A performance comparison of neural network architectures for single-sentence classification tasks

Marc Lüttecke*
Department of Economics
marc.luettecke@uni-konstanz.de
Student ID: 910426
Advisor: Dr. Lyudmila Grigoryeva

Abstract

This overview details an applied approach to a systematic comparison of neural network architectures for a single-sentence text classification task. Fine-tuning high-dimensional feature vectors (word-embeddings) substantiate as the main driver of accuracy for even the most complex network structures. Generally, more intricate architectures perform better than elementary ones, but the increased accuracy does not outweigh the computational complexity of the classification task. Even simple CNNs reach adequate results if combined with a trainable custom-built embedding. Within more complex models, Attention and state-of-the-art BERT embedding models provide promising results for classification tasks of minimal input size (only up to a couple hundred words per row, often less). The analysis classifies authors for less than 20k total (training and testing) single-sentence examples.

1 Introduction

Even though the rise of digital data allows for advanced analysis, most of the world's information remains in written form. Many of the techniques developed over the years to keep the pace of analysis on par with the pace of data retrieval address numerical data, which computers can process conveniently. This discrepancy is the nourishing ground for the field of *Natural Language Processing* (from here on out "NLP"), which transforms words into machine-readable entities for quantitative analysis.

Within the field of NLP, many different tasks have proven as highly relevant. *Part-Of-Speech tagging*[1], *Chunking*[2], *Named Entity Recognition*[3], and *Semantic Role Labeling*[4], represent some of the core tasks applied to text data. This paper avoids addressing any of these tasks specifically, to recognize a more universal question: What are fundamental features of a classification network to reach maximum accuracy?

To resolve the question, this project represents a systematic comparison of neural network architectures for the task of single sentence classification. The high-relevance of classification tasks with minimal input is exemplified in topics, such as author identification (see early research on plagiarism detection Ma and Hovy [44]), or Twitter user research (research on Twitter bot detection Kudugunta and Ferrara [38]). A simple neural net might prove helpful to at least highlight cases, which seek further investigation.

This short and intentionally less technical write-up aims to build intuition for the main ideas of architectural designs. Section 2 will develop the necessary intuition to understand the details of the research design (further explained in section 3. The discourse starts with an introduction to the baseline model for performance comparison, namely a logistic classifier (section 2.1). It progresses to detail the inclusion of word-embeddings (section 2.2) as input-formatting to neural networks and concludes with the detailed analysis of the included network architectures (section 2.3). The report extends a short description of the data in section 3.1 through visualizations and exploratory data analysis by comparing the models and the utilized hyper-parameters of the architectures. Section 4 closes with results and conclusive remarks.

*Lüttecke is a Masters student in *Social and Economic Data Science* at the University of Konstanz. This research project represents the written report for the *Machine Learning* class taught by Lyudmila Grigoryeva. The author would like to explicitly express his gratitude for the forum to conduct an applied data science research endeavor and the feedback through discussions with Dr. Grigoryeva. The project's entire code can be found under: https://github.com/marcluettecke/ML_project.

1 Automatically derive the role of a word in the sentence, i.e., Noun, Verb, Adverb, etc.
2 Groups words together and classifies them similar to POS tagging for individual words: i.e., noun phrase, etc.
3 Categorization of entities within the text into predefined bins, such as *person*, *location*, *organizations*, etc.
4 Labels the role of a word in the sentence, such as agent, result, location, etc.

2 Technical discourse

This chapter aims to build a common understanding of the more applied discussion of the actual comparative methodology. The explanations are intentionally condensed and only include the necessary details to understand the encompassed vocabulary. Technical mathematical jargon is minimized to the point, where it can prove useful to understand why certain techniques fail and others achieve accurate results.

2.1 Logistic regression

A *Logistic Regression Model* (from here on *Logit model*) is a probabilistic classification algorithm used in machine learning. It is often one of the first concepts taught at universities due to its intuition and computational ease. Traditionally a Logit model is implemented to conduct binary classification task, due to the functional form of the logistic function (or softmax function):

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

which translates to the probabilities of z = 1 and z = 0 of:

$$P(t = 1|z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$
$$P(t = 0|z) = 1 - \sigma(z) = \frac{e^{-z}}{1 + e^{-z}}$$

This form allows the function to bound the output between 0 and 1, which translates naturally into a probability mapping, as depicted in figure 1.

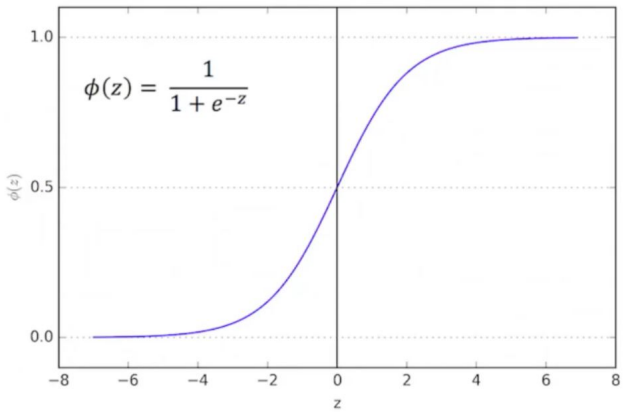


Figure 1: Source: Logistic Regression: A Simplified Approach Using Python. Illustration of how the input of a Logit function is mapped to a 0 to 1 output space, which is often used for probability mappings and classification tasks.

For a multi-label classification task, the binary approach will fail, and the probability function needs the extension to a joint-probability distribution. This adjustment allows the model to process multi-label inputs[5]:

$$P(t, z|\theta) = P(t|z, \theta)P(z|\theta)$$

To then define the objective function[6], we can minimize the negative of the log[7]-likelihood, which is called the *cross-entropy loss*:

$$\begin{aligned}\xi(t, y) &= -\log \mathcal{L}(\theta|t, z) \\ &= -\sum_{i=1}^{n} [t_i \log(y_i) + (1 - t_i) \log(1 - y_i)] \\ &= -\sum_{i=1}^{n} [t_i \log(\sigma(z)) + (1 - t_i) \log(1 - \sigma(z))]\end{aligned}$$

By continuously adjusting the input weights to minimize the objective function, a logistic classifier offers a valid benchmark for our experiment with a minimal computational cost.

2.2 Word embeddings

Typical problems in NLP input stages arise, when text is transformed into computer readable format. Transformation to lower-case of the input, stop-word removal[8], tokenization[9] and lemmatization[10] represent common pre-processing tasks.

Nevertheless, the resulting input matrix is often vast for the most common mapping technique coined *one-hot encoding*[11]. This representation suffers from sparsity problems since a corpus often contains many thousand words, and any sentence input will only contain a small fraction of these words. This discrepancy results in a matrix mainly filled with zeros and a few entries labeled with ones.

5 The calculation surmises the characteristic of a joint probability, namely a probability distribution determined by two freely moving variables, is the same as the probability conditioned on one, times the probability that the conditional variable will occur with the fixed value.

6 *Objective* referring to the function, which throughout the machine learning phase serves as the focal point to address, here to minimize.

7 The logarithmic conversion helps with the exponential component of the original logistic function and can be applied without loss of interpretability to a monotone function.

8 The removal of the most common word of a given language due to their limited explanatory value. Examples are "*it*","yourself",or "by".

9 Splitting a sentence into a list of input tokens: such as "*Are you here?*" → ["*Are*", "*you*", "*here*", "*?*"].

10 Removing a word to its core component, such that "*loving*" and "*loves*" both reduce to "*love*".

11 A technique which summarizes each input sentence into an array of "number of examples" × "unique words in text corpus" with a one if given the word is present in the row of input and 0 otherwise.

A more computationally efficient alternative are *word-embeddings*[12]. A neural network trains a resulting feature representation either on a large general (not the input text) text corpus, which is then forwarded as input to fine-tuning for more specific tasks via transfer learning[13], or, alternatively, a custom embedding can be trained on the input text itself. The vector representations of a word embedding carry semantic meaning, such that tokens close in vector space, induce a closeness in relative interpretation as well. Necessary extensions allow for sentiment specific embeddings Tang et al. [65], contradiction-detection via word embeddings (Li et al. [42], and multiple fine-tuned models, which are systematically compared in Turian et al. [71][14].

2.3 Network architectures

This section will describe the most common neural network architectures from a more theoretical perspective. The main purpose is to build intuition for potential drivers of the results in section 4. Many references along the way allow for a more in-depth overview of the actual methods.

2.3.1 Convolutional neural networks

Convolutional Neural Networks (from here on *CNN*[15]) are a form of feed-forward neural nets[16] and are most prominently applied to problems of computer vision. This structure allows researchers to summarize pixel information by repeatedly stacking so-called *convolution layers* to eventually feed a small array of information into a dense layer for classification, labeling, or other tasks. In NLP, these input vectors are word tokens (either a one-hot encoding or word-embeddings (for details see section 2.2), which then lead to a dense layer for the desired output.

