A Comparison of Neural Document Classification Models

Matthias Nitsche, Stephan Halbritter
{matthias.nitsche, stephan.halbritter}@haw-hamburg.de
Hamburg University of Applied Sciences, Department of Computer Science
Berliner Tor 7, 20099 Hamburg, Germany

Abstract

In this paper we explore a wide range of different neural network architectures for text classification. We use a dataset from the news agency Deutsche Presse-Agentur (dpa) in German language. The architectures range from n-grams, convolutional neural networks (CNN) to long short-term memory networks (LSTM), including standardized baselines like FastText by Bojanowski et al. (2016), the first CNN for text classification by Kim (2014) up to state of the art models like *ULMFiT* by Howard and Ruder (2018). Since this is a real world dataset we explore the different problems that accompany them. Preprocessing despite using complex universal approximators is still very relevant but not as tedious as in bag of words methods. New generations of text models are very intensive in their computational requirements. Training takes days, while practical approaches like FastText achieve great performances without the need for these requirements.

Keywords – Neural Networks, NLP, Text Classification, FastText, CNN, LSTM, ULMFiT, Hierarchical Attention Network

1 Introduction

In this work we will explore different architectures of neural networks for text classification. Commonly text models are based on recurrent neural networks (RNNs) as they can keep the temporal state intact. Other approaches show that convolutions can be used to extract contextual text windows and keep some of the spatial regions as well. State of the art models however are build on LSTMs using either attention mechanism or finely tuned parameter tweaking. Text classification helps in making sense about vast amounts of documents by organizing them into buckets, even when the function to optimize is in a very high dimensional space. The need for ever more general and at the same time detailed classification is thusly still an important question. Real world data contains a lot of noise that many academic datasets do not inhibit, making classification a much harder task.

In our previous work (Halbritter and Nitsche, 2019), we established a baseline on the Deutsche Presse-Agentur (dpa) dataset. Using the Multinomial Naive Bayes algorithm, we achieved accuracy of up to 93%. The question is if an approach based on a neural network can beat this baseline score. The short answer is: yes, but with a few constraints. The hardware requirements on memory and GPU and CPU power rise immensely, the training time increases and there's a high prevalence to overfitting. In addition, by trading linear models for much more complicated non-linear approximators, neural networks are not easy to debug. Newer models like *ULMFiT* (Howard and Ruder, 2018) essentially train a generalized language model that can be trained once and fine tuned to solve different tasks later. This additional effort can be worth it, in particular when the classification task is not the only one to be solved.

We approach this question by comparing different classifiers. Five classifiers from the last four years were selected by two criteria. On the one hand the models should be battle proven and should be used

or likely to be used in production. On the other hand we chose models that provided a new approach, insight or other noteworthy improvement. The very fast and simple *FastText* by Bojanowski et al. (2016) provides a solid baseline. *CNN for Sentence Classification* by Kim (2014) is a ground-breaking model which introduced the concept of CNNs for text classification, on which others based their research, for example *Dynamic CNN* by Kalchbrenner et al. (2014). *Hierarchical Attention Networks* by Yang et al. (2016) presents a different model architecture using *Gated Recurrent Units (GRU)* and an attention mechanism to learn from the hierarchy embedded in texts. Finally we chose *ULMFiT* by Howard and Ruder (2018), the state of the art at the time, to pretrain a language model to finetune it on an out of domain task.

2 Data and Preprocessing

In this section we briefly describe the data we try to classify and discuss some of the preprocessing steps. The dpa groups articles into six resort as depicted in Table 1. Of these resorts, dpacat:rs is not meant for publication, but plays a huge role in classification performance. For an in depth look we refer to our previous work (Halbritter and Nitsche, 2019). We will briefly discuss the problems and ideas around the data.

Name	Field Name	Abs. Count	Percentage
Culture	dpacat:ku	11985	5.52
Politics	dpacat:pl	67410	31.05
Editorial Service	dpacat:rs	29067	13.39
Sports	dpacat:sp	37740	17.38
Mixed	dpacat:vm	40245	18.54
Economics	dpacat:wi	30648	14.12
Total		217095	100.00

Table 1: Overview of categories

The category data is heavily skewed towards politics, sports and the mixed category. The dpa corpus has several text fields, for example the article body, headline and description. For this work we worked entirely with the full article, that was available in plain text without html. Figure 2 shows the basic statistics of character and word counts per category and overall of the text field. The word count is based on a simple splitting on whitespace. Only texts with a length larger than zero are taken into account.

	Character Count				Word Count			
Field Name	Min	Max	Mean	Std	Min	Max	Mean	Std
dpacat:ku	54	12634	2410	1824	7	1893	303	214
dpacat:pl	17	24030	2058	2297	2	3565	254	237
dpacat:rs	47	36034	4419	3444	7	3058	436	313
dpacat:sp	56	40435	1931	1561	7	2625	276	221
dpacat:vm	53	20919	2050	1905	6	2984	256	226
dpacat:wi	53	18009	2396	1979	7	1912	289	199
All fields	17	40435	2418	2388	2	3565	291	246

Table 2: Character and word count statistics before stop word removal

The overall mean word count is around 290 per document. In the following we will discuss some of the choices we made concerning the preprocessing. The datasets were shuffled and split into different sets for training (63%), validation (27%) and testing (10%).

2.1 Category dpacat:rs

dpacat:rs (standing for *Redaktioneller Service* or *editorial service*) is a special category. It is not intended for publication but for editorial messages as well as overviews and previews of upcoming events and news. Therefore, it often consists in large parts of content from other categories. It accounts for around 13% of the total data items, while having the largest mean word count (see Table 2). Together, this make this category very prone to mis-categorization in both directions. Depending on the use case, this can be fixed by simply removing this category and its items from the dataset. This allowed us to train the models on a dataset with rather distinctive categories. We will see that classification results improve drastically when doing so. On the other hand, it's a very interesting case to keep this category and have a look how the different models handle the situation of a rather inconsistent dataset. In the end, we trained all models with and without dpacat:rs.

2.2 Relabeling

We coined the term *Relabeling* to deal with dpacat:rs. Fundamentally we found that dpacat:rs is not only inconsistent in its content, but sometimes duplicates entries of other categories. Typically, those are summaries that use the vocabulary and style from the respective category. The following is an extract whose content is clearly part of the politics category and can be found in dpacat:rs and dpacat:pl.

Auf www.dpa-news.de bieten wir Ihnen einen laufend aktualisierten Überblick über die dpa-Topthemen des Tages. Auch Ihre Fragen und Anregungen beantworten wir dort online. Die Planung für die nächsten 14 Tage finden Sie jederzeit auf dem aktuellen Stand auf www.dpa-agenda.de. Hier bekommen Sie Ihren persönlichen Zugang: service@dpa-news.de / +49 40 411332179.

