Movie Review Sentiment Analysis

David Hennemann
Hasso-Plattner-Institute
david.hennemann@student.hpi.de

Ivan Ilic
Hasso-Plattner-Institute
ivan.ilic@student.hpi.de

Jan Westphal
Hasso-Plattner-Institute
jan.westphal@student.hpi.de

Mats Pörschke
Hasso-Plattner-Institute
mats.poerschke@student.hpi.de

Maximilian Götz
Hasso-Plattner-Institute
maximilian.goetz@student.hpi.de

Abstract

Deep learning methods became increasingly efficient in solving Natural Language Processing (NLP) tasks. This paper will take a deep dive into various neural network architectures for sentiment analysis of human-written texts as a subfield of natural language classification. Whereas several subfields of NLP like natural language translation involve highly complex tasks, comparably simple models can achieve decent performances in natural language classification tasks. But in order to achieve superior results more sophisticated approaches are required. We will explain the used methods and concepts in detail, evaluate the approaches and compare them with each other to point out both advantages and disadvantages and give advice on when each approach should be used.

1 Introduction

Natural language classification algorithms have been subject to research for many years. This project explores a subset of the well-working approaches and implements adaptions of those in order to fit into our problem context. However, incorporation and adaption to the specifics of the problem involves difficult tasks which includes preprocessing the dataset efficiently, finding suitable text encodings and finally improving the machine learning model in order to achieve a higher prediction accuracy.

Sentiment analysis can be described as processing natural language text in order to output binary information that classifies a text as either having a positive or negative sentiment. This task can be solved by various neural network model architectures using different parameter configurations and therefore leaves a lot of space for exploration. Furthermore, our goal includes identifying words that are key to distinguishing between these two categories. We want to answer the question, which words are most *impactful* for the neural network to assign a text to its respective class.

2 Related Work

Recurrent Neural Networks (RNNs) are specifically designed to process sequential data and therefore well suited for NLP tasks as they are able to capture the context of sentences (all previous occurrences of words) in their hidden state [1]. A major downside of classical RNNs is the missing ability to reliably perform long-term memorization. The impact of words at the beginning of a sentence gets

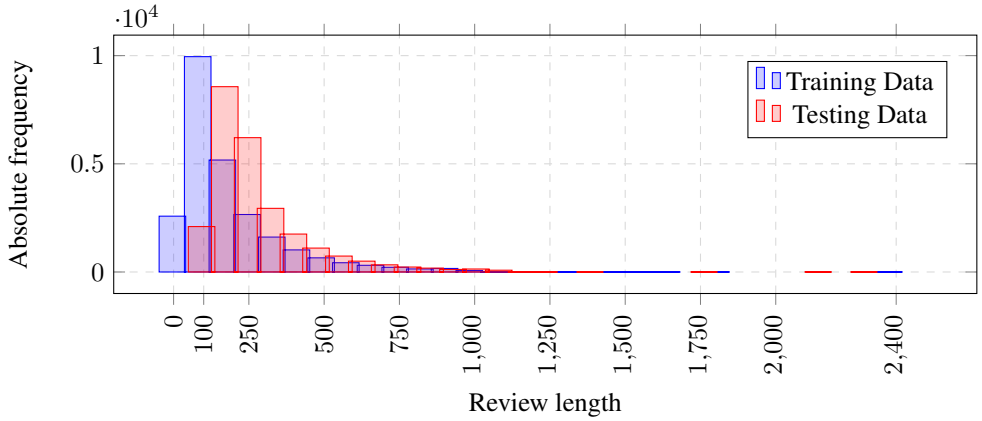

Figure 1: Review lengths in training and testing data

lost over time which is especially problematic when processing very long sequences. In tackling this issue, Long Short-Term Memory (LSTM) models were introduced which use a more sophisticated internal structure, called a gating unit [2]. Bidirectional RNNs use multiple hidden layers of opposite directions in order to learn from the past and the future simultaneously [3].