Convolutional layers perform surprisingly well and represent a computationally cheap alternative to more sophisticated models while being ignorant of the sequential nature of the data. One of the main arguments for CNNs is that even (the simpler) architectures that include a sequentiality notion into their derivation, rarely consider more than 3- or potentially 5-grams[17] as input. While figure 2 gives a concrete example of a small network[18] for an NLP task, figure 3 highlights the essential operations of a convolutional network in general.

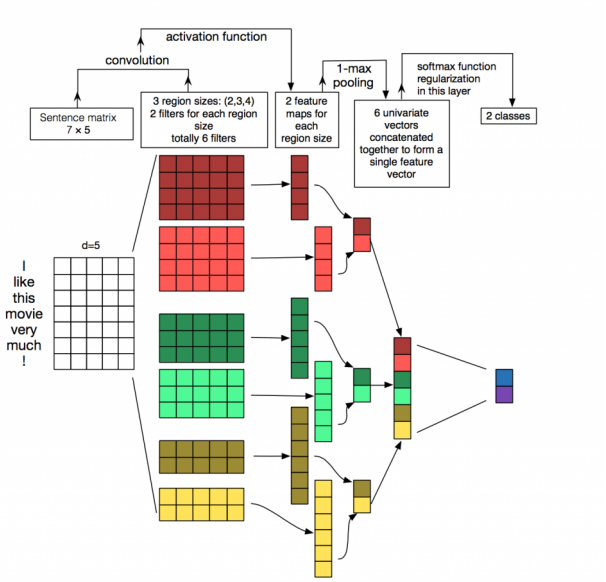


Figure 2: Source: Zhang and Wallace [77]. A simple demonstration of a language classification into two classes via a CNN. The architecture includes a convolutional layer, feeding into a size 1 max-pool layer and a softmax activation function of a dense layer into two target classes.

2.3.2 Recurrent neural networks

The overall structure of the neural network architecture needs to be adapted to allow for the processing of sequential information. The individual units will not only derive results from the current input but also from previous inputs, which leads to a recurrent structure, most prominently coined as *Recurrent Neural Networks* (from here on *RNN*[19]).

While a CNN understands patterns over space, RNNs learn patterns over time. Each unit within an RNN passes on information to its successor, which allows for a more natural interpretation and analysis of especially text data[20]. While RNNs capture the critical aspect of sequences within the input data, this characteristic of the text might just be essential to specific tasks in NLP. For a text classification task, while the interchangeability of

12 Word representation via a high-dimensional (usually in the range of 50 to 300) feature vector, in which the vector length and position carry actual semantic meaning. Prominent pre-trained models are offered by Google's word2vec (as detailed in Mikolov et al. [45]) and Stanford's Glove (published in Pennington et al. [50]).

13 Transfer Learning means that general NN models are pre-trained and more specific environments use the resulting model weights as an input, which then, in turn, fine-tunes it for a specific task.

14 Techniques compared are *distributional representations* (Yarlett and Ramscar [75]), *Brown Clustering* (Brown et al. [10]), *Collobert and Weston embeddings* (Collobert et al. [16]), and *HLBL embeddings* (Mnih and Hinton [47]).

15 First explained in the seminal paper LeCun et al. [41].

16 Architectures that do not form a cycle.

17 N-grams in general, or uni-grams or bi-grams for one and two units, respectively, refer in computational linguistics to the size of the section of n items from a given sample or speech. In this case, it describes the number of words, fed as input to the classification algorithm.

18 Built off an input token, a convolutional layer, a pooling layer and a fully-connected layer for classification.

19 The first one-dimensional RNN structures are explained in [19]

20 "The car hit the man." and "The man hit the car." allow for distinctly different semantic interpretations.

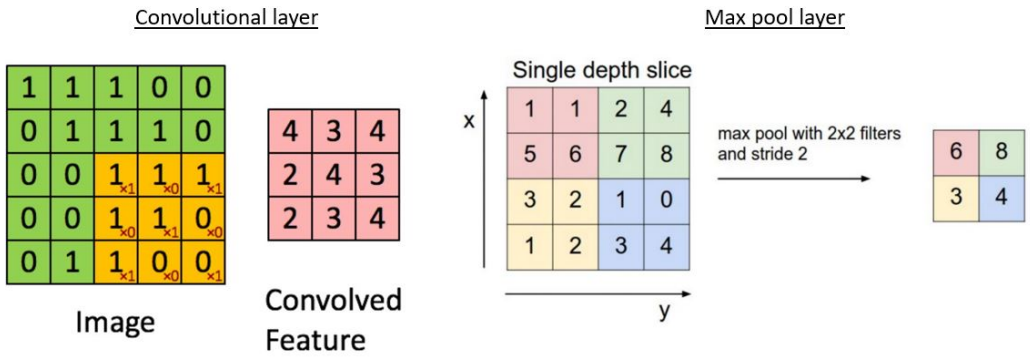


Figure 3: Source: Understanding Convolutional Neural Networks for NLP. Illustration of a typical sequence of convolutional layer and max pool layer, which are often used in sequence to reduce the parameter dimensions, while increasing the feature size of the accompanying weight vector.

words might distort the overall sentiment in certain edge cases, the sentence: "I was very disappointed with the service" might still be interpretable if it reads "very service. I disappointed was with the". Computationally, recurrent networks are more expensive than CNNs, which underlines the task-specificity of the underlying architectures.

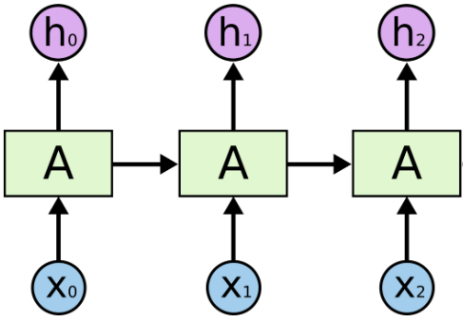


Figure 4: Source: Understanding LSTM Networks. Illustrating the sequential nature of a recurrent neural network. Here X stands for the input at the various stages, h for the output and A for the calculation of the weights, for example via a tanh activation function.

Even though RNNs recognize sequential data, conventional optimization techniques, such as most classically *SGD*[21], often suffer from a so-called *vanishing-gradient*[22] problems to detect long-term relationships. The following architectures offer solution proposals to this complication.

21 Stochastic Gradient Descent, which extends the classical paper of Cauchy [11], an optimization algorithm (introduced through Robbins and Monro [54], Kiefer et al. [33]), which updates the training parameters stochastically in the *direction* of the gradient of the objective function with a step-size of the learning rate α.

22 Refers to the phenomenon that through repeated derivative-calculation as done during the training of a NN for long input sequences, the impact of one unit to subsequent units not immediately following, becomes mathematically minuscule.

2.3.3 Gated recurrent units

To address the vanishing gradient problem, *Gated Recurrent Units* (from here on *GRU*) introduce update and reset gates to the RNN architecture. It allows preserving information from "long-ago" by selectively deciding which information to pass on. Chung et al. [14] first introduces these mechanics.

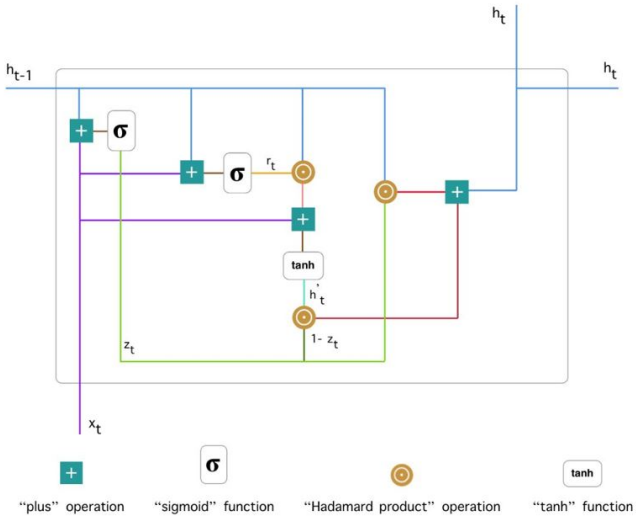


Figure 5: Source: Understanding GRU Networks. Illustrating the individual operations of the GRU unit including the update and reset gate, as well as the overall update function.

The **update gate** follows the following formula and is depicted by a green line in figure 5.

$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$

It helps the model to determine how much of the past information (from previous time steps) is necessary for the future. This model proves especially powerful for vanishing gradients since it can decide to copy all the data from the past.

A **reset gate** plays the antagonist to the update gate, deciding how much of past information to forget. Figure 5 illustrates the general formula in blue and purple.

$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$

By multiplying the input with the weight functions for the current period and previous information with a different weight matrix, we can find the desired mapping[23] from the reset gate to present information. As usual, a non-linear (traditionally the tanh activation) function transforms the output to an adequate output-space.