Redaktion Politik

Ausland: Tel.: +49 30 2852-31302; E-Mail: politik-ausland@dpa.com

Inland: Tel.: +49 30 2852-31301; E-Mail: politik-deutschland@dpa.com

---

Hauptthemen Politik International:

Rom/Jerusalem/Bethlehem

- Weihnachtsfeierlichkeiten im Heiligen Land und im Vatikan + 1000 (MEZ) Jerusalem/Bethlehem: Traditionelle Weihnachtsprozession geistlicher Würdenträger von Jerusalem nach Bethlehem unter der Führung des lateinischen Patriarchen von Jerusalem, Pierbattista Pizzaballa. Ankunft an der Geburtskirche in Bethlehem erwartet für etwa 1230 (MEZ). + Vatikan: Papst Franziskus feiert Christmette im Petersdom + 2300 (MEZ) Bethlehem: Mitternachtsmesse des lateinischen Patriarchen von Jerusalem, Pierbattista Pizzaballa, in der St. Katharinenkirche neben der Geburtskirche Jesu. Palästinenserpräsident Mahmud Abbas wird ebenfalls dazu erwartet.

- Vorausmeldung «Papst Franziskus feiert Christmette - Weihnachtsprozession», am Samstag 1900

- Meldung «Tausende Christen feiern Weihnachten im Heiligen Land», aus dem Heiligen Land

- Zusammenfassung «Christen feiern Weihnachten im Heiligen Land», Namensbericht aus Jerusalem

- Meldung, aus Rom, bis 2230

- Zusammenfassung, aus Rom mit Material aus dem Heiligen Land, bis 2345

+++ Vatikan/Israel/Palästinensische Autonomiegebiete/Kirche/Papst/Weihnachten/ +++

We considered two different approaches to this problem, both using relabeling the dpacat:rs entries with duplicates to the corresponding duplicate's category. In the first approach, only the training

data would be relabeled. This would mean that in production, the model would likely confuse dpacat:rs entries with their duplicate's category, which could be the desired behaviour. The second approach assumes that the duplicates are the result of false data labeling and relabeling on the whole data set would fix this error. However, both approaches are not really acceptable for typical text classification tasks. They would introduce another level of preprocessing. We only peek into the effects of relabeling in the context of a single model in Section 3.5.5.

2.3 Syntactical preprocessing

Neural networks and especially embedding models are known to not need a lot of preprocessing. As a result, we kept the preprocessing to a minimum. Since we are not using any character based models and words from the test set are not known ahead of time, out-of-vocabulary words become a problem. The basic preprocessing consisted of the following six steps:

1. Lowercasing all words
2. Removing newlines
3. Removing special characters
4. Removing punctuation
5. Using cut off frequencies keeping only the top 30000 words
6. Stripping recurring headers and outros from the text

The idea of the preprocessing steps is to strip useless information, for example formatting like newlines and characters likely not found in the embeddings or the recurring first string *(dpa)* which does not carry significant information. We avoid resource and/or time-hungry preprocessing.

Additional preprocessing is customized to fit the requirements of the models. The baseline CNN uses raw word sequences. The HAN model uses hierarchal features that need sentence boundaries provided by a neural sentence tokenizer from SpaCy. ULMFiT language model is solely trained on an AWD-LSTM which uses a bptt window over the sequences, learning contextual information. More details about the relevant preprocessing steps that were necessary for each model are described in the corresponding model sections.

2.4 Embeddings

Facebook provides pre-trained 300-dimensional word vectors created with FastText. The original version is trained on *Wikipedia* using the same parameters as in Bojanowski et al. (2016). It is available for 294 languages. An updated version, trained on *Common Crawl* in addition to *Wikipedia* and with adapted parameters is available for 157 languages (Grave et al., 2018). These are the default embeddings we used in our experiments if pre-trained embeddings were suitable. In all our tests the accuracy dropped by roughly 1-2% when not using embeddings. This suggests that the models we used do a decent job at predicting the correct category even without them.

3 Models and Experiments

In this section we will present multiple (deep) neural network models for text classification and discuss their respective training results. Text classification is a good task to evaluate different models as well as the capacity of the underlying data set to be separated into relevant categories. In our previous work (Halbritter and Nitsche, 2019), we established a baseline using the Multinomial Naive Bayes algorithm. It achieved 83% accuracy on all six categories and 93.2% accuracy without category dpacat:rs. We will present five different text classification models exploring different architectures and approaches.

3.1 FastText

FastText, developed by Facebook, is an open-source library and command-line tool. The tool provides functionality for training word representations and training classifiers. It is based on the idea of representing words as continuous vectors as laid out in *Bag of Tricks for Efficient Text Classification*

(Joulin et al., 2016). The underlying model is fairly simple. Its power lies in the way it creates the word embeddings as presented by Bojanowski et al. (2017).

3.1.1 Embeddings

To calculate the word embeddings, it extends the *skip-gram model* known from *word2vec* (Mikolov et al., 2013). The main difference is that FastText sees words as the sum of their character n-grams. For example, the word *where* is represented by the character n-grams <*wh*, *whe*, *her*, *ere*, *re*> and the whole word <*where*> for n=3. Special characters < and > are used to mark boundaries of words. In practice, all n-grams where n is greater or equal to 3 and smaller or equal to 6 are extracted.

The final word vector consists of the sum of the vector representation of its n-grams, which means subword information is incorporated. This is in stark contrast to the traditional skip-gram model, where each word has its own distinctive vector. This new approach has clear advantages, as it can calculate embeddings even for *out-of-vocabulary (OOV)* words. This comes from the expectation that rare or unknown words consist of character n-grams that can be found in more common and known words. Word2vec and GloVe are both not capable to create embeddings for OOV. Another result of this is that rare words are much better contextualized.

The disadvantage of character n-grams lies mainly in the hardware requirements to generate them, especially the amount of RAM. They depend on hyperparameters like the vocabulary size and minimum and maximum length of the n-grams.

3.1.2 Model

The FastText model for text classification consists of a neural network with just a single hidden layer. The text is represented as a *Bag of Words (BoW)*. The embedding for each of the words is retrieved using a lookup table. This results in an input matrix with the shape $number\_of\_words \times embeddings\_size$. These embeddings are then averaged so that a single embeddings vector (e.g. of size 300) represents the whole input. Linear transformation is then applied before a softmax function calculates the class probabilities.

