Modern Approaches in Natural Language Processing

Contents

Preface

In the last few years, there have been several breakthroughs concerning the methodologies used in Natural Language Processing (NLP). These breakthroughs originate from both new modeling frameworks as well as from improvements in the availability of computational and lexical resources.

In this seminar, we are planning to review these frameworks starting with a methodology that can be seen as the beginning of modern NLP: Word Embeddings.

We will further discuss the integration of embeddings into end-to-end trainable approaches, namely convolutional and recurrent neural networks. As Attention-based models and transfer learning approaches are the foundation of most of the recent state-of-the-art models, we will cover these two topics extensively in the second part of our seminar.

We will furthermore talk about software implementations of these methods and benchmark tasks/data sets for evaluating state-of-the-art models.

This book is the outcome of the seminar "Modern Approaches in Natural Language Processing" which took place in summer 2020 at the Department of Statistics, LMU Munich.

FIGURE 1: Creative Commons License

[1] http://creativecommons.org/licenses/by-nc-sa/4.0/

Foreword

Author: Christoph Molnar

This book is the result of an experiment in university teaching. Each semester, students of the Statistics Master can choose from a selection of seminar topics. Usually, every student in the seminar chooses a scientific paper, gives a talk about the paper and summarizes it in the form of a seminar paper. The supervisors help the students, they listen to the talks, read the seminar papers, grade the work and then … hide the seminar papers away in (digital) drawers. This seemed wasteful to us, given the huge amount of effort the students usually invest in seminars. An idea was born: Why not create a book with a website as the outcome of the seminar? Something that will last at least a few years after the end of the semester. In the summer term 2020, some Statistics Master students signed up for our seminar entitled "Limitations of Interpretable Machine Learning". When they came to the kick-off meeting, they had no idea that they would write a book by the end of the semester.

We were bound by the examination rules for conducting the seminar, but otherwise we could deviate from the traditional format. We deviated in several ways:

1. Each student project is part of a book, and not an isolated seminar paper.
2. We gave challenges to the students, instead of papers. The challenge was to investigate a specific limitation of interpretable machine learning methods.
3. We designed the work to live beyond the seminar.
4. We emphasized collaboration. Students wrote some chapters in teams and reviewed each others texts.

Technical Setup

The book chapters are written in the Markdown language. The simulations, data examples and visualizations were created with R (R Core Team, 2018). To combine R-code and Markdown, we used rmarkdown. The book was compiled

with the bookdown package. We collaborated using git and github. For details, head over to the book's repository[2].

[2] link/to/repo

1

Introduction

Author:

Supervisor:

1.1 Intro About the Seminar Topic

1.2 Outline of the Booklet

2

Chapter 1

Authors: Author 1, Author 2

Supervisor: Supervisor

2.1 Lorem Ipsum

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

R Core Team (2018)

2.2 Using Figures

Referencing can be done by using the chunk label e.g. `\@ref(fig:ch01-figure01)` for 2.1.

NOTE!!! Do not use underscores in chunk labels! This will crash the compilation …

2.3 Using Tex

HTML rendering uses MathJax while pdf rendering uses LaTeX:

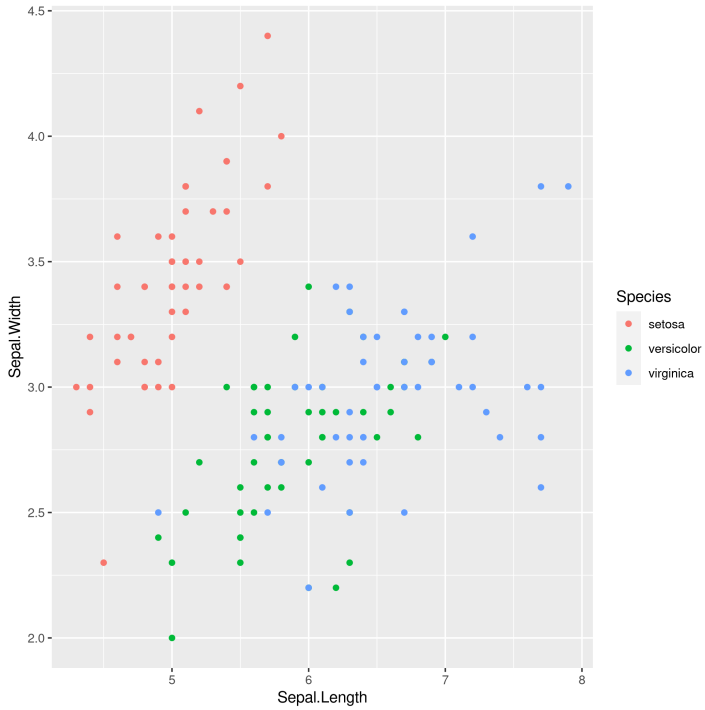


FIGURE 2.1: This is the caption of the figure!

$$f(x) = x^2$$

2.4 Using Stored Results

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	2.1713	0.2798	7.760	0.0000
Sepal.Width	0.4959	0.0861	5.761	0.0000
Petal.Length	0.8292	0.0685	12.101	0.0000
Petal.Width	-0.3152	0.1512	-2.084	0.0389
Speciesversicolor	-0.7236	0.2402	-3.013	0.0031
Speciesvirginica	-1.0235	0.3337	-3.067	0.0026

3

Introduction: Deep Learning for NLP

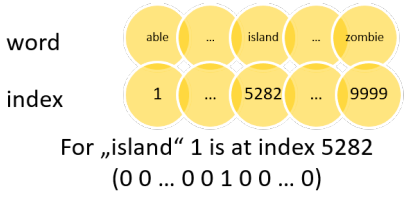
Authors: Viktoria Szabo, Marianna Plesiak, Rui Yang

Supervisor: Christian Heumann

3.1 Word Embeddings and Neural Network Language Models

In natural language processing computers try to analyze and understand human language for the purpose of performing useful tasks. Therefore, they extract relevant information from words and sentences. But how exactly are they doing this? After the first wave of rationalist approaches with handwritten rules didn't work out too well, neural networks were introduced to find those rules by themselves (see Bengio et al. (2003)). But neural networks and other machine learning algorithms cannot handle non-numeric input, so we have to find a way to convert the text we want to analyze into numbers. There are a lot of possibilities to do that. Two simple approaches would be labeling each word with a number (One-Hot Encoding, figure 3.1) or counting the frequency of words in different text fragments (Bag-of-Words, figure 3.1). Both methods result in high-dimensional, sparse (mostly zero) data. And there is another major drawback using such kind of data as input. It does not convey any similarities between words. The word "cat" would be as similar to the word "tiger" as to "car". That means the model cannot reuse information it already learned about cats for the much rarer word tiger. This will usually lead to poor model performance and is called a lack of generalization power.

The solution to this problem is word embedding. Word embeddings use dense vector representations for words. That means they map each word to a continuous vector with n dimensions. The distance in the vector space denotes semantic (dis)similarity. These word embeddings are usually learned by neural networks, either within the final model in an additional layer or in its own model. Once learned they can be reused across different tasks. Practically all NLP projects these days build upon word embeddings, since they have a lot of advantages compared to the aforementioned representations. The basic



Document 1
What time is it?

Document 2
The cat is black.