In a final step, the update gate needs to apply its information to the current memory gate. It connects all the data from previous actions. Through this recurrent application and filtering of relevant information, the network addresses the vanishing gradient problem.

2.3.4 Long-short term memory models

An extension to the GRU cell is found in a more complex updating process through *Long-short term memory cells* (from here on *LSTM*). Hochreiter and Schmidhuber [28] first describe the process, which is governed through four gates, namely the **cell state**, the **forget gate**, the **update gate**, and the **output gate**.

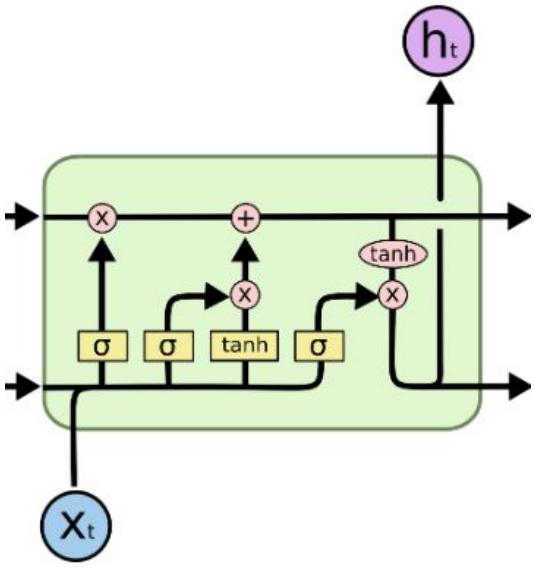


Figure 6: Source: Understanding LSTMs. Illustrating the individual operations of the LSTM cell including the conveyor belt on top, which represents the cell state, and the forget, update and final output functions at the bottom.

The following equations describe the general computations for the LSTM cell:

$$\begin{aligned}
\tilde{\boldsymbol{c}}_t &= \tanh\left(W_c\left[a_{t-1}, x_t\right] + b_c\right) \\
G_u &= \sigma\left(W_u\left[a_{t-1}, x_t\right] + b_u\right) \\
G_f &= \sigma\left(W_f\left[a_{t-1}, x_t\right] + b_f\right) \\
G_o &= \sigma\left(W_o\left[a_{t-1}, x_t\right] + b_o\right) \\
c_t &= G_u * \tilde{c}_t + G_f * c_{t-1} \\
a_t &= G_o * \tanh\left(c_t\right)
\end{aligned}$$

Without too much detail, the overall intuition is that forgetting G_f and updating G_u operations update the cell state $\tilde{\boldsymbol{c}}_t$.

LSTMs have a separate update gate and forget gate. This differentiation makes LSTMs more sophisticated than GRUs but, simultaneously, more complex as well. LSTMs require distinct memory units, which allow them, in theory, to remember longer sequences. GRUs train faster due to only two separate gate-operations and might, therefore, outperform, LSTMs for smaller training data sets. Chung et al. [14], Yin et al. [76], Kaiser and Sutskever [32] offer a comprehensive starting point for comparative analysis.

Many extensions to LSTM cells are available. Tang et al. [67] provides a particularly interesting one in dealing with target-dependent sentiment analysis. Most architectures only allowed for a single target classification within an input sequence.

This lack of differentiation results in the fact that for a given sentence, such as "The weather today was nasty, but the dog still behaved great.", the sentiment given a classical LSTM classifier will be identical for the targets *weather* and *dog*.

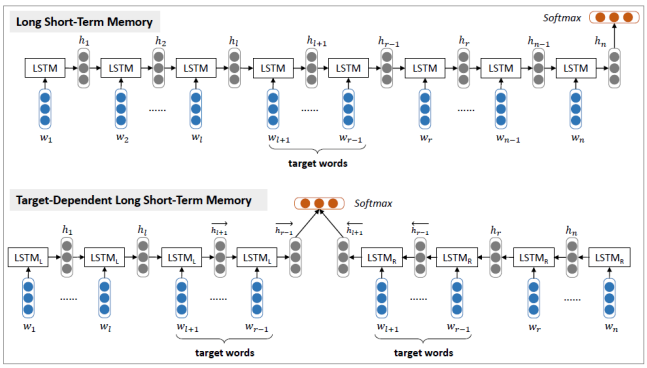


Figure 7: Illustration by Tang et al. [66] demonstrating their TD-LSTM architecture and comparing it to the regular LSTM architecture. The classification through a softmax layer is fed the input of multiple targets, which allows for a more contextual interpretation of the results.

This idea is captured by the top part of Figure 7, which depicts the forward-passing nature of the memory cells. More specifically, in the graphic, a low-dimensional feature representation represents each word, namely an embedding (see Bengio et al. [6], Le and Mikolov [40], Pen-

23 A *Hadamard* (element-wise) multiplication performs this updating between the reset gate input and the new.

nington et al. [50]) and stacks it into an embedding matrix. The embedding training is used from Google's glove learning algorithm, as described in Pennington et al. [50]. The general structure follows the logic of an RNN but allows for long-term dependencies through its four-fold structure of inputting, forgetting, updating, and outputting of specific te LSTM gates.

The results of this LSTM structure maps to a fully connected linear layer, which passes on its output through a softmax layer into a class probability. The reason this framework requires the updated LSTM characteristics to allow for target-dependent analysis stems from the vanishing or exploding gradient problem of deep RNNs, which Hochreiter [27] explains in detail as early as 1998.

2.3.5 Bidirectional models

Bi-directional models allow an extension to all primary **sequential** architectures detailed so far. Schuster and Paliwal [58] introduce the idea that general RNNs can be split into two directions, namely a *forward-* and a *backward-state*. This training stage can be particularly useful for prediction tasks that require to know the next word of a text sequence. It mimics the human peeking forward to understand underlying mechanics better. The forward- and backward stages are not intrinsically connected, to avoid perfect prediction, due to knowing what the next word will be.

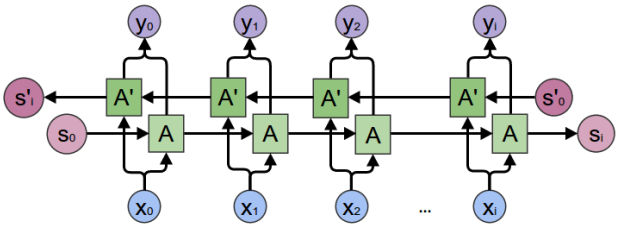


Figure 8: Source: Understanding Bidirectional RNN in PyTorch. The unidirectional RNN of section 2.3.2 is extended and split into two RNNs, one fed the forward the other the reverse sequence. Eventually, both information are then concatenated or summed up at every time step.

Graves and Schmidhuber [22] allows an extensive comparison of uni-directional (regular) LSTM architectures versus their bidirectional counterpart, in which they observe more accuracy for bidirectional structures.

2.3.6 Attention models

Vaswani et al. [72]'s paper proposes an alternative to the predominant, and often convoluted architectures, which base their feature extraction on the interplay between an encoder and a decoder through a *attention* mechanism. The authors introduce a novel single-model state-of-the-art approach that outperforms existing frameworks on popular translation benchmarks while reducing the computational cost to a fraction of competing architectures.

Historically, the architecture consists of an interplay between encoder transforming the input of symbolic representation to a continuous representation and the decoder, autoregressively mapping the continuous format to an output sequence. An attention mechanism, which forms a weighted sum over the output with the help of key-value pair filter, fine-tunes the algorithm. Since this function grows fast in complexity and experiences problems for establishing long-term dependencies, the authors introduce a *scaled-dot-product attention* term, which summarizes the query through a dot-product operation and then applies a softmax function to obtain the respective weights for the attention operation. This calculation further parallelizes and, therefore, also calibrates linear projections of the attention mechanism and concatenation of the results.

Through this application, the authors improved the best results on the WMT[24] English-German and the WMT English-French data-set while only training for 3.5 days.

2.3.7 Bidirectional Encoder Representations from Transformers (BERT)

Devlin et al. [17] proposes s a new language representation model, coined *Bidirectional Encoder Representations from Transformers* (from here on *BERT*) to allow for easy transfer learning[25] incorporation for specific sub-tasks.

Even though the advent of utilizing pre-trained models is common in adjacent cases, most suffer from the unidirectionality of the models (see Radford et al. [52], Vaswani et al. [72]). The paper uses, at its core, the inspiration of an old approach by Taylor [68], which randomly masks single words during pretraining and teaches the network to predict the shaded words (so the model can not *peak* at the words it is supposed to either predict or be feed subsequently). The training follows a two-stage approach; first, the *pre-training* stage, in which a large corpus of unlabeled data trains the architecture, over different pretraining tasks.