Pengfei Liu et al. employ the concept of multi-task learning to text classification to achieve better performance [4]. Siwei Lai et al. combine the ability of a bidirectional RNN to capture contextual information with the strengths of a max-pooling convolutional layer to identify key features of texts [5]. The approach of combining Convolutional Neural Networks and RNNs has also been done in the context of sentiment analysis showing improvements to prior solutions [6]. In recent years a new type of architecture, called transformers, became state-of-the-art by breaking various kinds of NLP benchmarks [7]. A Facebook AI Research group evaluated a baseline for text classification where they used a bag-of-words sentence representation with a linear classifier [8]. They are able to scale to big text sizes while keeping the model training time low. This simpler approach was shown to be very much comparable with state-of-the-art deep networks in terms of accuracy.

3 Contribution

3.1 Dataset

This project uses a movie sentiment dataset presented by Maas et al. [9] It consists of an overall number of 50,000 reviews which the authors applied a 50:50 split to, resulting in a training and testing subset with 25,000 reviews each. Within those two subsets there are 12,500 reviews belonging to the binary classes of either good or bad reviews respectively. While data subsets are equal with regards to the number of reviews and the class distribution, one can find a difference in the review lengths. We can see in Figure 1 that reviews in the testing data tend to be slightly larger than those within the training data. Whether this has an effect on a model's generalization error trained on the training data can be seen in Chapter 4. Moreover, the large majority of reviews within the testing and training data consists of less than 1,000 words.

3.2 Bag-of-words

One of the more straightforward and easy to train solutions is a dense neural network with a single output using a bag-of-words representation as input. We will refer to this architecture as bag-of-words model.

Preprocessing The bag-of-words model applies two common preprocessing steps to the input data. First, brackets, arrows and other punctuation marks are removed. This reduces the number of different words by a factor of roughly two and thereby helps to reduce the number of model parameters dramatically. Second, the encoding is generated by converting a sequence of words of arbitrary length into a fixed-sized vector representation. In order to do so, we define the vocabulary

as all distinct words within the training dataset. As a result, words that are not present within the training dataset need to be ignored during evaluation. Each individual review is transformed into a vector with a length equal to the vocabulary size which can be interpreted as one histogram per review. There are many different techniques that can be used for the transformation step. We focused on two of them – namely the `CountVectorizer` and `BinaryVectorizer`.

> **CountVectorizer** As the name already suggests, the `CountVectorizer` counts the frequency of a word occurring within a single review. Thereby, the index assigned to the word "movie" contains the absolute frequency "movie" occurred with.
>
> **BinaryVectorizer** The `BinaryVectorizer` we implemented simplifies the idea of the `CountVectorizer` by just indicating whether a word was present within a review. Instead of telling the absolute frequency, a word-index is set to 1 if the review contains the word at least a single time and 0 otherwise.

Structure As we saw in 3.2 each review of arbitrary length is transformed into a fixed-sized vector. This is independent of the actual used vectorizer and therefore allows us to apply a simple dense layer to the vectorized reviews, similar to usual classification tasks. The dense layer was set up in a way, that it has a single output neuron with the sigmoid function applied to afterwards. As a result, the network outputs the probability for a single review to have a positive sentiment.

3.3 LSTMClassifier

The `LSTMClassifier` aims to overcome a major downside of the bag-of-words models from 3.2 which cannot take the sentence structure i.e. the order of words into account for their decision making process. As the `LSTMClassifier` model is structurally completely different from the bag-of-words model, we will start explaining the required preprocessing steps before going into detail of the model structure.

Preprocessing The `LSTMClassifier` implementation uses four preprocessing steps. Similar to the bag-of-words model we start by removing all punctuation marks. Then, lengthy reviews are removed to speed up the training process while keeping the major portion of the data, more specifically, we remove all reviews containing more than 1,000 words. Third, the preprocessor matches the sequence length of all reviews. Those with less than the maximum number of words are padded with a *start of sequence (<SOS>)* token at the beginning to fit the maximum length. This token tells the LSTM cells that the sequence will start with the next input. Fourth, the words contained in the training data are tokenized, meaning that all distinct words are assigned a unique ID. By doing so the characters and words are translated into a numeric representation. Instead of sequences of characters, the data now consists of sequences of integers.