The training process tries to minimize the negative log-likelihood as presented in Equation 1. x_n represents the normalized BoW of the n-th of N words, A is the lookup matrix for the word embedding. B is the linear output transformation and f the softmax function to compute the probability distribution of the corresponding class, represented by its label y_n.

$$-\frac{1}{N}\sum_{n=1}^{N} y_n \log(f(BAx_n)), \tag{1}$$

3.1.3 Experiment

As this is the baseline model, we refrained from fine-tuning too much. This also concerns the preprocessing step, which therefore differs from the preprocessing in the other experiments as laid out in Section 3.1. The following preprocessing steps were applied to the dataset once with and once without items of the category dpacat:rs.

1. Lowercasing
2. Removal of German stop words
3. Removal of newlines and multiple spaces
4. Adding of space around some characters[1]

We used the FastText Python library for both model training and prediction. In the training process, we adopted the predefined default values for most of the arguments. Through experimentation, only two changes had a recognizable impact on the accuracy. The learning rate was changed from 0.1 to a rather large 1.0. Training ran for 9 epochs instead of the default number of 5. More iterations did not

[1] That concerned quotation marks, dots, commas, brackets, exclamation marks, question marks, semicolons and colons.

improve the outcome. No pretrained word vectors were used. Noteworthy default values are the word vector size of 100, the context window of 5 and the use of softmax in the loss function.

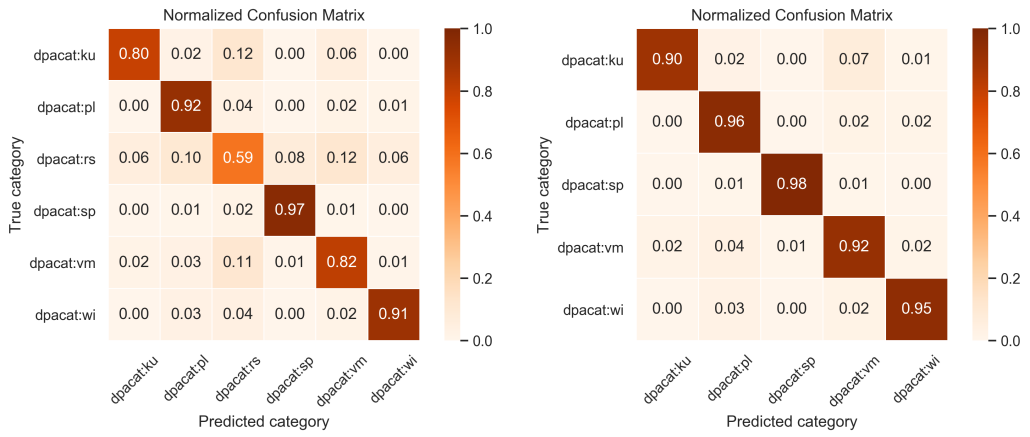
3.1.4 Results

Results are presented in Table 3 and Figure 1. A hardly surprising finding is that the results on the dataset without dpacat:rs are a lot better than with data of this category. Accuracy improved by ~11% to 0.95 while loss improved by ~45% to 0.18. Especially the categories Culture (dpacat:ku) and Mixed (dpacat:vm) are prone to get confused with dpacat:rs, but also to some extend with each other as can be seen in the confusion matrix in Figure 1a.

		With Editorial Service			Without Editorial Service		
Category	Field	Precision	Recall	F1 score	Precision	Recall	F1 score
Culture	dpacat:ku	0.79	0.80	0.79	0.92	0.90	0.91
Politics	dpacat:pl	0.91	0.92	0.92	0.96	0.96	0.96
Editorial Service	dpacat:rs	0.61	0.59	0.60	n/a	n/a	n/a
Sports	dpacat:sp	0.93	0.97	0.95	0.99	0.98	0.99
Mixed	dpacat:vm	0.84	0.82	0.83	0.92	0.92	0.92
Economics	dpacat:wi	0.90	0.91	0.90	0.95	0.95	0.95
Micro average		0.86	0.86	0.86	0.95	0.95	0.95
Macro average		0.83	0.83	0.83	0.95	0.94	0.94
Accuracy			0.8548			0.9519	
Loss			0.3302			0.1721	

Table 3: FastText results

The better results also express themselves in the micro and macro averages. Micro average first calculates the metrics for each category and then averages the result. Macro average on the other hand first sums all the values of the categories and then calculates the metric. An outlier result for one of the categories has a different impact on the result of the micro average than of the macro average. This results in a noticeable difference of these averages for the dataset including dpacat:rs. As can be expected when the results of the single categories lie closer together, the averages without dpacat:rs are close together.



(a) Trained with category *dpacat:rs*

(b) Trained without category *dpacat:rs*

Figure 1: Confusion matrices for *FastText* models

Overall, the results for both with and without Editorial Service are quite good and did exceed our expectations. We will see that the results are often on par in comparison to complicated deep learning

models. Training and prediction on the whole dataset was a matter of minutes on a common laptop.[2] In addition to providing an easy-to-use command-line program, this makes FastText a very good first baseline model.

3.2 Convolutional Neural Networks for Sentence Classification

Convolutional Neural Networks (CNN) are well-known for image classification. In this highly influential paper *Convolutional Neural Networks for Sentence Classification*, Kim (2014) laid out the idea to apply this architecture for text classification tasks.

3.2.1 Model

If a CNN is used for computer vision purposes, the image pixel values are used directly as the input matrix. In case of color images each color channel has its own matrix and the input is a tensor. While the models in computer vision can get quite deep and complicated, the model used by Kim (2014) is relatively simple. It consists of a single convolutional layer neural network. The general architecture of the model is shown in Figure 2.

For text classification, we first tokenize the text input. In general, this is done by using pretrained word embeddings. We used the FastText embeddings. This representation is then converted to a matrix. Each row consists of the vector representation of a single world. For example, if the embeddings are of dimension $d = 300$ and the sentence length is $s = 50$ words, this results in a matrix of dimensions $d \times s$, here 300×50. After that, we can continue in the same way we would if this matrix would represent an image and apply filters using convolution.