Second, a model is initialized during the *fine-tuning* stage, with the previous final model parameters and fine-tuned with labeled data from the downstream tasks. The authors adopt Vaswani et al. [72]'s model for the pre-training stage, unlike Radford et al. [52], Peters et al. [51], a bi-directional framework (as introduced by Graves and Schmidhuber [22]) replaces the left-to-right models, which becomes critical for the performance of the entire endeavor. Figure 9 provides an intuitive insight into the masking of the input tokens, as well as the classification stage of the process.

24 Bojar et al. [9].

25 Transfer Learning means that general NN models are pre-trained and more specific environments use the resulting model weights as an input, which then, in turn, fine-tunes it for a specific task.

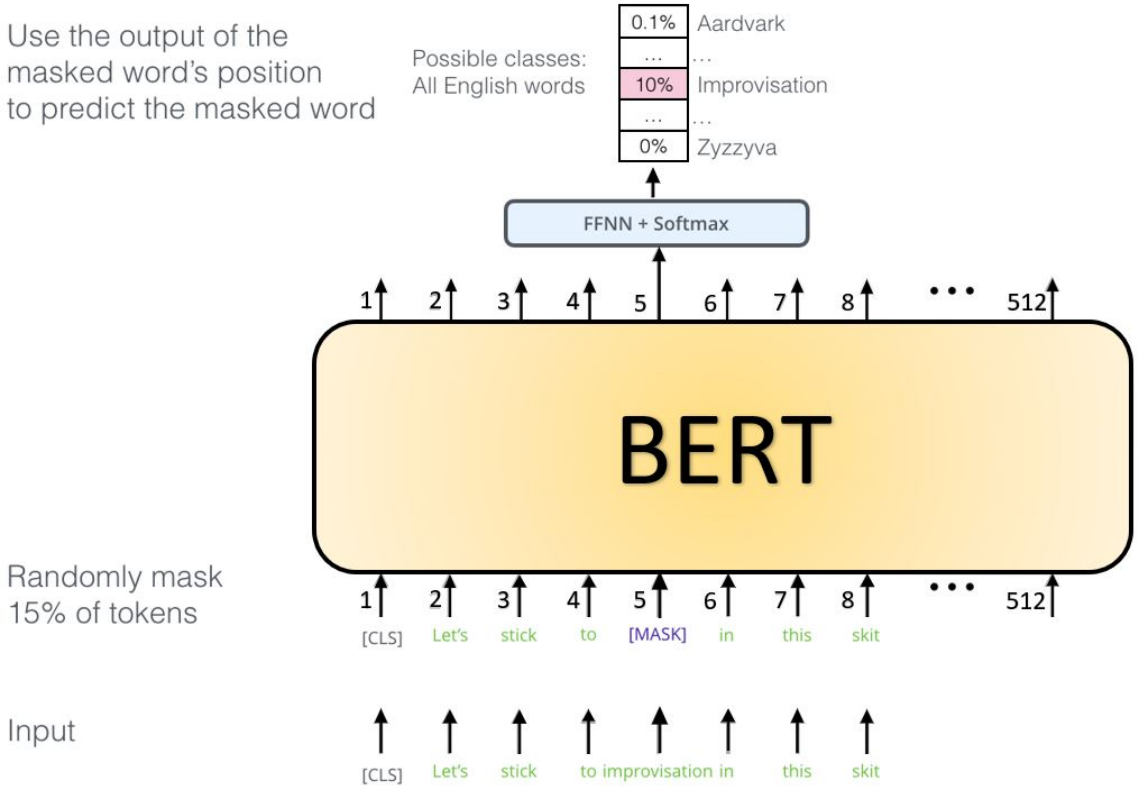


Figure 9: Source: The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning). Illustration of the BERT mechanism to utilize an encoder-decoder interaction while incorporating a bi-directional structure via masking for embedding classification tasks.

Next sentence prediction and *question answering* rely heavily on Taylor [68]'s masking method. For the fine-tuning stage, the relevant tasks require swapping the input data and minor layers and re-train the network. BERT represents an extension to the ELMo network Peters et al. [51], which introduced the notion of context into word embeddings[26], achieved through a Bi-directional LSTM training phase.

3 Research design

This section details the research design of the study. It explains the data source, the framework of the loaded data, and will build intuition on the data through visual data explorations in subsection 3.1. It will then explain model comparison and chosen hyper-parameters for a significant comparative structure in subsection 3.2. The code, a short Readme file and the data can be found under: https://github.com/marcluettecke/single-sentence-classification-NN.

3.1 Data

The dataset contains about 20k single sentences of one of three authors: Edgar Allen Poe[27], H.P. Lovecraft[28] and Mary Shelley[29]. The endeavor stems from the 2017 Halloween challenge posted on Kaggle, which describes the dataset as follows:

> *"The competition dataset contains text from works of fiction written by spooky authors of the public domain: Edgar Allan Poe, HP Lovecraft, and Mary Shelley. The data was prepared by chunking larger texts into sentences using CoreNLP's MaxEnt sentence tokenizer, so you may notice the odd non-sentence here and there. Your objective is to accurately identify the author of the sentences in the test set."*

Since the test data set does not contain any labels, the training data set will build the entirety of our data (which separates into training, validation, and testing sub-datasets, detailed in subsection 3.2).

26 The same word might have different semantic relations to other words, depending on the setting.

27 January 19, 1809 – October 7, 1849 - American writer most prominently known for poetry and short stories, such as *The Raven*.

28 August 20, 1890 – March 15, 1937 - American writer of horror fiction and creator of the *Cthulhu Mythos*.

29 August 30, 1797 – February 1, 1851 - English novelist, creator of *Frankenstein; or, The Modern Prometheus*.

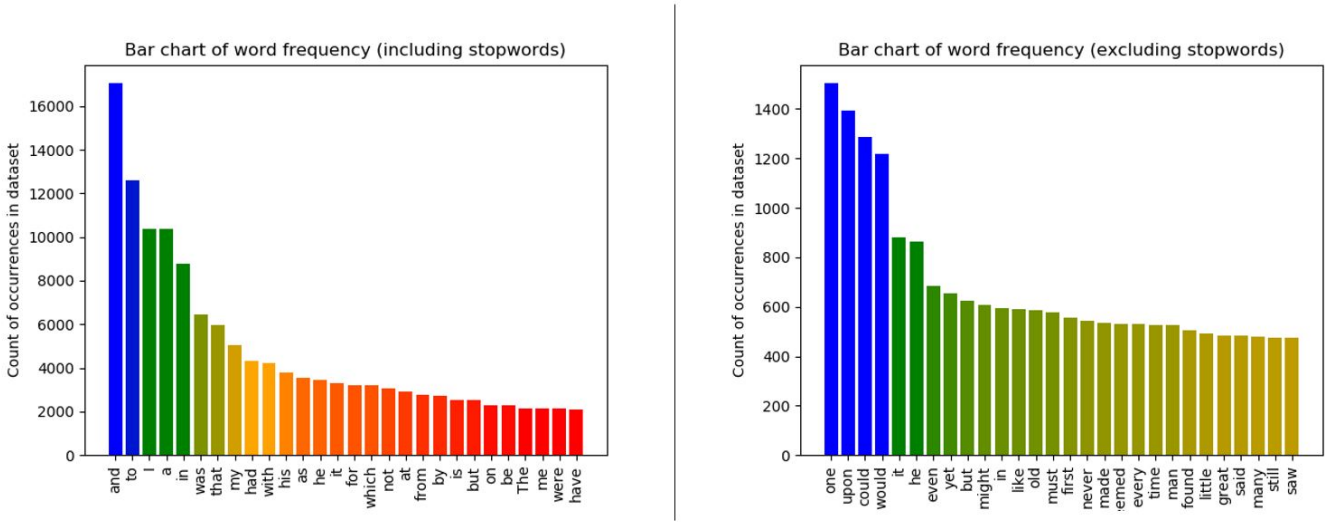


Figure 10: Comparing the 30 most frequent words in the entire data set, (left) including stop words and (right) with stop words removed. The Figure shows how most of the almost tokens on the left are common building-blocks of the English language without identification value. They are removed and the most frequent words for the cleaned data carry more interpretative value towards an individual classification.

For data pre-processing, shortly detailed in the technical aside on word embeddings (see subsection 2.2), the text is cleaned of stop words through the individual tokens for author classification. Figure 10 aims to explicitly compare the word frequencies with and without the most common English stop words included [30].

To dive into the individual data-sub-sets figure 11 highlights the overall counts of sentences in the dataset, split by the authors. Even though Edgar Allen Poe (EAP) has the most training labels, the difference has no impact on the results (investigated of error sources for distinct misclassification) over many epochs and shuffled feeding of the data into the networks via small batches.