Structure The model is responsible for interpreting the numeric sequences and performing the classification task. Therefore, it can be divided into two major parts. The first of those is called encoder and the second one performs the actual classification task. An overview of the model's structure can be seen in Figure 2.
Within the encoder, two different approaches were combined. On the one hand, the integer sequences are embedded into a vector representation. As the word indices are in the range from 0 to $\sim$64,000, neural networks can have a tough time approximating the desired function due to unstable input distributions [10]. One possible solution to address this problem is using one-hot-encoding where each integer is converted into a vector containing all zeros but at the word index. There, the entry is set to one. The vector size here is equal to the number of distinct words. Another widely spread approach that can be seen in e.g. Word2Vec models is learning a vector representation based on the training data [11]. Thereby, one can use way less dimensional vectors compared to the approach above. This helps to reduce the model size drastically but also introduces another hyper-parameter which is subject to tuning. As many modern neural network architectures seem to profit from trainable embeddings, we decided to make use of them as well [12][13]. Figure 2 shows that the embedding is of dimensionality `B x PAD x EMB`. Whereas `B` indicates the batch size used for training, `PAD` indicates the maximum sequence length as explained in the preprocessing paragraph and `EMB` refers

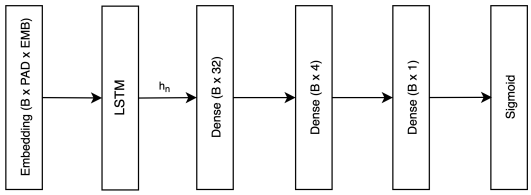

Figure 2: Schema of LSTMClassifier

to the size of the embedding. `EMB` is a hyper-parameter that is subject to tuning and will be further addressed in Chapter 4.

The embedding layer output is fed into an LSTM cell to generate a context vector containing all necessary information on the review. Resulting from that, we discard the LSTM cells' output, as it is not needed for this approach. LSTM cells internally use gates which helps with remembering important information over a long period of time [2]. This way the context vector is not that prone to forgetting information retrieved from the very first items within a sequence.
The second major part of our classifier builds up on the context vector generated in the previous step. Here, a feedforward neural network is used to predict the sentiment. It uses two hidden layers each activated by the rectified linear unit (ReLU) function. In order to perform the two-class classification task, the sigmoid function gets applied to the last layer's output neuron.

Regularization Regularization can be used to prevent overfitting on the training data. Very famous among the many possible methods are weight decay and early stopping which both were used when training the `LSTMClassifier`. Therefore, we made use of the hyper-parameter `weight_decay` of the Adam optimizer used for training and implemented an early stopping regularizer with a default `patience` of `5`. A model that is subject to training can benefit from both regularizers as they help to reduce the risk of overfitting [14][15].

3.4 BERT

Since 2018 several new deep pre-trained language models have been introduced. All of them achieved better results than task specific deep learning models (e.g. models based on LSTM).
One of the newly introduced models is BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) by Google [16]. It has been pre-trained unsupervised on a large document-level corpus with 3,300M words. A large part of the corpus consisted of English Wikipedia. One way the model gets pre-trained is by randomly masking some of the input tokens. The model predicts the id of the token based on the context left and right to the masked token. We will omit an exhaustive part of the model architecture and the details on how the model got pre-trained and refer readers to the original work by Devlin et al. [16].

Based on the results BERT achieved in various NLP benchmarking tasks like text classification, question answering and the prediction of the next sentence, it is safe to say that BERT has a good internal representation of the English language [16]. Another NLP task, BERT achieved great results in, was sentiment classification. Therefore, we examine how BERT performs on the classification task of the sentiment of movie reviews. Please note that given the transformer architecture, BERT is only able to process the first *512* tokens. This could affect the prediction performance on long reviews but we expect the reviewers to express the sentiment within the first sentences of their review.

Fine-Tuning An important part of applying BERT is fine-tuning the model to the specific task. By fine-tuning the model we need way less data and computation time than building a model of this magnitude from scratch. The fine-tuning is straightforward given the nature of the model and various open-source libraries like *Transformers*[1] by HuggingFace [17]. In addition, there are a lot of guides on how to apply the library to a specific use case[2].