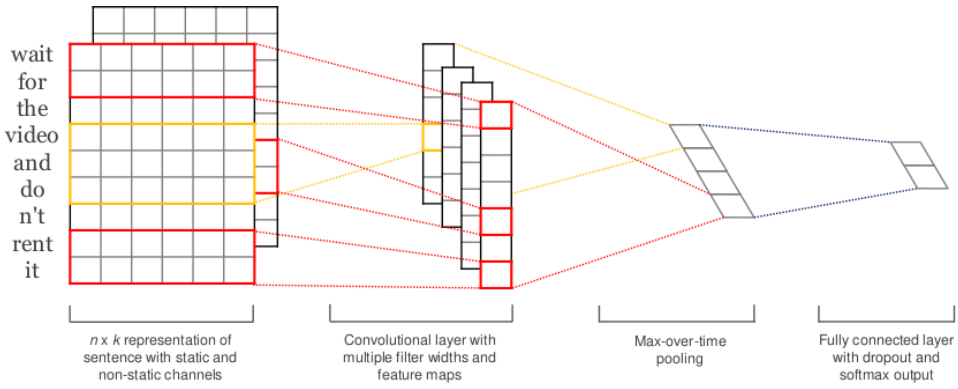


Figure 2: Model architecture example with two channel inputs (Kim, 2014)

In image matrices, each entry represents the discrete value of a single pixel. For text, the discrete value of a word is represented in a whole line. As we want to slide the filter kernel over neighboring words, it makes sense to freeze the kernel size's width to the size of a row d. Variations of the filter size are now restricted to different heights or number of rows representing neighboring words. This adds some sensitivity to the word order in the input sentences. Multiple filters of the same size are used to try to learn different aspects of the same neighborhood. The filters run for each possible window of words in a sentence. The results are saved in a feature map per filter.

For regularization, these feature maps are further passed through a pooling layer. *Max-over-time pooling* is applied to extract the highest value from each feature map, followed by a fully-connected layer with dropout and softmax to classify the input.

Kim (2014) presents slight variations of this model architecture in regard of the input matrices by differentiating between the embeddings as static and non-static. Static embeddings are fixed, while non-static embeddings are fine-tuned during training. Another variation combines those approaches and used both matrices, which then are called *channels*. This name is based in the wording of color channels used in computer vision. The same filters are applied to both channels, the static and the non-static one. Their results are summed up before being saved in the corresponding feature map.

[2] Lenovo Thinkpad X250 with i5-5300U processor and 8GB RAM

3.2.2 Experiment

Kim (2014) uses filter sizes of 3, 4 and 5 and 128 filters per size. As stated earlier, we used the FastText embeddings for German to initialize the word vectors. This results in a matrix width of 300. These embeddings are non-static and are fine-tuned during training. The maximum number of words per document was set to 1000, which in regard of Table 2 is a rather large value. We used zero padding for documents with fewer words. Batch size was 256, dropout was set to 0.6. Our training setup used the *EarlyStopping Callback*[3] of Keras with a patience value of 3. This means that training stopped after three epochs with no improvement. The model converged after surprisingly few training epochs. For the dataset including dpacat:rs, it took 5 epochs. The dataset without dpacat:rs needed 9 epochs to converge.

3.2.3 Results

The overall accuracy results are comparable to those of FastText but with a larger loss, especially when including dpacat:rs. 57% of dpacat:rs are not categorized correctly, with around 26% of the entries of dpacat:rs getting confused with dpacat:vm. As can be expected, the model performs better without dpacat:rs.

		With Editorial Service			Without Editorial Service		
Category	Field	Precision	Recall	F1 score	Precision	Recall	F1 score
Culture	dpacat:ku	0.81	0.82	0.82	0.94	0.86	0.90
Politics	dpacat:pl	0.84	0.98	0.90	0.95	0.93	0.94
Editorial Service	dpacat:rs	0.76	0.43	0.55	n/a	n/a	n/a
Sports	dpacat:sp	0.91	0.99	0.95	0.98	0.98	0.98
Mixed	dpacat:vm	0.79	0.86	0.82	0.91	0.90	0.90
Economics	dpacat:wi	0.97	0.81	0.88	0.87	0.96	0.92
Micro average		0.85	0.85	0.85	0.93	0.93	0.93
Macro average		0.85	0.81	0.82	0.93	0.93	0.93
Accuracy			0.8560			0.9380	
Loss			0.6134			0.2526	

Table 4: CNN results

There is an interesting shift in the results for dpacat:wi (Economics) and dpacat:pl (Politics) between the datasets with and without dpacat:rs, which does not occur in the other models. For dpacat:wi, recall improves drastically from 0.81 to 0.96, but precision falls down from 0.97 to 0.87. For dpacat:pl, precision improves from 0.84 to 0.95, but recall declines from 0.98 to 0.93.

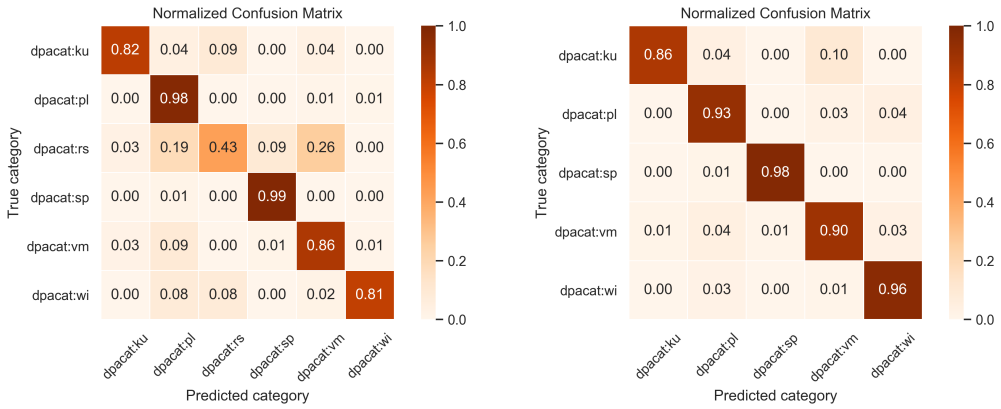
For dpacat:wi, as dpacat:rs was responsible for 8% of false categorization, the improvement of recall is not surprising. And it seems that the removal of dpacat:rs also makes it more likely for dpacat:pl and dpacat:vm to be misclassified as dpacat:wi, which causes the precision to go down.

For dpacat:pl, the improved precision is due to the large proportion of dpacat:rs which was misclassified as Politics in the first dataset. Similar to the former case, it seems that without dpacat:rs, the differentiation in regard to dpacat:vm and dpacat:pl got harder and as a result the recall results got worse.

3.3 Dynamic Convolutional Neural Network (DCNN)

This model presented by Kalchbrenner et al. is very similar to the one from Kim (see Section 3.2) and one of the examples how that model can be improved.