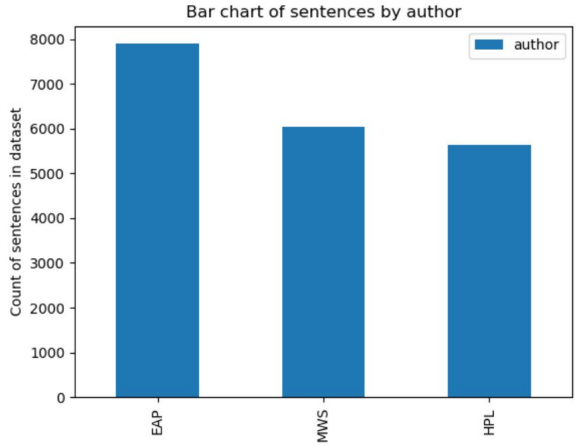


Figure 11: Overview of the relative count of sentences by author in the dataset shows a relatively balanced distribution between the three labels.

Figure 12 provides more anecdotal intuition for the most frequent words per author, as well as their distinct vocabulary, which is part of the underlying data structure for the text classification.

3.2 Model comparison

For the actual model comparison, the study follows these successive steps.

3.2.1 Research objectives

1. **Compare custom embeddings and pre-loaded embeddings** - during this phase, a simple CNN model is trained for a glove[31] and a custom word-embedding[32] based on our own text corpus. Due to computational capacity limits and the exponential increase in models to compare, the analysis discusses one embedding method onward, which is used for the entire rest of the comparative analysis.

2. **Build a linear baseline classifier as a baseline comparison** - A simple Logistic Regression model estimates the author attribution as an easy comparison. Even though the underlying structure of the text might not be understood by a strictly linear classifier (which does not allow for transfer learning or generalization of the results), the classifier is easily built and the computational complexity is a fraction of the efforts for a neural net.

30 As defined by the popular *Natural Toolkit Library* library for NLP tasks.

31 Stanfords model published in Pennington et al. [50], which is trained on 6 Billion tokens from Wikipedia.

32 Using Googles Word2Vec method.

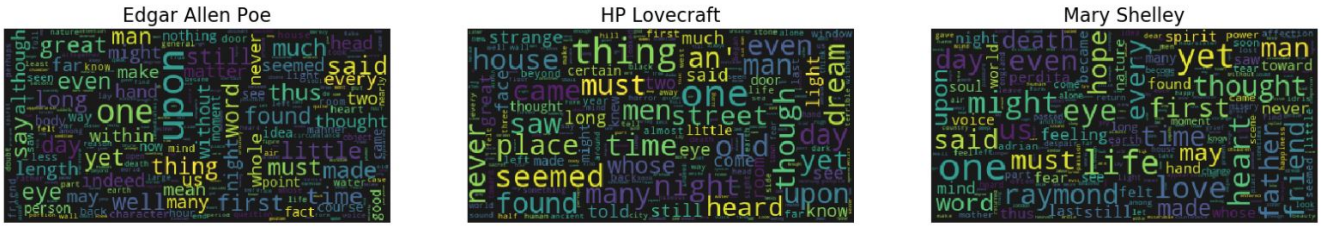


Figure 12: Wordcloud visualizations for the three authors of the labeled dataset. Interesting are some examples for descriptive tokens for the authors, such as *"raymond"*, as in *Lord Raymond*, a main character in Mary Shelley's *The Last Man*. Another example is the word *"thing"* for HP Lovecraft, which is often used to describe his *Cthulhu*-creation.

3. **Build a model comparison of increasing complexity**[33] - The following models are trained[34] and evaluated on a test data set:
   (a) Logistic regression model
   (b) Simple embedding layer
   (c) Embedding + CNN
   (d) Embedding + RNN
   (e) Embedding + BD[35]-RNN
   (f) Embedding + GRU
   (g) Embedding + BD-GRU
   (h) Embedding + LSTM
   (i) Embedding + BD-LSTM
   (j) Embedding + Attention
   (k) BERT
4. **Train the models with a trainable embedding layer** to understand how good of a classification accuracy can be achieved in a more realistic setting[36]. This results in the models (a)-(k) repeated with the training-parameter of the Embedding layer[37] set to *True*.
5. **Communicate the results via visualizations and accuracy metrics** to acquire a deeper understanding of how useful neural networks of increasing complexity prove for text classification tasks of minimal input.

3.2.2 Hyper-parameters

The following groups of hyper-parameters define the training process. Necessary details for the technical background, not covered in section 2, are listed in footnotes with numerous citations for further details.

1. **Word-specific parameters:**
   (a) Sentence max-length (tokens): 100
   (b) Train-size: 0.85
   (c) Test-size: 0.15
   (d) Embedding-number of words: 10,000
   (e) Embedding vector dimensions: 300
2. **Architecture parameters:**
   (f) Filter size of convolutional Layers: 32
   (g) Drop-out portion of drop-out layers: 0.2
   (h) Recurrent drop-out of recurrent layers: 0.2[38]
3. **Training parameters:**
   (i) Optimization algorithm: *Adam*[39]
   (j) Loss function: *Sparse-categorical cross-entropy loss*[40]
   (k) Epochs: 100[41]
   (l) Batch-size: 256
   (m) Early stopping patience: 50
   (n) Early stopping parameter: Validation loss

33 It is important to notice that the model includes the dense classification layer plus minimal pooling layers, even if not specifically mentioned.
34 The Python Deep Learning library *Keras* building on a *Tensorflow* backend is used to initiate, compile, train and evaluate all models.
35 Bi-directional networks, for technical details see section 2.3.5
36 The BERT model is trained under the Keras ktrain interface, which allows for much less customization and also lacks a simple visualization implementation. Therefore, for the BERT model only training and test accuracy scores will be listed.
37 See the full documentation for more details on the layer options here.
38 Problems of using regular drop-out in recurrent layers is discussed in depth in Bayer et al. [3].
39 Adam represents an adjustable loss function, which unlike SGD includes features of *Momentum* (originally discussed in Sutskever et al. [63]), *RMSprob* (extensively discussed in Ruder [55]) and the stochastic nature of SGD. It was first introduced in Kingma and Ba [36].
40 Which is also called Softmax Loss and combines a Softmax activation with a Cross-Entropy loss. The output covers probabilities over the vector of labels for a sentence and is therefore common in multi-class classification tasks.
41 Due to computational limitations, the BERT model was only trained for **4** epochs.

The training phase offered abundant opportunities to fine-tune the hyper-parameters of the numerous models. The resulting values have proven to result in the best trade-off between computational complexity and classification accuracy.

4 Results

This section details the results of the study. It explains the outcomes, along with the research goals mentioned in section 3.2.1. The code to reach the same results is listed under this GitHub repository.

1. **Embedding comparison:** Since computational rigor was a limiting factor, only the CNN model trains on both a custom embedding and the glove embedding (Pennington et al. [50]). For various dimensions (50, 100, 200, 300) of the feature vector, the embeddings did not impact the test accuracy and test loss results. Table 1 lists these metrics for CNN. For the subsequent model specifications, the models train with the custom embedding, since it allows for more potential to highlight discrepancies between including the embedding layer into the model and not.

2. **Linear classifier as baseline:** To understand how easy or hard the task at hand is, a logistic classifier is implemented and achieves an accuracy of 80%. The F1 score[42], precision[43] and recall[44] are on par with the accuracy measures (see table 1). Table 3 highlights the true positives (correct labeling) and the false predictions for each class.

3. **Model comparison without training embedding:** This result lies at the heart of the study. This goal, as well as the comparison with trainable embedding models, contrasts the performance of neural networks, which are not explicitly tweaked for text classification, for single sentence labeling on a small dataset. Table 1[45] details all accuracy measures and observed classification metrics. Most notably, none of the models reach the benchmark of a simple multiclass logistic classifier highlighted in table 3.

		Predicted label			
		HPL	**EAP**	**MWS**	**Total**
Actual label	**HPL**	656	137	71	864
	EAP	65	1026	109	1200
	MWS	36	149	687	872
	Total	757	1312	867	2936

Table 3: The confusion matrix for the Logisti Regression result shows the strong accuracy and the balanced classification for the different authors. The diagonal indicates the true positives in green and the off-diagonal numbers highlight either the true negatives or false positives (which is a deceiving specification for a multi-label case) in red.

The second highest test accuracy provides either the multiple dense layers of the baseline model or the CNN model, both with just below 50% accuracy. This outcome is higher than chance (which would be 33% in a three classes case) but is not an excellent performance by any means. Figure 13 offers some explanation to the poor performance: While the Baseline model in subfigure 13a and CNN model in 13b both follow smooth learning patterns (with maybe a slight tendency to overfitting[46]), some models show partially strong tendencies of underfitting[47] (see GRU in subfigure 13e) or overfitting (see BD-GRU in subfigure 13f). The more complex models in figure 14 seem to converge more smoothly and learn more adequately. Without any problems of over- or underfitting. The smooth curve of the more complex classification algorithms demonstrates that data as high-dimensional as sentence-tokens might need more exhaustive model-representations to be generalized.

