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.

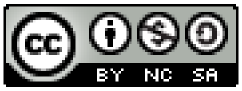


FIGURE 1: Creative Commons License

[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 (**?**). 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.

?

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:

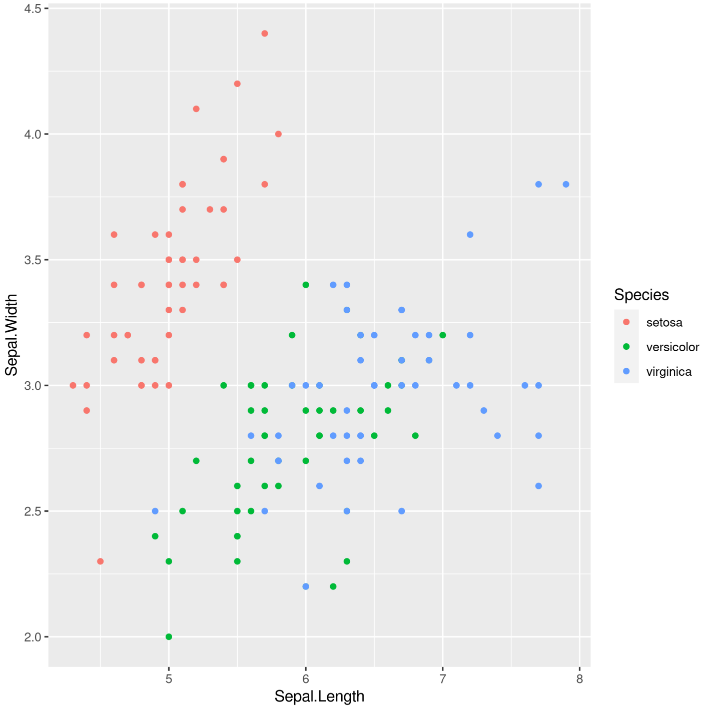


FIGURE 2.1: This is the caption of the figure!

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

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

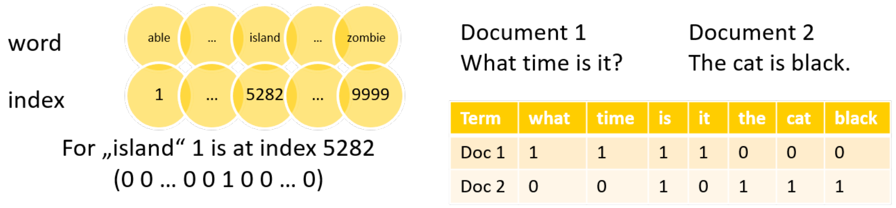
Authors: Viktoria Szabo, Marianna Plesiak, Rui Yang

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 **?**). 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 **?**). 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 **?** and GloVE from **?**. 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 (**?**). 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 memory. Core idea

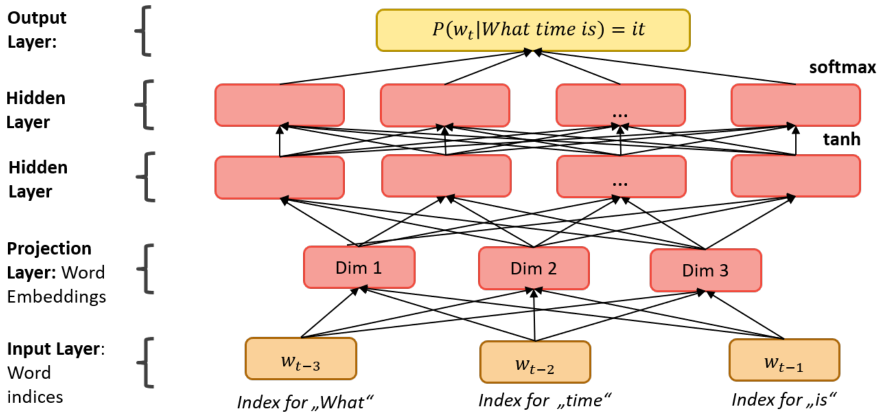


FIGURE 3.2: Feed-forward Neural Network. Source: Own figure based on Bengio et al. 2013.