[1] https://github.com/huggingface/transformers

[2] For example, Chris McCormick and Nick Ryan. (2019, July 22). BERT Fine-Tuning Tutorial with PyTorch. Retrieved from http://www.mccormickml.com

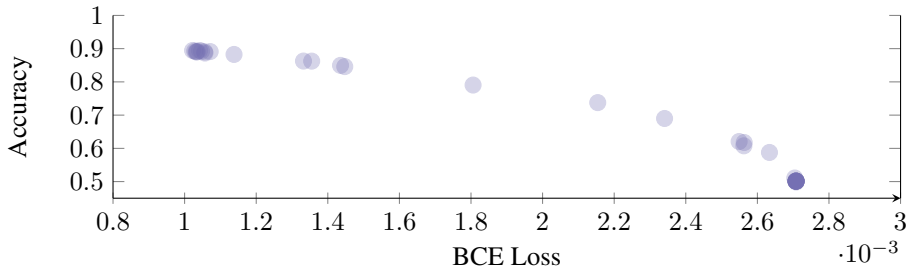

Figure 3: Validation loss and accuracy of different hyper-parameter configurations

4 Training and Evaluation

The following will elaborate on the training and resulting performances of the individual approaches. During the training procedure a validation dataset was split off the training dataset randomly to identify the expected generalization error. The eventually performed hyper-parameter optimization was also performed using the training and validation set only. All models were finally evaluated on the testing dataset provided with the dataset [9].

4.1 Bag-of-words

The major advantages of the bag-of-words model are its simplicity and its interpretability. Since we only have a single dense layer, which connects all dictionary words with a single output, we can expect our network to assign all the positively connoted words a large weight and all the negative connoted words a small (large negative) weight. After 3 epochs of training, the network is already capable of identifying word sentiments reasonably. The Listings 1 and 2 show the highest and lowest lowest weighted words. Most of them are rather self-explanatory. While the *7* and *8* are not intuitively associated with a positive sentiment, they might stem from the ratings like *7/10* and *8/10* being associated with positive reviews. The $\sim$ on the other hand does not have an intuitive explanation for being weighted negatively, but shows nicely that the model does not have a deep understanding of the English language.

```
7 (1.0401), excellent (0.9463), 8 (0.8799), refreshing (0.8532),
superb (0.8092), flawless (0.8052), hooked (0.7822),
perfect (0.7795), rare (0.773), favorite (0.7662)
```

Listing 1: words with largest weights

```
worst (-1.2656), waste (-1.147), disappointment (-1.0781),
poorly (-0.9816), awful (-0.9292), unfunny (-0.8798),
disappointing (-0.8785), uninteresting (-0.8443), ~ (-0.8336),
boring (-0.8218)
```

Listing 2: words with smallest weights

The model achieved an accuracy of $\sim$88% on the testing data. This result is already better than our initial expectation. In Section 3.2 we introduced two different vectorizers. We initially thought that using the word vectorizer, which takes the absolute frequency of a word into consideration, would increase the accuracy, since repeating a word several times can be an indicator of the sentiment of a review. At least for our dataset this was not the case. One possible interpretation could be, that a lot reviews in our dataset, are just too short to properly repeat words on a regular basis. Both vectorizers achieved nearly the same performance ($\Delta 0.2\%$).

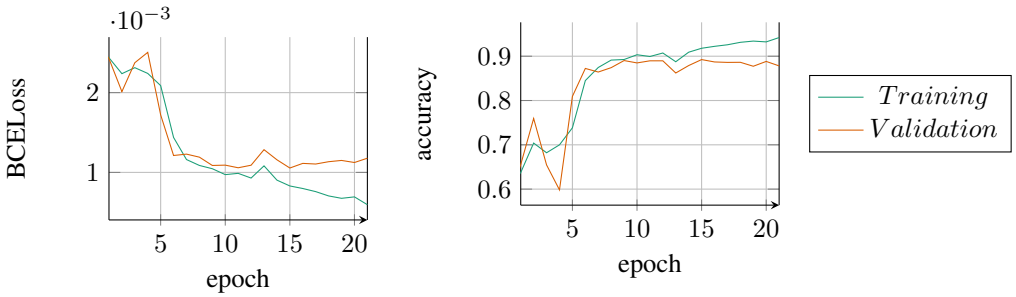

Figure 4: Losses and accuracy scores of `LSTMClassifier` on training and validation data