4. Generally, the performance of the models, including the embedding training, is much more accurate than without it. Accuracies, which were just better than random guessing, now include the fine-tuning of the embedding representation and allow for results as high as 75 - 80%. The benchmark of the simple logistic regression still represents a high burden, but the BERT model (not highlighted in the figures) was able to exceed it with 88% accuracy on

42 A combined score for traditionally binary classification, extends to multi-class labeling (by definition of a weighting algorithm). The formula is: $2 \times \frac{precision \times recall}{precision + recall}$.

43 Measure for the true positive rate over the predicted positives in a classification.

44 Measure for the true negative rate over the predicted negatives.

45 Notice that different y-axis values indicate an *Early Stopping* option by Keras, which allows abandoning the training process, if a particular evaluation metric, for us the validation loss, did not improve over a predefined span of epochs.

46 Refers to the problem in machine- and deep-learning that the model learned patterns from the training data well, but suffers from lack of generalization. It performs poorly on unseen data, here the test data set and just follows the patterns of the training data, without the inclusion of more general concepts.

47 The antagonist to *overfitting*, which indicated problems of the model to grasp the complexity of the data. A higher training than testing loss hint at a potential underfitting.

	Training data		Testing data				
	Accuracy	**Loss**	**Accuracy**	**Loss**	**F1-score**	**Precision**	**Recall**
Logit	-	-	**0.8068**	0.4949	0.8067	0.8099	0.8068
Baseline	0.4722	1.024	0.4785	1.0219	0.4500	0.4800	0.4800
CNN	0.5253	0.9776	0.4989	0.9975	0.4800	0.5000	0.4900
RNN	0.4147	1.0697	0.4090	1.2500	0.4100	0.2900	0.3600
BD-RNN	0.4735	1.0208	0.4516	1.0432	0.4700	0.4700	0.4700
GRU	0.4030	1.0851	0.4087	1.0859	0.2400	0.1700	0.4100
BD-GRU	0.5439	0.9427	0.5177	0.9764	0.5000	0.5200	0.5200
LSTM	0.4982	0.98828	0.4843	1.0061	0.4700	0.4800	0.4900
BD-LSTM	0.5444	0.9386	0.5119	0.9797	0.5100	0.5100	0.5100
Attention	0.5488	0.9253	0.5350	0.9636	0.5200	0.5400	0.5400
BERT	-	-	-	-	-	-	-

Table 1: This table shows the results of the model evaluation for the training **without** additional training of the Embedding layer (other than using a custom embedding which was trained on the text corpus in the first place). We can see that model complexity does not correlate with test-accuracy and that the embedding training in itself accounts for a high amount of accuracy.

	Training data		Testing data				
	Accuracy	**Loss**	**Accuracy**	**Loss**	**F1-score**	**Precision**	**Recall**
Logit	-	-	0.8068	0.4949	0.8067	0.8099	0.8068
Baseline	0.9940	0.0597	0.8273	0.4543	0.8300	0.8300	0.8300
CNN	0.9996	0.0018	**0.7840**	1.8428	0.8100	0.8100	0.8100
RNN	0.4030	1.0869	0.4087	1.0869	0.2400	0.1700	0.4100
BD-RNN	0.9975	0.0099	**0.7758**	1.078	0.8000	0.8000	0.8000
GRU	0.6088	0.9333	0.5149	0.9983	0.4500	0.4700	0.5100
BD-GRU	0.99807	0.0096	0.7704	1.8309	0.8100	0.8100	0.8100
LSTM	0.9976	0.0054	0.7690	1.4893	0.7700	0.7700	0.7700
BD-LSTM	0.9353	0.1947	**0.7905**	0.5745	0.7900	0.8000	0.7900
Attention	0.9992	0.0022	0.7700	2.2913	0.7700	0.7700	0.7700
BERT	-	-	**0.8800**	-	0.8800	0.8800	0.8800

Table 2: This table shows the results of the model evaluation for the training **including** training of the Embedding layer. Even simple models increase the accuracy and exceed any models that were trained without the Embedding layer in table 1.

the test set. Other models are closer to the 80% as well, with the best runner-up performance (82.73%) in the dense-baseline model of just connecting the embedding layer to subsequent dense layers. For the more complex candidates, the BD-LSTM model achieves 79.05% accuracy, and the Attention model 77%. All models suffer from strong overfitting problems, which makes intuitive sense since these models increase their model complexity without acquiring more data or additional regularization methods[48] to balance it. An interesting extension would be to train a complex model, with additional regularization to benefit from the accurate feature-extraction without paying the cost of limited data access.

48 If the models had been tweaked more between the first round (without embedding training) and the second round (including embedding training) the comparability had suffered from too many dimensions changed at once.