[3] `https://keras.io/callbacks/#earlystopping`



(a) Trained with category *dpacat:rs* (b) Trained without category *dpacat:rs*

Figure 3: Confusion matrices for *CNN 2014* models

3.3.1 Model

The main significant difference to Kim (2014) lies in the pooling layers that come after the convolutionals. *Dynamic k-max pooling* is a generalization of max-over-time pooling. Instead of a single value, a whole subsequence of the k largest values is returned. The value k can be dynamically calculated using different aspect of the model like sequence length and model depth. It is also possible to use a fixed value.

A consequence of dynamic k-max pooling is that the model has the ability to weight one of the smaller values the most, which makes it more robust for similar content in different categories. To a certain extend, the context can be included in the weights. A good example where this approach shows its advantage is the inclusion of the editorial service category in the dataset.

		With Editorial Service			Without Editorial Service		
Category	Field	Precision	Recall	F1 score	Precision	Recall	F1 score
Culture	dpacat:ku	0.80	0.67	0.73	0.88	0.90	0.89
Politics	dpacat:pl	0.93	0.87	0.90	0.94	0.93	0.94
Editorial Service	dpacat:rs	0.56	0.75	0.64	n/a	n/a	n/a
Sports	dpacat:sp	0.92	0.94	0.93	0.99	0.98	0.98
Mixed	dpacat:vm	0.83	0.76	0.79	0.90	0.91	0.90
Economics	dpacat:wi	0.91	0.87	0.89	0.93	0.94	0.93
Micro average		0.84	0.84	0.84	0.94	0.94	0.94
Macro average		0.82	0.81	0.81	0.93	0.93	0.93
Accuracy			0.8474			0.9351	
Loss			0.3155			0.2082	-

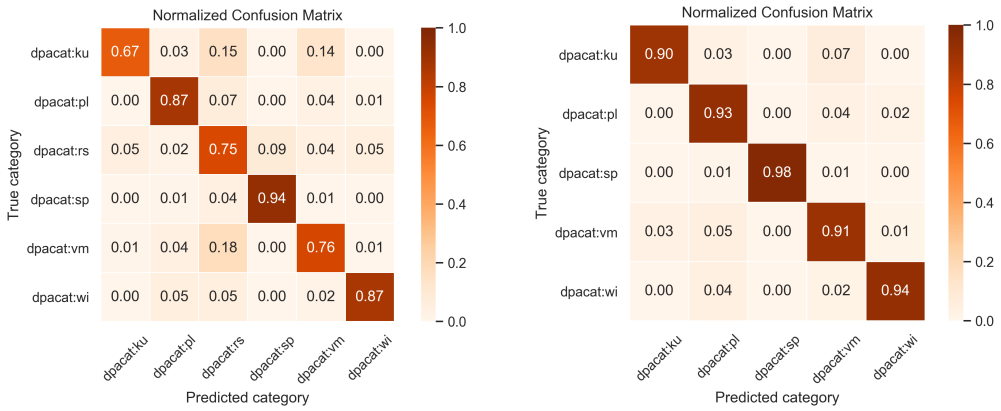
Table 5: DCNN results

3.3.2 Experiment

We applied the same hyperparameters as for the CNN as described in Section 3.2.2 and used a static value of k=5 for k-max pooling. The training process converged after five epochs including dpacat:rs and after just three epochs without this category.

3.3.3 Results

The impact of dynamic k-max pooling is clearly reflected in the confusion matrix (Figure 4a), especially in comparison with the result of Kim's CNN (Figure 3a). The overall results of precision, recall, F1 score and accuracy are very similar with a single significant exception of the training loss, 0.32 instead of 0.61. But the main difference lies in the precision and recall results per category.



(a) Trained with category *dpacat:rs*

(b) Trained without category *dpacat:rs*

Figure 4: Confusion matrices for *DCNN 2014* models

Here, the model improves drastically in regard to the categorization of *dpacat:rs*. While the CNN's recall for *dpacat:rs* is 0.43 and 26% were mislabeled as *dpacat:ku*, the DCNN increases the recall to 0.75 and only 4% were mislabeled as *dpacat:ku*. This comes at the cost of lower precision and recall values for the other categories. The DCNN is the model which achieves the worst result of all models for the precision of dpacat:rs.

3.4 Hierarchical Attentation Networks for Document Classification

Yang et al. (2016) introduce a new model architecture, called *Hierarchical Attention Network (HAN)*. It combines elements of a *Recurrent Neural Network (RNN)* with the *Attention* mechanism.

3.4.1 Model

There are two basic ideas behind this model:

1. Documents have a hierarchical structure and are formed by sentences, which themselves are formed by words.
2. Different words and therefore sentences carry different amounts of information relevant for the whole document

Even with stop words removed, not every word or sentence has the same relevance for the content category, while this relevance is dependent on the context, e.g. the neighboring words. As shown in Figure 5, the architecture can be separated in two parts.

The first part - depicted in the lower part of Figure 5 - works on a word level. A bidirectional GRU is used to learn about the information of words in the context of their sentence. Then, Attention weights these word information vectors in regard of how much they characterize the sentence. The intermediate result is a sentence vector of contextual, weighted word embeddings.

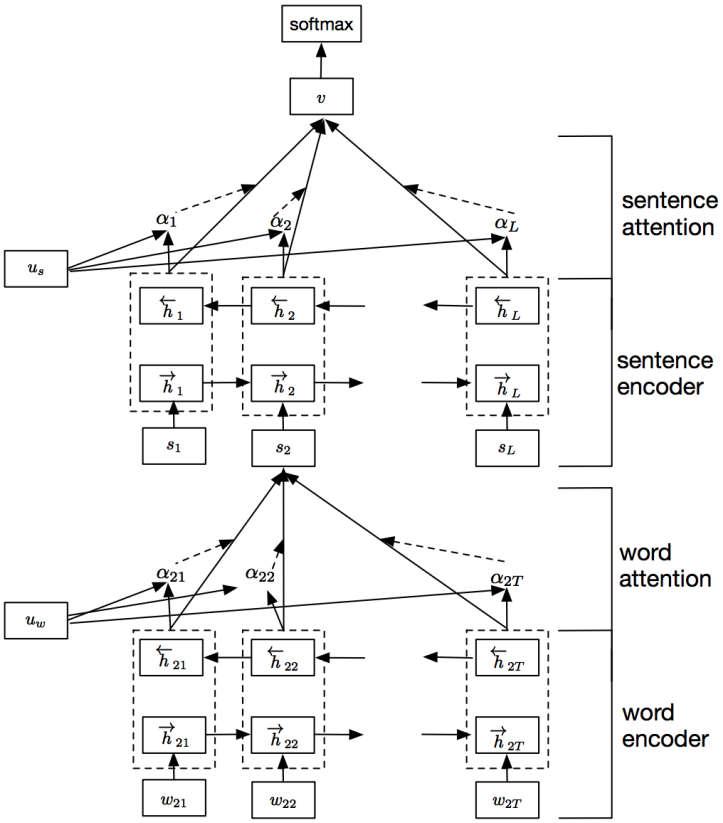


Figure 5: *HAN* basic idea as described in Yang et al. (2016)