Term	what	time	is	it	the	cat	black
Doc 1	1	1	1	1	0	0	0
Doc 2	0	0	1	0	1	1	1

FIGURE 3.1: One-Hot Encoding on the left and Bag-of-Words on the right. Source: Own figure.

idea behind learning word embeddings is the so called “distributional hypothesis” (see Harris (1954)). It states that words that occur in the same contexts tend to have similar meanings. The two best known approaches for calculating word embeddings are Word2vec from Mikolov et al. (2013) and GloVE from Pennington et al. (2014). The Word2vec models (Continous Bag-Of-Words (CBOW) and Skip-gram) try to predict a target word given his context or context words given a target word using a simple feed-forward neural network. In contrast to these models GloVe not only uses the local context windows, but also incorporates global word co-occurrence counts. As mentioned, a lot of approaches use neural networks to learn word embeddings. A simple feed-forward network with fully connected layers for learning such embeddings while predicting the next word for a given context is shown in figure 3.2. In this example the word embeddings are first learnt in a projection layer and are then used in two hidden layers to model the probability distribution over all words in the vocabulary. With this distribution one can predict the target word. This simple structure can be good enough for some tasks but it also has a lot of limitations. Therefore, recurrent and convolutional networks are used to overcome the limitations of a simple neural network.

3.2 Recurrent Neural Networks

The main drawback of feedforward neural networks is that they assume a fixed length of input and output vectors which is known in advance. But for many natural language problems such as machine translation and speech recognition it is impossible to define optimal fixed dimensions a-priori. Other models that map a sequence of words to another sequence of words are needed (Sutskever et al., 2014). Recurrent neural networks or RNNs are a special family of neural networks which were explicitely developed for modeling sequential data like text. RNNs process a sequence of words or letters $x^{(1)}, ..., x^{(t)}$ by going through its elements one by one and capturing information based on the previous elements. This information is stored in hidden states $h^{(t)}$ as the network

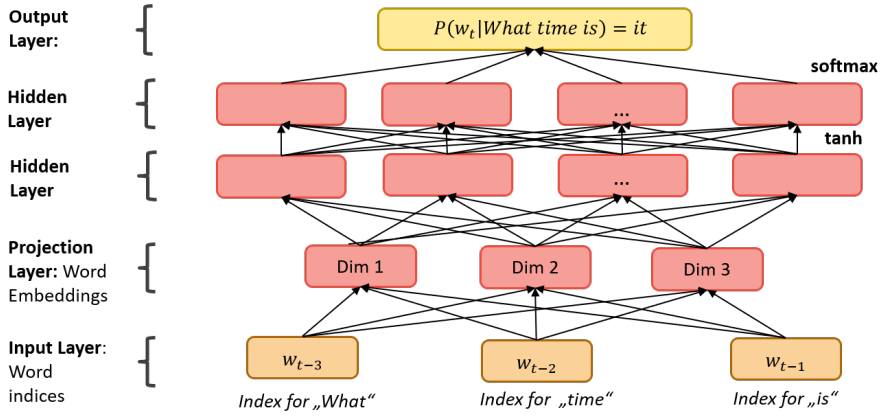


FIGURE 3.2: Feed-forward Neural Network. Source: Own figure based on Bengio et al. 2013.

memory. Core idea is rather simple: we start with a zero vector as a hidden state (because there is no memory yet), process the current state at time t as well as the output from the previous hidden state, and give the result as an input to the next iteration (Goodfellow et al., 2016).

Basically, a simple RNN is a for-loop that reuses the values which are calculated in the previous iteration (Chollet, 2018). An unfolded computational graph (figure 3.3) can display the structure of a classical RNN. The gray square on the left represents a delay of one time step and the arrows on the right express the flow of information in time (Goodfellow et al., 2016).

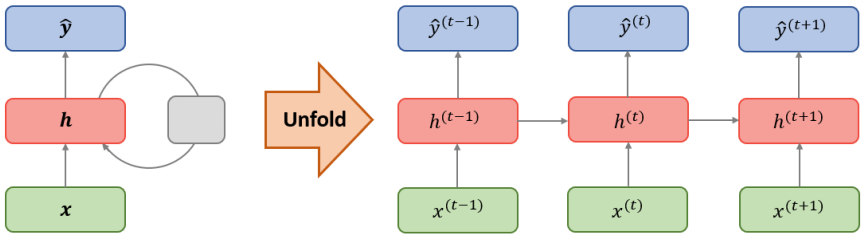


FIGURE 3.3: Right: Circuit diagram (left) and unfolded computational graph (right) of a simple RNN. Source: Own figure.