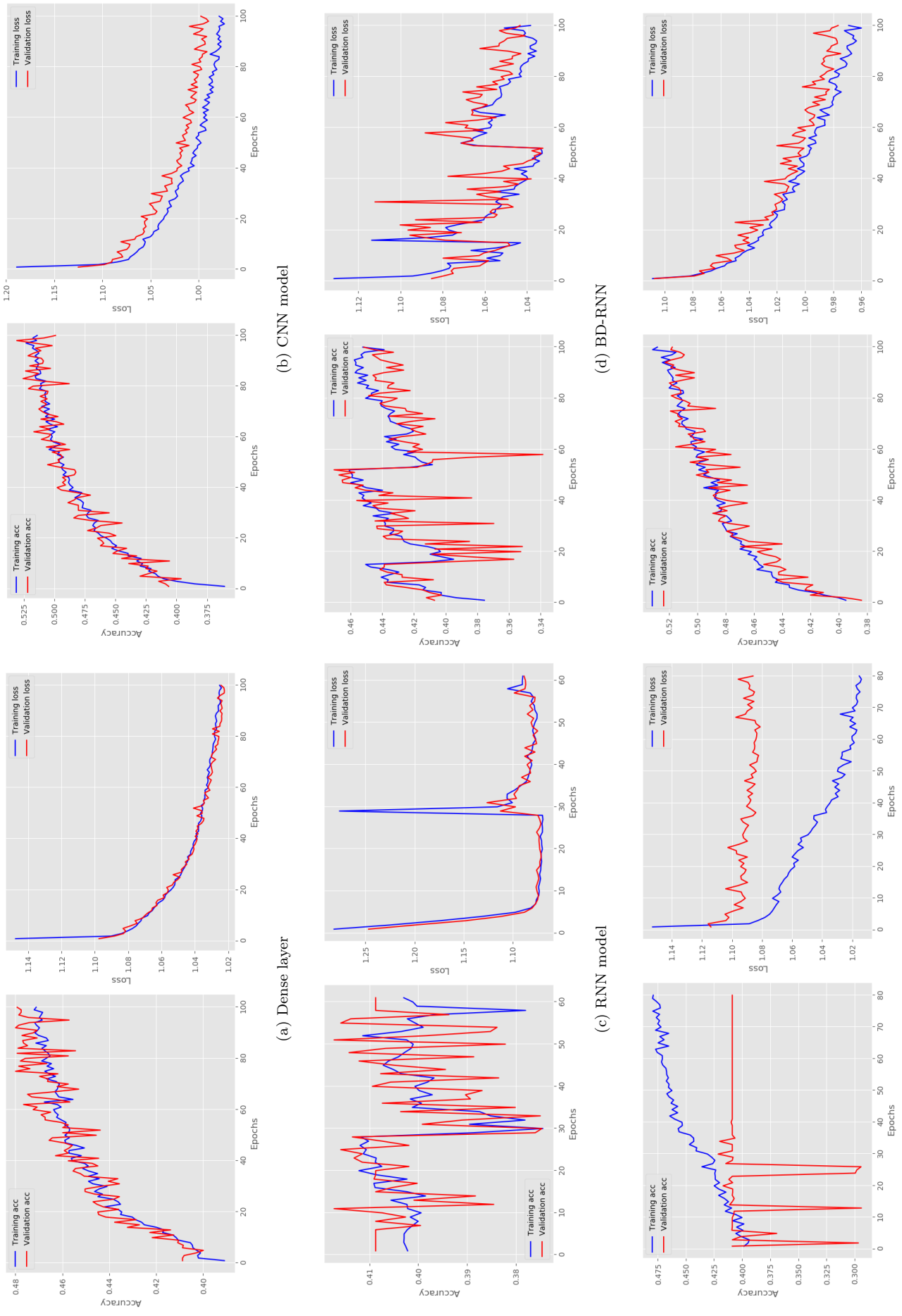
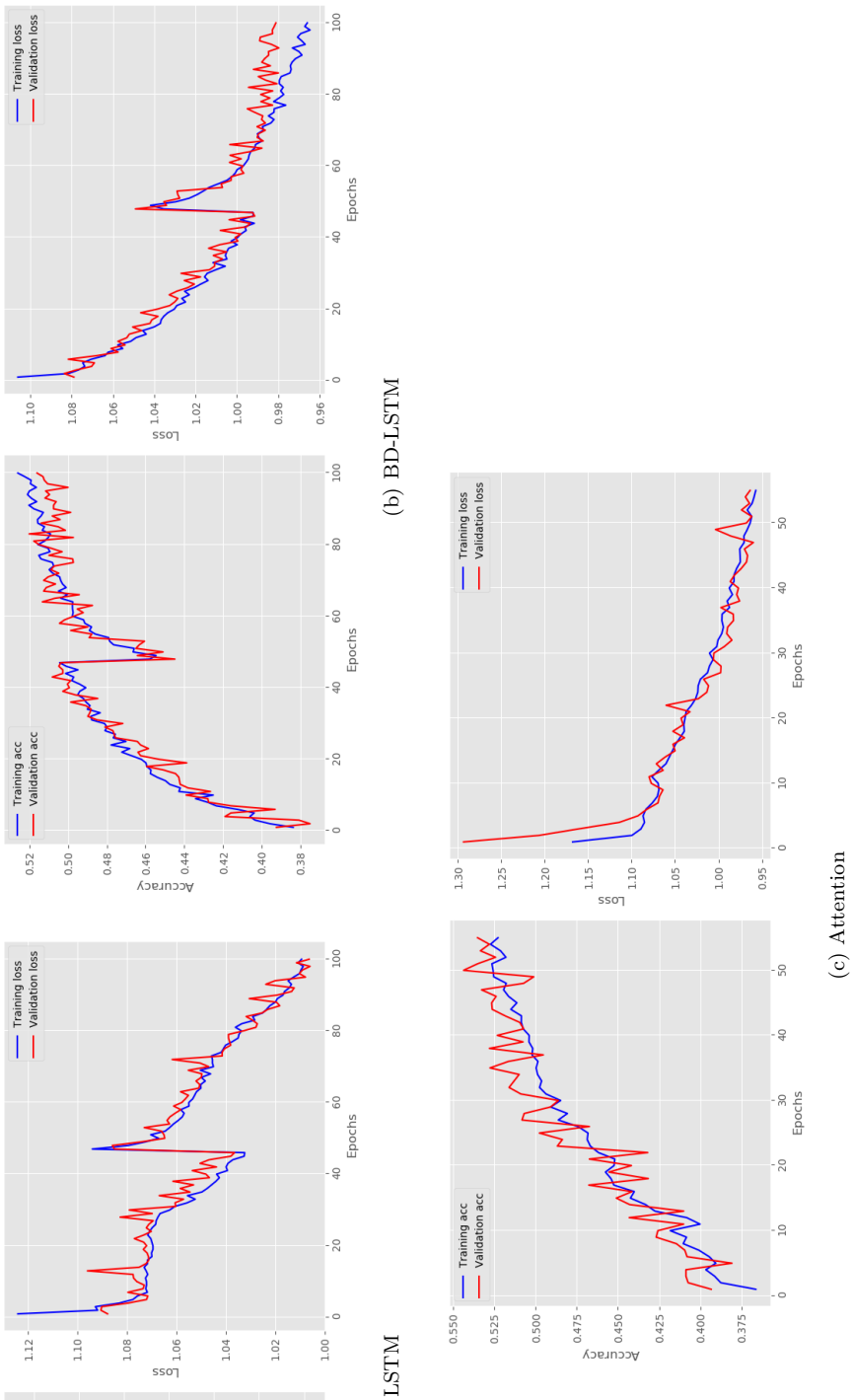
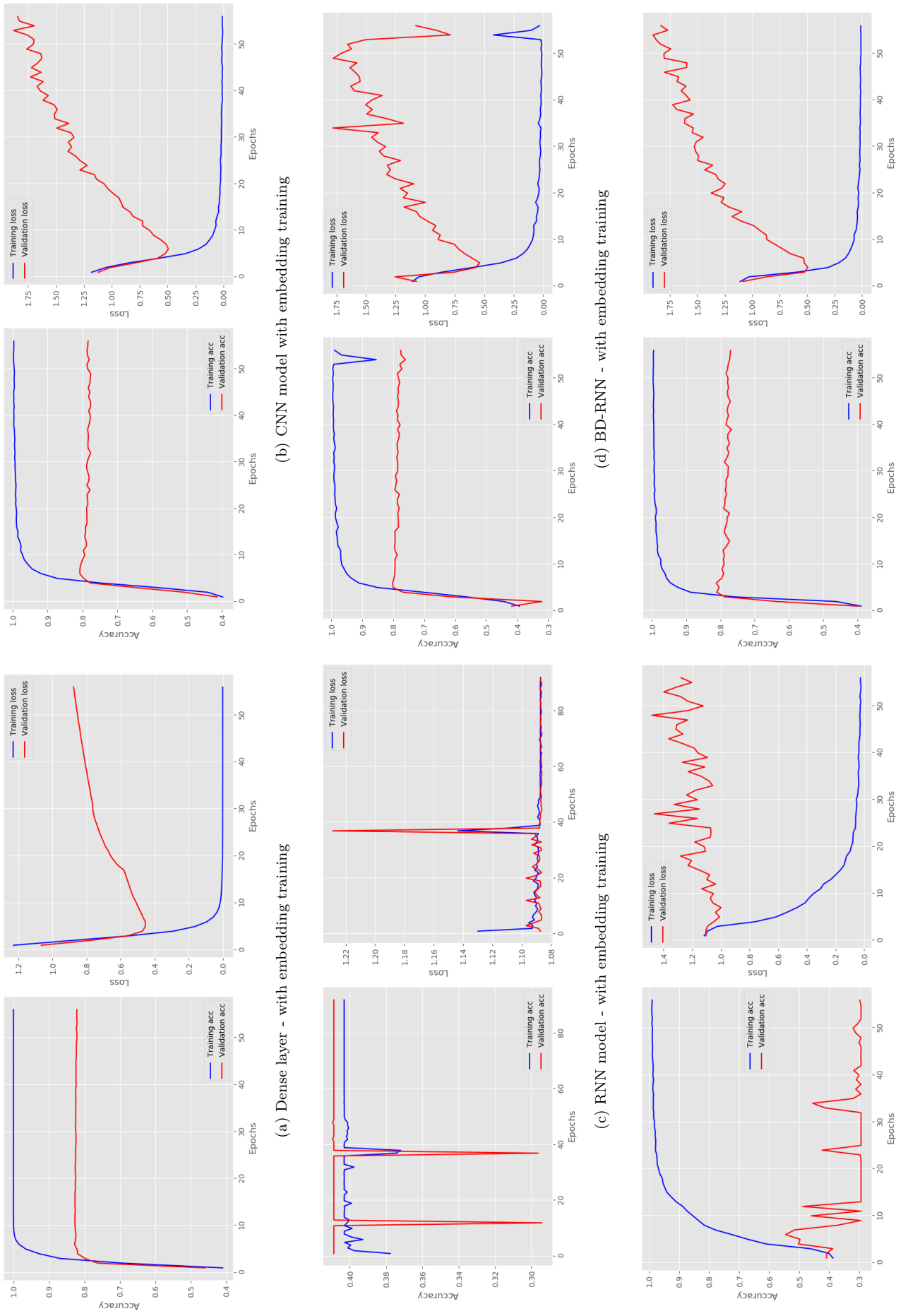


Figure 13: A comparison of the first 6 (with a tendency to be simpler than the upcoming) models **without** training the embedding layer specifically. The performance is poor and, for most models, even worse than just random chance classification. Especially sequential models without methods to retain "long-term" information suffer from overfitting and poor performance on training, as well as test data. The different scaling for the y-axis are due to an Early Stopping mechanism, which stops training if the target-parameters are not changing anymore.



(a) LSTM (b) BD-LSTM (c) Attention

Figure 14: A comparison of the other three models more complex models **without** training the embedding layer specifically. The performance improves, which indicates that increasing model complexity captures the intrinsic structure of the data well. The performance is not impressive with about 50% accuracy on the test data set, but the graphs indicate no overfitting of the training data. The different scaling for the y-axis are due to an Early Stopping mechanism, which stops training if the target-parameters are not changing anymore.