Since attention is a neat concept to connect encoders with each other and it is coined frequently in the literature, let us quickly gloss over attention. The attention formula used on the words is as follows

$$u_{it} = tanh(W_w h_{it} + b_w) \tag{2}$$

$$\alpha_{it} = \frac{exp(u_{it}^T u_w)}{\sum_t exp(u_{it}^T u_w)} \tag{3}$$

$$s_i = \sum_t \alpha_{it} h_{it} \tag{4}$$

u_{it} is a per word linear layer multiplied with the last hidden state h_{it} of the before going word GRU. α_{it} is a per word similarity measure, e.g. softmax normalized by all words, which is the per word probability given its position. u_w is a high-level context vector that is jointly or implicitly learned during back propagation. This learned magnitude α is then factored in to form the sentence vectors.

The second part works on the sentence level and is basically the same as the first part. Instead of words, the output of the first part, the sentence vectors are used as input. The resulting vector is a high-level representation of the document, formed from the extracted meaningful sentences. A final softmax layer is responsible to classify these into the different categories.

3.4.2 Experiment

Yang et al. (2016) create 200-dimensional word embeddings using word2vec. They only add words to the vocabulary which appear at least 6 times in the corpus. This is a huge difference to our approach of using the pre-trained 300-dimensional FastText embeddings and attempt to minimize out-of-vocabulary words.

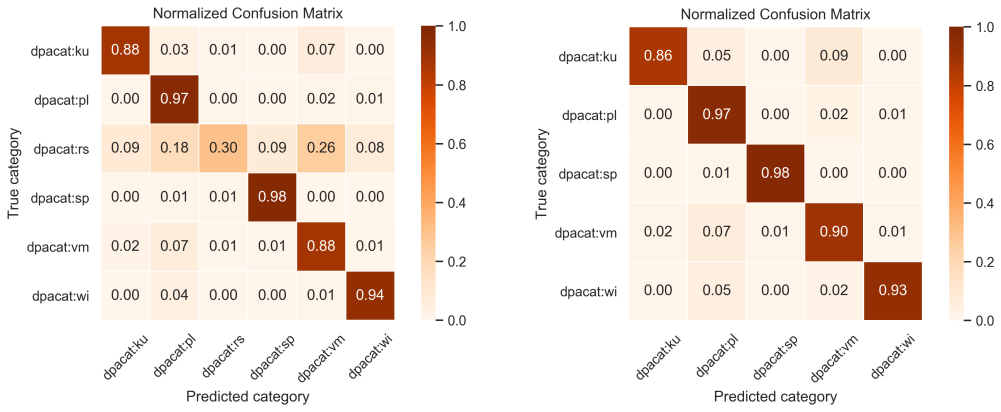
		With Editorial Service			Without Editorial Service		
Category	Field	Precision	Recall	F1 score	Precision	Recall	F1 score
Culture	dpacat:ku	0.74	0.88	0.81	0.93	0.86	0.89
Politics	dpacat:pl	0.86	0.97	0.91	0.93	0.97	0.95
Editorial Service	dpacat:rs	0.91	0.30	0.45	n/a	n/a	n/a
Sports	dpacat:sp	0.92	0.98	0.95	0.98	0.98	0.98
Mixed	dpacat:vm	0.78	0.88	0.83	0.92	0.90	0.91
Economics	dpacat:wi	0.89	0.94	0.91	0.97	0.93	0.95
Micro average		0.85	0.85	0.85	0.94	0.94	0.94
Macro average		0.85	0.82	0.81	0.95	0.93	0.94
Accuracy			0.8562			0.9430	
Loss			0.2876			0.1799	

Table 6: HAN results

We used the same hyperparameter as for the other models and outlined in Section 3.2.2 with the following exceptions. Due to memory constraints of the GPU, we had to reduce the amount of data. As a result, we applied a reduced batch size of 32 (Yang et al. (2016) use 64), a maximum sentence length of 100 words and maximum document length of 15 sentences. We did not fine-tune these parameters to exactly fit the used hardware, so there is some potential of tuning left. We also had to train this model on a single GPU, as we could not get it to work reliable on multiple GPUs. Both models with and without dpacat:rs converged within four epochs.

3.4.3 Results

As the model implies, this should work well for a classification task. And indeed, it shows very good results. It is interesting how its performance is in regard to *dpacat:rs*. Its recall result of 0.30 for this category is the worst of all the models, while on the other hand the precision result of 0.91 is far better than in all the other models. This means that if content is labeled as *editorial service*, this categorization is mostly correct, but that 70% of the content of this category are not recognized as such. This is a typical trade-of between precision and recall. Depending on the actual use case, this can be an advantage.



(a) Trained with category *dpacat:rs*

(b) Trained without category *dpacat:rs*

Figure 6: Confusion matrices for *HAN 2016* models

3.5 Universal Language Model Fine-tuning for Text Classification (*ULMFiT*)

Since 2017 there have been several new ideas to apply language modeling as a generalized preprocessing task. There has been wider work on this topic before, but none that were either trackable or worked as well as task specific models. The basic idea is to pretrain a language model on a well defined corpus like Wikipedia and then fine tune this model given the domain specific dataset in our case the dpa dataset. This idea is called transfer learning and is a reasonably new trend in natural language processing. At the heart of ULMFiT by Howard and Ruder (2018) is the AWD-LSTM by Merity et al. (2017) which is a 3 layer long-short term memory network leveraging advanced regularization strategies to perform state of the art language modelling on words.

Since RNNs and LSTMs are prone to overfitting, regularization techniques like dropout and L2 regularization make a huge difference in terms of validation performance. Since our main goal is to learn generalized text features that can be reused on multiple corpora, it is unfortunate that applying techniques like dropout are not easily incorporated in LSTMs. The authors of Merity et al. (2017) employ different regularization techniques to deal with problems of overfitting and subsequently underperforming on domain specific datasets. They found that DropConnect had the highest impact, where instead of applying dropout to the activation functions it is applied on the weight matrices. In addition they replaced gradient descent with average stochastic gradient descent ASGD, where the learning rate and backpropagation window (BPTT) is dynamically adjusted.