One particular reason why recurrent networks have become such a powerful technique in processing sequential data is parameter sharing. Weight matrices remain the same through the loop and they are used repeatedly, which makes RNNs extremely convenient to work with sequential data because the model

size does not grow for longer inputs. Parameter sharing allows application of models to inputs of different length and enables generalization across different positions in real time (Goodfellow et al., 2016).

As each part of the output is a function of the previous parts of the output, backpropagation for the RNNs requires recursive computations of the gradient. The so-called backpropagation through time or BPTT is rather simple in theory and allows for the RNNs to access information from many previous steps (Boden, 2002). In practice though, RNNs in their simple form are subject to two big problems: exploding and vanishing gradients. As we compute gradients recursively, they may become either very small or very large, which leads to a complete loss of information about long-term dependencies. To avoid these problems, gated RNNs were developed and accumulation of information about specific features over a long duration became possible. The two most popular types of gated RNNs, which are widely used in modern NLP, are Long Short-Term Memory models (LSTM) and Gated Recurrent Units (GRU) (Goodfellow et al., 2016).

Over last couple of years, various extentions of RNNs were developed which resulted in their wide application in different fields of NLP. Encoder-Decoder architectures aim to map input sequences to output sequences of different length and therefore are often applied in machine translation and question answering (Sutskever et al., 2014). Bidirectional RNNs feed sequences in their original as well as reverse order because the prediction may depend on the future context, too (Schuster and Paliwal, 1997). Besides classical tasks as document classification and sentiment analysis, more complicated challenges such as machine translation, part-of-speech tagging or speech recognition can be solved nowadays with the help of advanced versions of RNNs.

3.3 Convolutional Neural Networks

Throughout machine learning or deep learning algorithms, no one algorithm is only applicable to a certain field. Most algorithms that have achieved significant results in a certain field can still achieve very good results in other fields after slight modification. We know that convolutional neural networks (CNN) are widely used in computer vision. For instance, a remarkable CNN model called AlexNet achieved a top-5 error of 15.3% in the ImageNet 2012 Challenge on 30 September 2012 (see Krizhevsky et al. (2012)). Subsequently, a majority of models submitted by ImageNet teams from around 2014 are based on CNN. After the convolutional neural network achieved great results in the field of images, some researchers began to explore convolutional neural networks in the field of natural language processing (NLP). Early research was restricted to sentence classification tasks, CNN-based models have achieved

very significant effects as well, which also shows that CNN is applicable to some problems in the field of NLP. Similarly, as mentioned before, one of the most common deep learning models in NLP is the recurrent neural network (RNN), which is a kind of sequence learning model and this model is also widely applied in the field of speech processing. In fact, some researchers have tried to implement RNN models in the field of image processing, such as (Visin et al. (2015)). It can be seen that the application of CNN or RNN is not restricted to a specific field.

As the Word2vec algorithm from Mikolov et al. (2013) and the GloVe algorithm from Pennington et al. (2014) for calculating word embeddings became more and more popular, applying this technique as a model input has become one of the most common text processing methods. Simultaneously, significant effectiveness of CNN in the field of computer vision has been proven. As a result, utilizing CNN to word embedding matrices and automatically extract features to handle NLP tasks appeared inevitable.

The following figure 3.4 illustrates a basic structure of CNN, which is composed of multiple layers. Many of these layers are described and developed with some technical detail in later chapters of this paper.

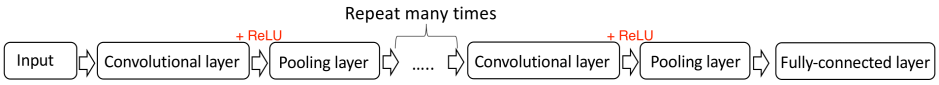


FIGURE 3.4: Basic structure of CNN. Source: Own figure.