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 (**?**).

Basically, a simple RNN is a for-loop that reuses the values which are calculated in the previous iteration (**?**). 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 (**?**).

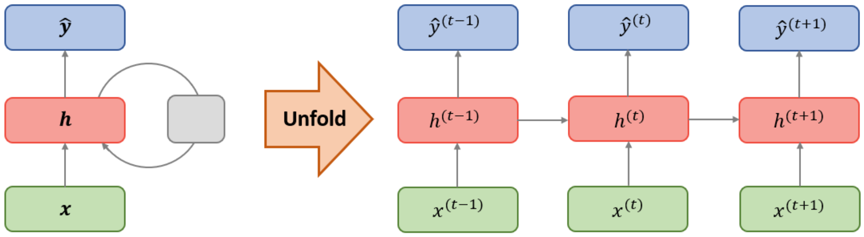


FIGURE 3.3: Right: Circuit diagram (left) and unfolded computational graph (right) of a simple RNN. Source: Own figure.

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 (**?**).

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 (**?**). 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) (**?**).

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 (**?**). Bidirectional RNNs feed sequences in their original as well as reverse order because the prediction may depend on the future context, too (**?**). 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 **?**). 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 (**?**). It can be seen that the application of CNN or RNN is not restricted to a specific field.

As the Word2vec algorithm from **?** and the GloVe algorithm from **?** 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.

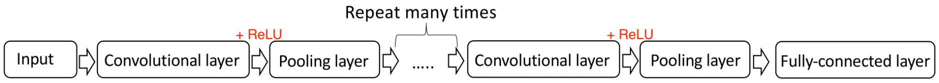


FIGURE 3.4: Basic structure of CNN. Source: Own figure.

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.

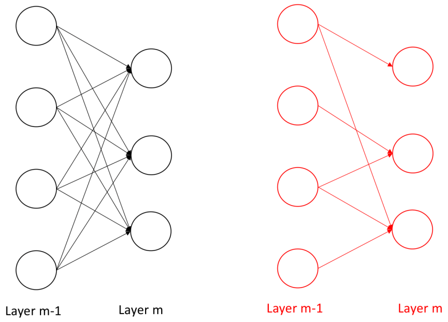


FIGURE 3.5: Comparison between the fully-connected and partial connected architecture. Source: Own figure.

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 (**?**). 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 **?**) and based on multiple data sets to detect the Very Deep Convolutional Networks (VD-CNN) for text classification (see **?**).

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).

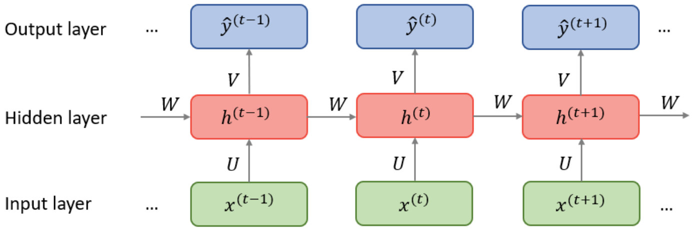


FIGURE 5.1: Unfolded computatinal graph of a RNN.

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 $\hat{y}^{(t)}$ 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:

$$\begin{aligned} h^{(t)} &= g^{(t)}(x^{(t)}, x^{(t-1)}, ..., x^{(2)}, x^{(1)}) \quad (5.1) \\ &= f(h^{(t-1)}, x^{(t)}|\theta) \quad (5.2) \end{aligned}$$

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

Author: Carolin Becker

Supervisor: Matthias Aßenmacher

8.1 Introduction

As mentioned in the introduction, the field of natural language processing (NLP) has seen rapid advancements due to the growing usage of transfer learning. A first enhancement began in 2018, where the transfer learning improved severals field in NLP, as the model layers have not to be trained from scratch. Word2Vec, which was explained earlier (chapter??), is one example of transfer learning.