3.5.1 Model and Training

We held the preprocessing simple, by only keeping 30000 tokens with the highest term frequencies in the train and test set. Since ULMFiT learns a new language model from scratch, we did not employ any embeddings on the words. We did not use any sentence piece tokenization which could improve results. On the word and sentence level we additionally employed the following preprocessing steps

1. Lowercasing
2. Removal of newlines and multiple spaces
3. Adding of space around some characters[4]

ULMFiT has three major steps that produce subsequent data for ingestion, as depicted in Figure 7. In the following we will do a simple walkthrough of these three steps and how we adjusted the parameters to our model.

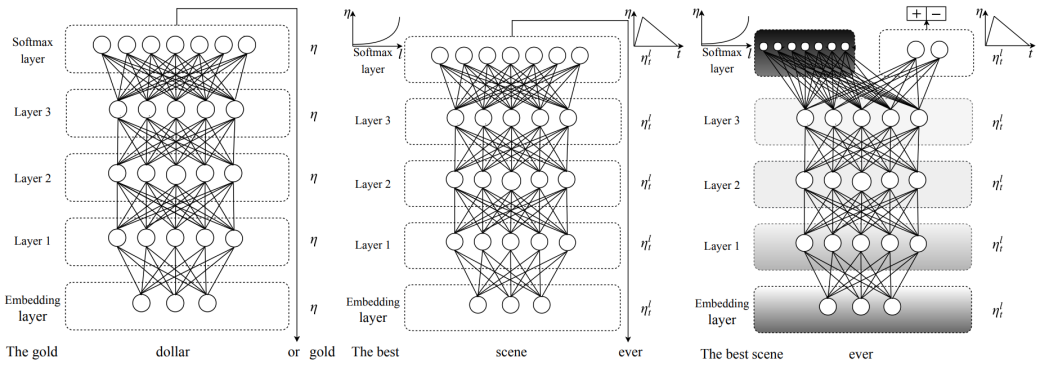


Figure 7: *ULMFiT* basic idea as described in Howard and Ruder (2018)

3.5.2 Pretraining

The first step is learning a general language model with the AWD-LSTM on the Wikipedia dataset. Since we work with a German language corpus we could not use a pretrained language model for

[4] That concerned quotation marks, brackets, exclamation marks, semicolons and colons.

English. Instead we trained one ourselves on the German Wikitext-103 dataset Merity et al. (2016) that consist of 28,595 preprocessed German articles with roughly 103 million words. The pretraining is not of much interest, since we took the standard parameters and trained the AWD-LSTM for 12 epochs, with a batch size of 192, which took 10 hours to complete on a Tesla V100. Fortunately this step has to be done only once, since it is the general pretrained language model.

3.5.3 Fine-tuning

The second step is to fine tune the Wikipedia language model with the dpa text dataset. During this step two novel techniques called discriminative fine-tuning and slanted triangular learning rates (STLR) are employed to adapt the general language model to our domain. Our results looked okay, but could be improved by finetuning the hyperparameters. We achieved a cross entropy training loss of 3.16 and validation loss of 2.78, with a final accuracy of 0.46. This took approximately 12 hours to train again on a Tesla V100.

Discriminative fine-tuning is a technique that adapts the learning rates for each layer, using the following gradient update rule

$$\theta_t^l = \theta_{t-1}^l - \eta^l \cdot \nabla_{\theta^l} J(\theta) \tag{5}$$

where η is the learning rate adapted with some exponent l that is adjusted on a given per layer basis. In practice the last layer of the language model is fine-tuned first, then moving down the learning rate decreasing by a constant factor of $\eta^{l-1} = \eta^l / 2.6$. Intuitively, the first layer contains finegrained features and the deeper we go, information are more general and features become more abstract.

Slanted triangular learning rates (STLR) are different learning rates that increase linearly at the beginning and decreasing slowly after. The intuition is that the target task, meaning the Wikipedia language model has a different distribution from the dpa dataset. Therefore we need to move the parameters of the Wikipedia dataset adjusting to the dpa dataset. After a small bursting period where the pretrained language model is adjusted we slowly decrease the learning rate. This shifts the distribution of parameters to the dpa dataset while retaining useful information from Wikipedia mitigating domain drift.

3.5.4 Classification and Experiments

The third and last step uses the fine-tuned language model training the multilabel classifier with gradual unfreezing and again STLR. Howard and Ruder (2018) also found that a bidirectional language model increases accuracy, instead of training one pretrained and fine-tuned AWD-LSTM, in reading direction, we also do this in reversed direction. Unfortunately we did not try if this had any effect on the results. Ensembling both models yields 1-2% accuracy.

Gradual unfreezing is the process of unfreezing one layer at a time beginning with the last and therefore most abstract one. The reasoning is that aggressively classifying through 1-2 fully connected layers essentially overwrites the learned weights of the fine-tuned language model. Each epoch additional weights are fine-tuned until all weights are trained at the same time. In the following we compare outcomes of different dataset splits and preprocessing strategies. As depicted in figure 7 we will especially focus on tests with and without the editorial service. In the following figures we see the confusion matrices where higher values of accuracy are dark-red while smaller values approach yellow and white.

As we can see it makes a huge difference keeping or dropping the editorial service from our dataset, yielding a 9-10% increase in accuracy. With editorial service we have a recall of 50% which means that 50% of editorial service documents are incorrectly classified as not editorial service, while the precision is also underwhelming with 72%, meaning that we predicted an incorrect class 30% of the time. This drastically changes when the category is missing, e.g. all scores go above 90%, which can be seen in the F1 score which combines precision and recall, on average going up 5%. The split in most experiments is 80% training and 20% test set.

