (purple), while the right plot contains an accuracy graph (green), all as a function of number of training epochs.

In Fig. 1 we see that both the validation and training loss drops nicely during the first couple of epochs. Then, around the 6th epoch, the validation loss starts to increase. Coincidentally, 6 is exactly the number of epochs that our hyperparameter optimization was based upon. What initially seemed perplexing to us, was that the accuracy seemed unaffected by the increase in validation loss. Our interpretation of this phenomenon, however, is that as training continues, the model becomes increasingly uncertain in its classifications. This does not change which category the model predicts,

[11] http://proceedings.mlr.press/v28/pascanu13.pdf
[12] See Appendix C for hyperparameter overview.
[13] See Appendix D for model setup overview.

it merely squeezes the probabilities monotonically closer to each other. This would allow both an increase in validation loss and a plateau in accuracy.

5.1 Discussion of Performance

It is difficult to compare our results with the other performances that have been published on Kaggle. It seems that not a lot of people have realized that there is a mistake in the scraping of short descriptions. The best performances, we have found on Kaggle, lay around 50-65% test accuracy, which is very similar to our results. Some people were able to achieve 92% accuracy, but only after minimizing the number of labels to 6. In general, we believe that reaching a higher accuracy than 65% is extremely difficult while maintaining the original categories. This is partly because the categorization chosen by Huffington Post is subjective, and sometimes even flawed – for example the headline "I'm tired of being taxed for being a woman" is categorized by HuffPost as "Politics" while it could also be categorized as "Women" in our opinion. With more categories to choose from, it is only natural that each article could correctly fit into multiple categories. Consequently, the best model in the world would not be able to reach 100%, because two authors could place the same headline in two different categories.

Another thought; we are interested in classifying news articles in general, but are only using articles published at HuffPost. This means that our domain of interest is news headlines and short descriptions in general, but our marginal distribution of our domain is headlines and short descriptions published at the website of HuffPost. This could be an issue with regards to the generalizability of our model when using it for other news websites. Perhaps the language used at HuffPost is much different than that used at other websites.

Furthermore, we are only dealing with the classes/genres which are used at HuffPost. There might exist other genres, to which an article belongs, which our model would not be able to classify - this could for example be an article about History, which is not a label in our dataset.

The fact that the genre of the articles were originally classified by the journalists, or someone else at HuffPost, could also be another potential issue with the generalizability of the model. The understanding- and the width of a genre at HuffPost could be much different than that of other news websites.

6 Conclusion

Our best model was able to correctly classify 65.48% percent of the 26 categories in the test set through the corresponding headlines and short descriptions. This is on par with other performances at Kaggle with this dataset. Whether the relatively low accuracy is due to the quality of the data or the quality of the model is up for debate. Generally, when the model incorrectly classifies the category of an article wrong, it makes an appropriate and sensible classification. Through this report, we have learned a lot about applying LSTMs and the importance of hyperoptimization for training neural networks.