It is obvious that neural networks consist of a group of multiple neurons (or perceptron) at each layer, which uses to simulate the structure and behavior of biological nervous systems, and each neuron can be considered as logistic regression.

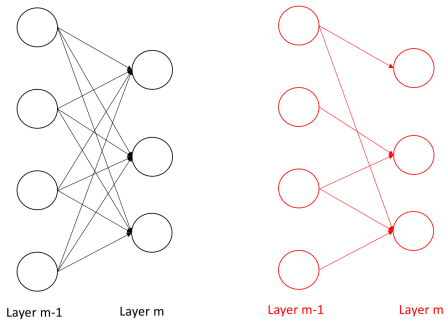


FIGURE 3.5: Comparison between the fully-connected and partial connected architecture. Source: Own figure.

The structure of CNN is different compared with traditional neural networks

as illustrated in figure 3.5. In traditional neural networks structure, the connections between neurons are fully connected. To be more specific, all of the neurons in the layer $m-1$ are connected to each neuron in the layer m, but CNN sets up spatially-local correlation by performing a local connectivity pattern between neurons of neighboring layers, which means that the neurons in the layer $m-1$ are partially connected to the neurons in the layer m. In addition to this, the left picture presents a schematic diagram of fully-connected architecture. It can be seen from the figure that there are many edges between neurons in the previous layer to the next layer, and each edge has parameters. The right side is a local connection, which shows that there are relatively few edges compared with fully-connected architecture and the number of visible parameters has significantly decreased.

A detailed description of CNN will be presented in the later chapters and the basic architecture of if will be further explored. Subsection 5.1 gives an overview of CNN model depends upon (Kim (2014)). At its foundation, it is also necessary to explain various connected layers, including the convolutional layer, pooling layer, and so on. In 5.2 and later subsections, some practical applications of CNN in the field of NLP will be further explored, and these applications are based on different CNN architecture at diverse level, for example, exploring the model performance at character-level on text classification research (see Zhang et al. (2015)) and based on multiple data sets to detect the Very Deep Convolutional Networks (VD-CNN) for text classification (see Schwenk et al. (2017)).

4

Foundations/Applications of Modern NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

5

Recurrent neural networks and their applications in NLP

Author: Marianna Plesiak

Supervisor: Christian Heumann

5.1 RNNs

5.1.1 Structure

Recurrent neural networks allow to relax the condition of non-cyclical connections in the classical feedforward neural networks which were described in the previous chapter. This means, while simple MLP (multilayer perceptrons) can only map from input to output vectors, RNNs allow the entire history of previous inputs to influence the network output. (@ Graves 2012)

The repetitive structure of RNNs can be visualised with help of an **unfolded** computational graph(see @ figure 1).

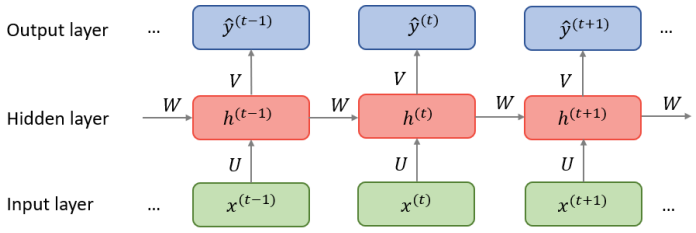


FIGURE 5.1: Unfolded computatinal graph of a RNN.