4.2 LSTMClassifier

Hyper-Parameter Optimization The `LSTMClassifier` contains several different hyper-parameters that need to be optimized to find a well performing configuration. In order to do so, we implemented the grid search algorithm in `GridSearch` which does a random search within a given parameter subspace. We evaluated 36 different parameter configurations generated from the following parameter grid. Each of those configurations was executed with a maximum of 50 epochs, but all were terminated by the early stopping regularization latest after the 35$^{\text{th}}$ epoch.

```
param_grid = {
    batch_size: [256, 512],
    embedding_size: [40, 80, 100],
    hidden_size: [32, 64, 128],
    learning_rate: [1e-2],
    weight_decay: [3e-3, 1e-3]
}
```

The results of this hyper-parameter search can be seen in Figure 3. Whereas some configurations end up with a result similar to random guessing by achieving a validation accuracy of $\sim$50%, others are able to achieve quite a decent performances with accuracies of $\sim$90%. The best performance within this grid search with a binary cross-entropy (BCE) validation loss of $\sim$0.001 and a validation accuracy of $\sim$90% was achieved through the configuration `batch_size: 256, embedding_size: 100, hidden_size: 64, learning_rate: 0.01, weight_decay: 0.001`. During the process of training the `AdamOptimizer` was used. We can see in Figure 3 that quite a large number of configurations end up having validation accuracies of slightly below 90%.

Training optimized model The following will give a short introduction into an exemplary training process of the `LSTMClassifier`. Figure 4.2 shows the losses and accuracy scores during training on the training and validation data of the model with the best performing hyper-parameter configuration. One can see that the BCE loss decreases both in validation and training data up to epoch 16. From there on, the training loss decreases further, but the validation loss starts increasing again which is a strong indicator of overfitting on the training data. Similar effects are visible in the accuracy over time. The accuracy on the validation data stops increasing from epoch 16 onwards while the accuracy on the training data is still rising up to epoch 21. Here, the above discussed regularization methods come into place. The early stopping mechanism recognizes the potential overfitting by tracking the validation loss over time. Once the validation loss does not decrease further for a given number of epochs, it stops the training process. This was the case after epoch 21. The model parameters computed in epoch 16 are finally taken as the training procedure's output.

Performance on Testing Data The `LSTMClassifier` is able to achieve an accuracy score of 89% on the testing data. As both accuracies on the validation and testing data are very close to each other, we can conclude that the trained model has a small generalization error.

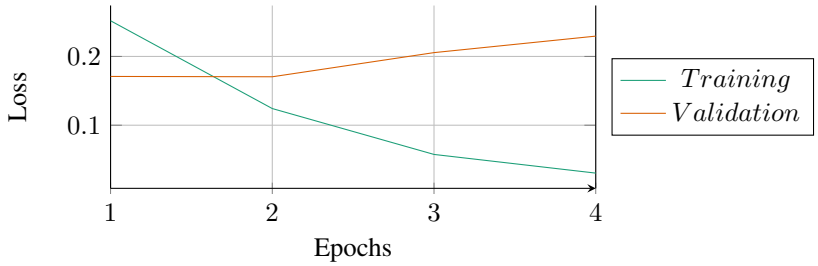

Figure 5: The progress of training and validation loss of the fine-tuned BERT model recorded after each training epoch. While the training loss decreases constantly, the validation loss increases after the second epoch. This leads to the conclusion that after the second epoch the model starts to overfit.

4.3 BERT

We chose a learning rate of `2e-5` (Adam) and a batch size of `32` in accordance with the suggestions of Devlin et al. Further, they recommend to fine-tune the model for 2, 3 or 4 epochs. Using a part of the training dataset as a validation dataset, we came to the conclusion that two epochs are sufficient for training as we would otherwise strongly overfit the training data. This can be seen in Figure 5 as the loss on the validation dataset increases after the second training epoch.
With the uncased base model of BERT (110M parameters) we were able to reach a test accuracy of 94%.