Scientific papers and research introduced many ideas like contextual word embeddings or fine-tuning. These central changes in the context of the language model are explained in the first section of this chapter, and in a second chapter, the most relevant NLP models will be presented and their contribution to the NLP.

8.2 Steps in sequential inductive transfer learning

[Pan and Young (2010)] divided **transfer learning** in different subfields: if the tasks are the same (*transductive learning*) or if the target and source task shares the same domain (*inductive transfer learning*). In the following, the focus will be on *sequential transfer learning* where tasks are learned sequentially, whereas, in multi-task learning, the tasks are learned simultaneously.

In the first step, all models are **pre-trained** on an extensive source data set, which is, in the best case, very close to the target task ([Peter et al., 2019]). The pre-trained language models in this chapter are uni-directional models that predict the next word.

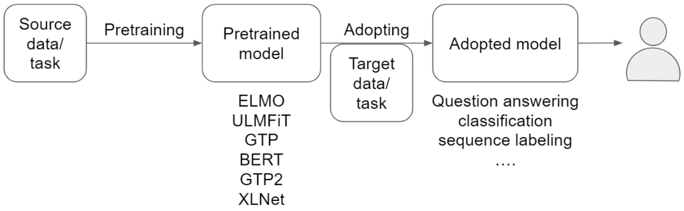


FIGURE 8.1: (ref:ch21-figure1)

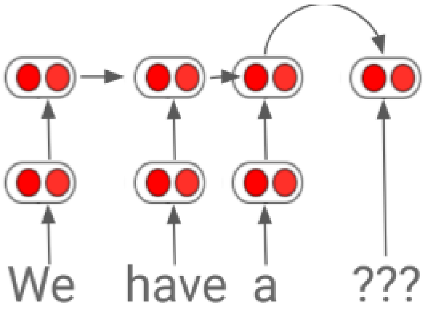


FIGURE 8.2: (ref:ch21-figure7)

In a second step, follows the **adoption** on the target task. Here, the main distinction is, if the pre-trained model architecture is kept (**embedding** or **feature extraction**) or the pre-trained model is adjusted to the target task (**fine-tuning**). [SOURCE??]

In **Embeddings**, single parts, which can be sentences or characters, are extracted to a fixed-length vector with the dimensions $\mathbb{R}^n \times k$ where k is the fixed-length. This matrix represents the context of every word given of ever y other word. So in the adoption phase, the weights in the LM do not change, and just the top layer of the model is used. The adopted model learns a linear combination of the top layer.

On the other side, **fine-tuning** trains the weights of the pre-trained model on a specific task, which makes it much more flexible and needs no specific adjustment. On the other side, the general knowledge and relationship between words can get lost in the adjustment phase. The term for that is the "catastrophic forgetting" (McCloskey & Cohen, 1989; French, 1999). Techniques for preventing this are freezing, learning rates, and regularization.

8.3 Most popular models

In the following sections, the most common models of 2018 **ELMO**, **ULMFiT**, and **GTP** are presented, which shaped the "first wave" of transfer learning before transformers like BERT developed and became popular.

8.3.1 ELMO - The "new age" of embeddings

In 2018, [Peter et al. 2018] from AllenNLP introduced ELMO (Deep-contextualized word representations), which most significant advance compared to previous models like word2vec and Glove (chapter??) is that ELMO can handle the different meanings of a word in a context (polysemy). For instance, is the meaning of the word "mouse" in the context of computers, a device with which you can control the cursor of your PC. Instead, in the context of animals, it means this small animal living in gardens. Until ELMO, this could not be captured by an NLP model, but with ELMO, the different meaning of the words is taken. For this purpose, ELMO can model the semantical and the synthetical characteristics of a word (like word2vec) but also the varying meanings in different contexts.