Each node is associated with a network layer at a particular time instance. Inputs $x^{(t)}$ must be encoded as numeric vectors, for instance word embeddings or one-hot encoded vectors, see previous chapter. Reccurently connected vectors h are called hidden states and represent the outputs of the hidden layers. At time t, a hidden state $h^{(t)}$ combines information from the previous hidden

state $h^{(t-1)}$ as well as the new input $x^{(t)}$ and passes it through to the next hidden layer. Obviously, such an architecture requires the initialization of $h^{(0)}$ since there is no memory at the very beginning of the sequence processing. Given the hidden sequences, output vectors $\widehat{y}^{(t)}$ are used to build the predictive distribution $Pr(x^{(t+1)}|y^{(t)})$ for the next input. Since the predictions are created at each time instance t, the total output has a shape [time_steps, output_features]. However in some cases this is not needed, for example in sentiment analysis the last output of the loop is sufficient because it contains the entire information about the sequence. (@ Chollet) (@ Graves 2012)

The unfolded recurrence can be formalized as following:

$$h^{(t)} = g^{(t)}(x^{(t)}, x^{(t-1)}, ..., x^{(2)}, x^{(1)}) \tag{5.1}$$

$$= f(h^{(t-1)}, x^{(t)}|\theta) \tag{5.2}$$

After t steps, the function $g^{(t)}$ takes into account the whole sequence $(x^{(t)}, x^{(t-1)}, ..., x^{(2)}, x^{(1)})$ and produces the hidden state $h^{(t)}$. Because of its cyclical structure, $g^{(t)}$ can be factorized into the repeated application of a same function f. This function can be considered a universal model which is shared across all time steps and is generalized for all sequence lengths. This is called parameter sharing and is illustrated in the unfolded computational graph as a reuse of the same matrices U, W and V through the entire network. (@ Goodfellow book)

Update equations:

5.1.2 Backpropagation and Drawbacks

5.2 Gated RNNs

5.2.1 LSTM

5.2.2 GRU

5.3 Versions

5.3.1 Bidirectional and Deep RNNs

5.3.2 Applications

6

Convolutional neural networks and their applications in NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

Test test

7

Introduction: Transfer Learning for NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

8

Transfer Learning for NLP I

Authors: Author 1, Author 2

Supervisor: Supervisor

9

Attention and Self-Attention for NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

10

Transfer Learning for NLP II

Authors: Author 1, Author 2

Supervisor: Supervisor

11

Introduction: Resources for NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

12

Resources and Benchmarks for NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

13

Software for NLP: The huggingface transformers module

Authors: Author 1, Author 2

Supervisor: Supervisor

14

Use-Bases for NLP

Authors: Author 1, Author 2

Supervisor: Supervisor

15

Use-Case I

Authors: Author 1, Author 2

Supervisor: Supervisor

16

Use-Case II

Authors: Author 1, Author 2

Supervisor: Supervisor

17

Acknowledgements

The most important contributions are from the students themselves. The success of such projects highly depends on the students. And this book is a success, so thanks a lot to all the authors! The other important role is the supervisor. Thanks to all the supervisors who participated! Special thanks to Christian Heumann[1] who enabled us to conduct the seminar in such an experimental way, supported us and gave valuable feedback for the seminar structure. Thanks a lot as well to the entire Department of Statistics[2] and the LMU Munich[3] for the infrastructure.

The authors of this work take full responsibilities for its content.

[1] https://www.misoda.statistik.uni-muenchen.de/personen/professoren/heumann/index.html

[2] https://www.statistik.uni-muenchen.de/

[3] http://www.en.uni-muenchen.de/index.html

Bibliography

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, (3):1137–1155.

Boden, M. (2002). A guide to recurrent neural networks and backpropagation. *the Dallas project*.

Chollet, F. (2018). *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Harris, Z. S. (1954). Distributional structure. *WORD*, 10(2-3):146–162.

Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, pages 3111–3119.

Pennington, J., Socher, R., Manning, and Christopher D. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

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

Schwenk, H., Barrault, L., Conneau, A., and LeCun, Y. (2017). *Very Deep Convolutional Networks for Text Classification*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Visin, F., Kastner, K., Cho, K., Matteucci, M., Courville, A. C., and Bengio, Y. (2015). *ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks.*

Zhang, X., Zhao, J. J., and LeCun, Y. (2015). *Character-level Convolutional Networks for Text Classification.*