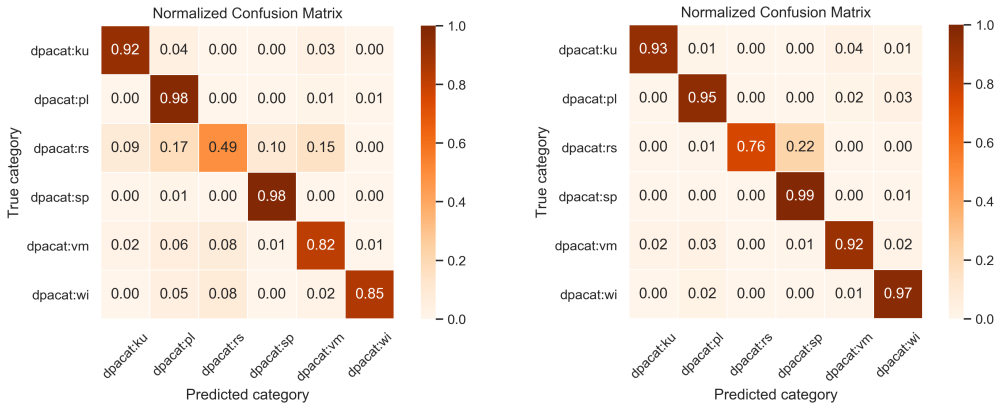
		With Editorial Service			Without Editorial Service		
Category	Field	Precision	Recall	F1 score	Precision	Recall	F1 score
Culture	dpacat:ku	0.75	0.92	0.83	0.94	0.91	0.93
Politics	dpacat:pl	0.87	0.98	0.92	0.94	0.97	0.96
Editorial Service	dpacat:rs	0.72	0.49	0.58	n/a	n/a	n/a
Sports	dpacat:sp	0.91	0.98	0.95	0.99	0.98	0.99
Mixed	dpacat:vm	0.84	0.82	0.83	0.95	0.92	0.93
Economics	dpacat:wi	0.97	0.85	0.91	0.97	0.94	0.95
Micro average		0.86	0.86	0.86	0.96	0.96	0.96
Macro average		0.84	0.84	0.84	0.96	0.95	0.95
Accuracy			0.8623			0.9559	
Loss			0.3239			0.1101	

Table 7: ULMFiT results

3.5.5 80-20 with rs and relabel

A major problem with the editorial service is that it contains articles that are duplicated in other categories. Those are mainly summaries about a respective category, with the wording and meaning of that category, that is not the editorial service. In this experiment we see how relabeling the editorial service category to its respective category dramatically improves performance. The relabeling in the following was done on the training dataset. We have to note here that relabeling is drastically influenced by the test split and in reality is part of an inconsistency in the dataset. It makes sense to relabel the entire dataset or at least use a unique constraint on the documents.

On the left in figure 8 we see the confusion matrix where the editorial service is left in. As we can see on the right when relabeling the editorial service we tremendously decrease the misclassification rates between the editorial service and all other categories. This increases the recall by almost 40%.



(a) Trained with category *dpacat:rs*

(b) Trained with category *dpacat:rs* with relabeling

Figure 8: Confusion matrices for *ULMFiT 2018* models

From the images it becomes clear that classifying economics and politics, mixed and editorial service vs. everything else is much harder than say cultural and sports. Interestingly in the relabel case we increase the rate of misclassification with sports and editorial service. Since the editorial service contains duplicated texts from all categories (almost 13.000 documents) and are therefore impossible to classify correctly. Howard and Ruder (2018) claim that using much less data still yields a good accuracy. We did an additional run and trained a model on 20% training and 80% test data. Our accuracy comparably dropped to 84.22% which we believe is good when considering that we tested on 4 times the training data.

4 Conclusion

We trained and evaluated all models with and without the category *dpacat:rs*. The results are presented in Table 8. Note that accuracy and loss are calculated during training on the validation set, while the F1 score is calculated using the trained model on the test set.

With Editorial Service included, F1 score and accuracy are almost the same for all models and there is no clear winner on the overall results. Without Editorial Service, ULMFiT performs the best on all F1 score, accuracy and loss. But again, the results do lie close to each other for all models. The imbalance of the Editorial service lies in duplicated texts across categories. No model is able to make sense of a text occurring in two categories and assigning it a distinct label.

Even though all models achieve roughly the same performance levels, the models each have their own strengths and weaknesses. As we can see in the confusion matrices in the respective sections, there are clear differences how the models perform per category, especially when including *dpacat:rs*. Each model is capable to a different degree to distinguish *dpacat:rs* from the other categories and categorize either based on its actual content or its category. For example the DCNN has a comparatively high recall of 0.75 for the editorial service, but lacks in precision. HAN on the other side has a rather low recall of 0.30, but a rather high precision. A consequence of this different behavior is that there is not *the universal best model*, but you have to look at your requirements.

	With Editorial Service			Without Editorial Service		
Model	F1 score	Accuracy	Loss	F1 score	Accuracy	Loss
FastText	**0.86**	0.8548	0.3302	0.95	0.9519	0.1721
CNN	0.85	**0.8660**	0.6134	0.93	0.9380	0.2536
DCNN	0.84	0.8474	0.3155	0.94	0.9351	0.2082
HAN	0.85	0.8562	**0.2876**	0.94	0.9430	0.1799
ULMFiT	**0.86**	0.8623	0.3239	**0.96**	**0.9559**	**0.1101**

Table 8: Comparison of all models. F1 score based on test set, accuracy and loss on validation set.

With the exception of ULMFiT, surprisingly few training epochs were needed for convergence. The reason likely lies in the large data set. With around 200 000 items for five to six categories, there was no need for the model to iterate very often over it to learn about its specifics. As could be expected, more epochs did result in overfitting.

All in all, each classifier has its up and downsides. Given a production environment where speed and resources are important we found FastText to be the most impressive classifier. Even on weak hardware its simple model takes just a few seconds to minutes to complete the training process. The multinomial naive Bayes is in this regard equally impressive, being even faster than FastText, but very prone to overfitting and not easily dealing with OVV words. In contrast, if you have the resources and the need for very high accuracies ULMFiT might be the choice. The downside is that the training can easily take multiple days, with the upside of training an AWD-LSTM language model that can be reused in other NLP tasks.

Our findings suggest that modern neural network architectures outperform baseline bag-of-words models by a small margin. For us this implies that classification might not be a suitable task that leverages the advantages of neural networks at whole. Neural networks excel at tasks that are inherently difficult to model such as sequence alignment, generation of unseen sequences and fluency of text. Since accuracy and loss are quantitative measures, factors like speed, space, explainability or simplicity of the model are often not discussed. The choice for the right classifier should be based on the setting it is applied in.

Acknowledgment

We thank Dr. Gerd Kamp for the great dataset and Professor Kai von Luck and all people at the CSTI for their support and especially for making the hardware available.

References

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *CoRR*, abs/1607.04606.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning Word Vectors for 157 Languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Halbritter, S. and Nitsche, M. (2019). Development of an End-to-End Deep Learning Pipeline.

Howard, J. and Ruder, S. (2018). Fine-tuned Language Models for Text Classification. *CoRR*, abs/1801.06146.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2016). Bag of Tricks for Efficient Text Classification. *CoRR*, abs/1607.01759.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. *CoRR*, abs/1404.2188.

Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. *CoRR*, abs/1408.5882.

Merity, S., Keskar, N. S., and Socher, R. (2017). Regularizing and Optimizing LSTM Language Models. *CoRR*, abs/1708.02182.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models. *CoRR*, abs/1609.07843.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, abs/1310.4546.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., and Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489. Association for Computational Linguistics.