7 Appendix

7.1 Appendix A

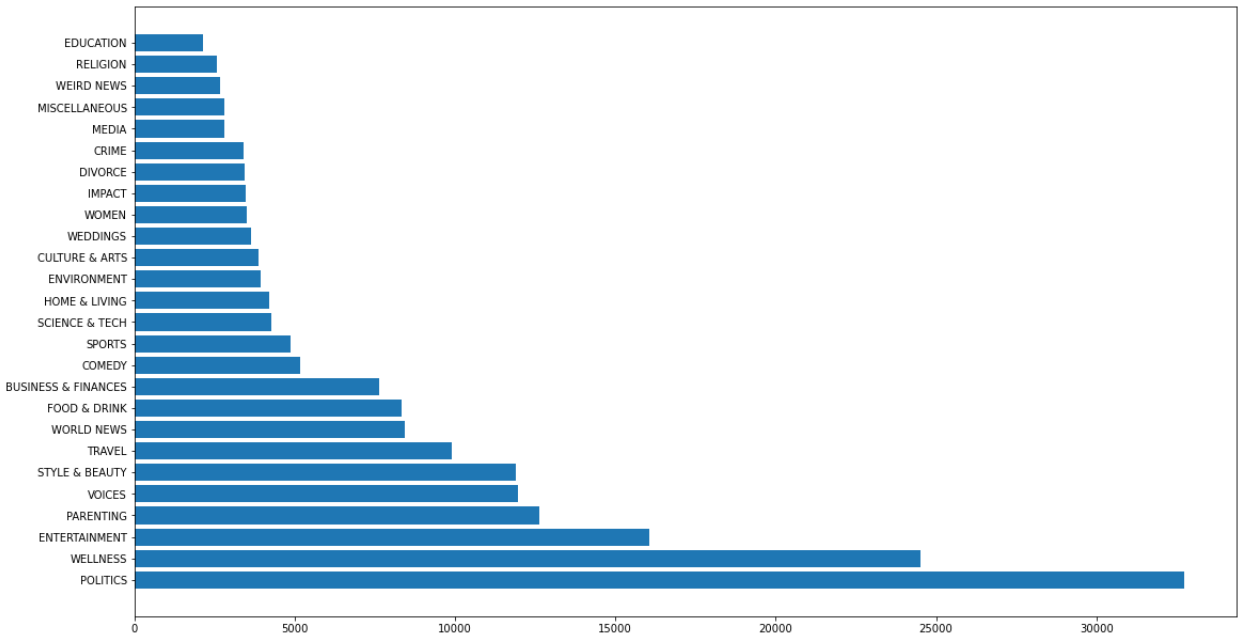

Fig. 2 The distribution of labels after they have been grouped. Using class weights, we try to further make up for the fact that there is a large difference in category sizes.

7.2 Appendix B

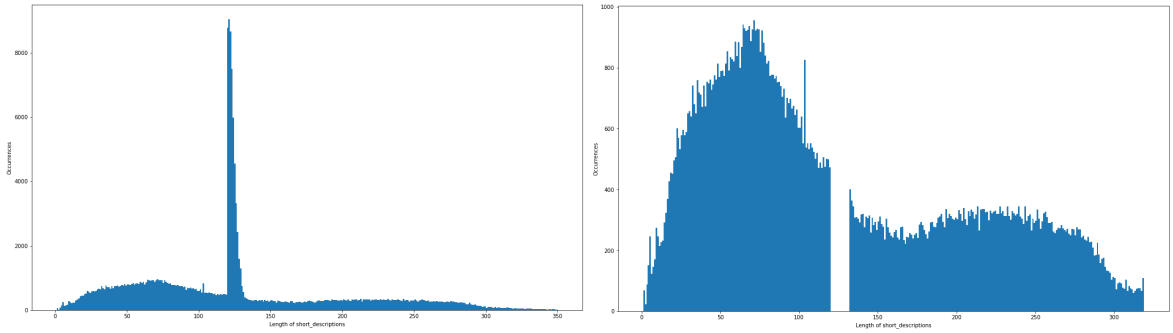

Fig. 3 The plot to the left shows occurrences of the short_description character length before data removal. The plot to the right is the same, but with the faulty descriptions removed and the right tail cutoff.

7.3 Appendix C

Hyperparameter	Optimized by HyperOpt	Value
Batch size	No	300
Use class weights?	Yes	Yes
Dropout	Yes	0.4888
Dropout (LSTM)	Yes	0.0450
Embedding size	Yes	106
Learning rate	Yes	0.001
Number of hidden layers	Yes	6
Size of hidden layer	Yes	96
Optimizer name	No	Adam
Scheduler name	No	ReduceLROnPlateau

Table 1 Hyperparameters used for training the model that achieved 65.48% accuracy.

7.4 Appendix D

	Datapoints	Model description	Test accuracy
Model 1	$n = 200,572$	Without attention	65.48%
Model 2	$n = 200,572$	With attention	64.40%
Model 3	$n = 124,410$	With attention, faulty descriptions removed	59.81%
Model 4	$n = 200,572$	Without attention, arbitrarily chosen hyperparameters	55.16%

Table 2 Overview of different model setups and their test accuracy.