5 Conclusions and Future Work

In 4.1 we saw that linear models perform surprisingly well on this sentiment analysis task. Compared to the bag-of-words model the `LSTMClassifier` wasn't able to reach a superior performance. Both might be caused by the task's quite simplistic nature. But we can see a huge difference in model confidence between the two architectures. In general, the `LSTMClassifier` predicts the sentiment with higher confidence scores than the bag-of-words model. While linear models are fairly robust when it comes to overfitting on the training data, more complex models tend to overfit more strongly. We saw this in 4.2 during the evaluation of the `LSTMClassifier` where we used regularization techniques such as early stopping and weight decay to prevent that. Nonetheless, very complex models such as BERT are able to perform way better than the two simpler approaches. BERT seems to profit from a good understanding of a word's meaning in its context as BERT was trained on a very large text corpus. Similar to the `LSTMClassifier` we saw in 4.3 that BERT is also prone to overfitting, but early stopping also solved the issue here.

When aiming to reach state-of-the-art performances in NLP tasks, we would recommend to make use of large pre-trained models such as BERT. Unfortunately, those models tend to be rather uninterpretable. Perfectly interpretable models can be reached by simpler models such as the linear bag-of-words model described above.

Future work might include the evaluation of the large model of BERT (340M parameters). Devlin et al. showed for various tasks that one can expect to improve classification performance further [16]. Another interesting model architecture, to take a deep dive into, could be a bidirectional LSTM using an attention mechanism. First experiments we did in this field seemed promising as this architecture is capable of taking a look at all inputs a second time, after evaluating the sequence as a whole. Using this approach, there is no need to remember the whole sentence within the LSTM hidden state anymore.

As described in this paper, we implemented multiple neural networks with various kinds of models. Evaluating their performance shows advantages and disadvantages of each of those techniques. In conclusion, there are multiple valid approaches to this sentiment analysis task, as is reflected in the related literature. Therefore, the approach to choose strongly depends on the use case, the requirements concerning the transparency of the decision making process, the required accuracy and the available computational resources.

References

[1] Elvis S. *Deep Learning for NLP: An Overview of Recent Trends*. 2018. URL: https://medium.com/dair-ai/deep-learning-for-nlp-an-overview-of-recent-trends-d0d8f40a776d.

[2] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (1997), 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[3] M. Schuster and K.K. Paliwal. "Bidirectional Recurrent Neural Networks". In: *Trans. Sig. Proc.* 45.11 (1997), 2673–2681. ISSN: 1053-587X. DOI: 10.1109/78.650093. URL: https://doi.org/10.1109/78.650093.

[4] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. "Recurrent Neural Network for Text Classification with Multi-Task Learning". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI'16. New York, New York, USA: AAAI Press, 2016, 2873–2879. ISBN: 9781577357704.

[5] Siwei Lai et al. "Recurrent Convolutional Neural Networks for Text Classification". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI'15. Austin, Texas: AAAI Press, 2015, 2267–2273. ISBN: 0262511290.

[6] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. "Combination of Convolutional and Recurrent Neural Network for Sentiment Analysis of Short Texts". In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, 2016, pp. 2428–2437. URL: https://www.aclweb.org/anthology/C16-1229.

[7] Ashish Vaswani et al. "Attention is All You Need". In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf.

[8] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, 2017, pp. 427–431. URL: https://www.aclweb.org/anthology/E17-2068.

[9] Andrew L. Maas et al. "Learning Word Vectors for Sentiment Analysis". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, 2011, pp. 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[10] Shibani Santurkar et al. *How Does Batch Normalization Help Optimization?* 2018. arXiv: 1805.11604 [stat.ML].

[11] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL].

[13] Shervin Minaee et al. *Deep Learning Based Text Classification: A Comprehensive Review*. 2020. arXiv: 2004.03705 [cs.CL].

[14] Rich Caruana, Steve Lawrence, and Lee Giles. "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping". In: *Proceedings of the 13th International Conference on Neural Information Processing Systems*. NIPS'00. Denver, CO: MIT Press, 2000, 381–387.

[15] Anders Krogh and John A. Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Proceedings of the 4th International Conference on Neural Information Processing Systems*. NIPS'91. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1991, 950–957. ISBN: 1558602224.

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*. 2019.

[17] Thomas Wolf et al. "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *ArXiv* abs/1910.03771 (2019).