(a) Dense layer - with embedding training

(b) CNN model with embedding training

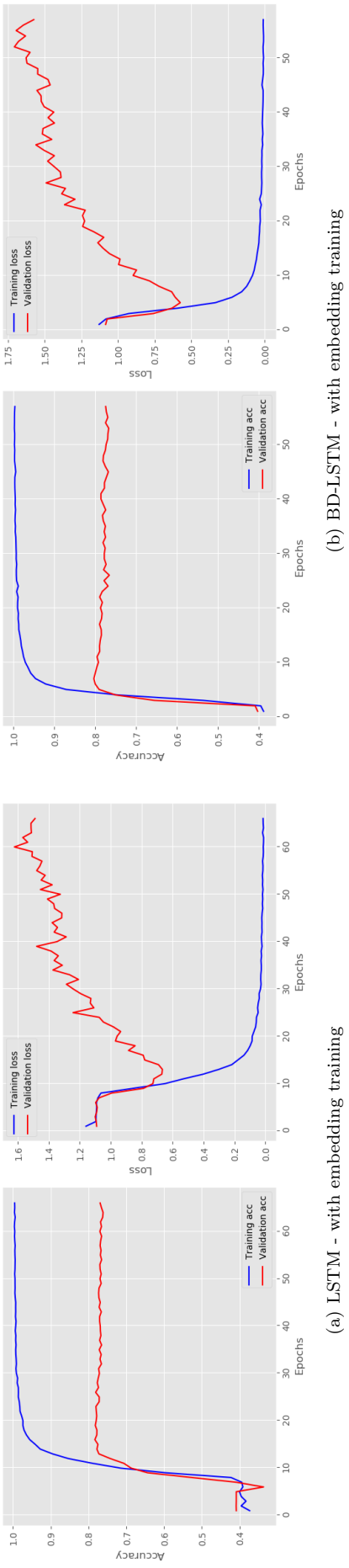
(c) RNN model - with embedding training

(d) BD-RNN - with embedding training

(e) GRU - with embedding training

(f) BD-GRU - with embedding training

Figure 15: The same six simpler models are compared, as in figure 13, but with the inclusion of a training phase for the custom embedding layer. The models perform significantly better, even though the results demonstrate tendencies of overfitting. The different scaling for the y-axis are due to an Early Stopping mechanism, which stops training if the target-parameters are not changing anymore.



(a) LSTM - with embedding training

(b) BD-LSTM - with embedding training

(c) Attention - with embedding training

Figure 16: The most complex models, including training of the embedding layer, allow for robust increases of results. All models suffer from overfitting the training data, which still does not detrimentally decrease their test accuracy performance. The different scaling for the y-axis are due to an Early Stopping mechanism, which stops training if the target-parameters are not changing anymore.

5 Discussion and conclusion

This paper illustrates the accuracy of a classification task for single sentence author labeling for different neural network models. The main driver in performance seems to be the training of a meaningful word-representation, i.e., the embedding layer, or for more sophisticated models, the BERT embedding. While more complex models allow for better feature extraction to represent the high-dimensionality of text as an input vector, they also come at a significant computational cost. Rich architectures, such as Attention, or BERT models require significant effort to train, but consequently, reach state-of-the-art performance. Further research is necessary for a meaningful quantitative model, which balances the trade-off between computational complexity and accuracy. CNNs afford a simple but still well-performing architecture, which does not necessarily convey sequential information well. However, through their convolution operations, they reduce the computational burden while still extracting the main features. Model adaptions, which extend vanilla CNN structures for computer vision, such as ResNet, VGG, or Inception[49] might be a fruitful starting point for further improvements of a text classification model. It might be important to notice that while a single-sentence classification proves to be a consequential example for a common problem in NLP, it does not characteristically represent the more sequential complications of many tasks. The average performance of LSTM cells for this specific simplification for other tasks does not invalidate the importance of sequential cell-models for text interpretation in general.

49 Original seminal work: He et al. [25], Simonyan and Zisserman [60], Szegedy et al. [64], respectively.

References

[1] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853, 2005.

[2] Ismail Badache, Sébastien Fournier, and Adrian-Gabriel Chifu. Contradiction in reviews: is it strong or low? 2018.

[3] Justin Bayer, Christian Osendorfer, Daniela Korhammer, Nutan Chen, Sebastian Urban, and Patrick van der Smagt. On fast dropout and its applicability to recurrent networks. *arXiv preprint arXiv:1311.0701*, 2013.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, mar 1994. doi: 10.1109/72.279181.

[5] Yoshua Bengio. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pages 1–37. Springer, 2013.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[7] Steven Bethard, Hong Yu, Ashley Thornton, Vasileios Hatzivassiloglou, and Dan Jurafsky. Automatic extraction of opinion propositions and their holders. In *2004 AAAI spring symposium on exploring attitude and affect in text*, volume 2224, 2004.

[8] Douglas Biber and Edward Finegan. Adverbial stance types in english. *Discourse processes*, 11(1):1–34, 1988.

[9] Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, 2016.

[10] Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479, 1992.

[11] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[12] Jason PC Chiu and Eric Nichols. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370, 2016.

[13] Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 355–362. Association for Computational Linguistics, 2005.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[15] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[16] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Li Dong, Furu Wei, Chuanqi Tan, Duyu Tang, Ming Zhou, and Ke Xu. Adaptive recursive neural network for target-dependent twitter sentiment classification. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)*, pages 49–54, 2014.

[19] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[20] Steve Engels, Vivek Lakshmanan, and Michelle Craig. Plagiarism detection using feature-based neural networks. In *Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 34–38, 2007.

[21] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. Ppdb: The paraphrase database. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 758–764, 2013.

[22] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[23] Maryam Habibi, Leon Weber, Mariana Neves, David Luis Wiegandt, and Ulf Leser. Deep learning with word embeddings improves biomedical named entity recognition. *Bioinformatics*, 33(14):i37–i48, 2017.

[24] Devamanyu Hazarika, Soujanya Poria, Sruthi Gorantla, Erik Cambria, Roger Zimmermann, and Rada Mihalcea. Cascade: Contextual sarcasm detection in online discussion forums. *arXiv preprint arXiv:1805.06413*, 2018.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[26] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[27] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[28] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[29] Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60, 2006.

[30] Yuheng Hu, Fei Wang, and Subbarao Kambhampati. Listening to the crowd: automated analysis of events via aggregated twitter sentiment. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[31] Long Jiang, Mo Yu, Ming Zhou, Xiaohua Liu, and Tiejun Zhao. Target-dependent twitter sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 151–160. Association for Computational Linguistics, 2011.

[32] Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

[33] Jack Kiefer, Jacob Wolfowitz, et al. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[34] Soo-Min Kim and Eduard Hovy. Identifying and analyzing judgment opinions. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 200–207. Association for Computational Linguistics, 2006.

[35] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[36] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[37] Peter Koomen, Vasin Punyakanok, Dan Roth, and Wen-tau Yih. Generalized inference with multiple semantic role labeling systems. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, pages 181–184. Association for Computational Linguistics, 2005.

[38] Sneha Kudugunta and Emilio Ferrara. Deep neural networks for bot detection. *Information Sciences*, 467: 312–322, 2018.

[39] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[40] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[41] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[42] Luyang Li, Bing Qin, and Ting Liu. Contradiction detection with contradiction-specific word embedding. *Algorithms*, 10(2):59, 2017.

[43] Percy Liang. *Semi-supervised learning for natural language*. PhD thesis, Massachusetts Institute of Technology, 2005.

[44] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

[45] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[46] Scott Miller, Jethran Guinness, and Alex Zamanian. Name tagging with word clusters and discriminative training. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 337–342, 2004.

[47] Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM, 2007.

[48] Franz Josef Och and Hermann Ney. Giza++: Training of statistical translation models, 2000.

[49] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[50] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[51] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[52] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding with unsupervised learning. Technical report, Technical report, OpenAI, 2018.

[53] Frédéric Ratle, Jason Weston, and Matthew L Miller. Large-scale clustering through functional embedding. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 266–281. Springer, 2008.

[54] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[55] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[56] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[57] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[58] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[59] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141. Association for Computational Linguistics, 2003.

[60] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[61] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[62] Yaming Sun, Lei Lin, Duyu Tang, Nan Yang, Zhenzhou Ji, and Xiaolong Wang. Modeling mention, context and entity with neural networks for entity disambiguation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[63] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[65] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, 2014.

[66] Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. Effective lstms for target-dependent sentiment classification. *arXiv preprint arXiv:1512.01100*, 2015.

[67] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[68] Wilson L Taylor. "cloze procedure": A new tool for measuring readability. *Journalism Bulletin*, 30(4):415–433, 1953.

[69] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for computational Linguistics, 2003.

[70] Joseph Turian, Lev Ratinov, Yoshua Bengio, and Dan Roth. A preliminary evaluation of word representations for named-entity recognition. In *NIPS Workshop on Grammar Induction, Representation of Language and Language Learning*, pages 1–8, 2009.

[71] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[72] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[73] Duy-Tin Vo and Yue Zhang. Target-dependent twitter sentiment classification with rich automatic features. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[74] Theresa Wilson and Janyce Wiebe. Annotating opinions in the world press. In *Proceedings of the Fourth SIGdial Workshop of Discourse and Dialogue*, pages 13–22, 2003.

[75] D Yarlett and M Ramscar. Language learning through similarity-based generalization. *Unpublished PhD Thesis, Stanford University*, 2008.

[76] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

[77] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[78] Kazimierz Zielinski, Radoslaw Nielek, Adam Wierzbicki, and Adam Jatowt. Computing controversy: Formal model and algorithms for detecting controversy on wikipedia and in search queries. *Information Processing & Management*, 54(1):14–36, 2018.