In contrast to previous word embeddings word, the representations of ELMo are functions of the whole input sentence. These representations are calculated on top of a biLMs (bidirectional language model) with two-layer character convolutions (1) and are a linear combination of the internal network states (2).

8.3.1.1 Bidirectional language model (biLM)

ELMo is based on the shallow concatenation of independently trained left-to-right and right-to-left multi-layer LSTMs. Bidirectional is, in this case, misleading, as the two steps occur independently from each other. A forward langugage model calculates the probability of a sequential token (word or character) t_k at the position k with the a provided history $t_1, \dots, t_{k-1}$ with:

$$p\left(t_1, t_2, \dots, t_N\right) = \prod_{k=1}^{N} p\left(t_k | t_1, t_2, \dots, t_{k-1}\right)$$

The backward LM can be defined accordingly to the forward LM:

$$p\left(t_1, t_2, \dots, t_N\right) = \prod_{k=1}^{N} p\left(t_k | t_{k+1}, t_{k+2}, \dots, t_N\right)$$

In both directions, a context-independent token representation $\mathbf{x}_k^{LM}$ by either token embeddings or a CNN over characters is computed and passed through a forward/backward LSTM. Addionally, at each position k every LSTM layer j outputs a context-dependent representation $\overrightarrow{\mathbf{h}}_{k,j}^{LM}$ (or $\overleftarrow{\mathbf{h}}_{k,j}^{LM}$ in the backward direction).

In the forward direction, a next token t_{k+1} can predict the top layer $\overrightarrow{\mathbf{h}}_{k,L}^{LM}$ with a Softmax layer. In the biLM the direction are combined and optimized with a log likelihood:

$$\begin{gathered}\textstyle\sum_{k=1}^{N}\left(\log p\left(t_k|t_1,\ldots,t_{k-1};\Theta_x,\vec{\Theta}_{LSTM},\Theta_s\right)\right.\\ \left.+\log p\left(t_k|t_{k+1},\ldots,t_N;\Theta_x,\overleftarrow{\Theta}_{LSTM},\Theta_s\right)\right)\end{gathered}$$

where Θ_s are the parameters for the token representations and Θ_x are the parameters of the Softmax-layer.

8.3.1.2 ELMo representation

The ELMO specific task is formulated by

$$\mathbf{ELMo}_k^{task}=E\left(R_k;\Theta^{task}\right)=\gamma^{task}\sum_{j=0}^{L}s_j^{task}\mathbf{h}_{k,j}^{LM},$$

where γ is the optimization paramter which allows to scale the model, s_j^{task} are “softmax-normalized weights” and R_k is the reprresentation of the tookens k:

$$\begin{aligned}R_k&=\left\{\mathbf{x}_k^{LM},\overrightarrow{\mathbf{h}}_{k,j}^{LM},\overleftarrow{\mathbf{h}}_{k,j}^{LM}|j=1,\ldots,L\right\}\\&=\left\{\mathbf{h}_{k,j}^{LM}|j=0,\ldots,L\right\}\end{aligned}.$$

For every task (question answering, sentiment analysis, etc.) the ELMo representation needs a task-specific calculation

8.3.2 ULMFiT - cutting-edge model using LSTMs

Also, 2018, [Howard and Ruder (2018)] proposed **Universal language model fine-tuning (ULMFiT)**, which exceeded many of the cutting-edge models in text classification, as it decreased the error by 18-24% in most of the datasets. ULMFiT is based on an AWD-LSTM combined with novel techniques like “discriminative fine-tuning,” “slanted triangular learning rates,” and “gradual unfreezing of layers.” Hence, it can fine-tune a generalized language model to a specific language model for multiple tasks.

8.3.2.1 AWD-LSTM

As language models with many parameters tend to overfit, [Merity ()] introduced the **AWD-LSTM**, a highly effective version of the Long Short Term

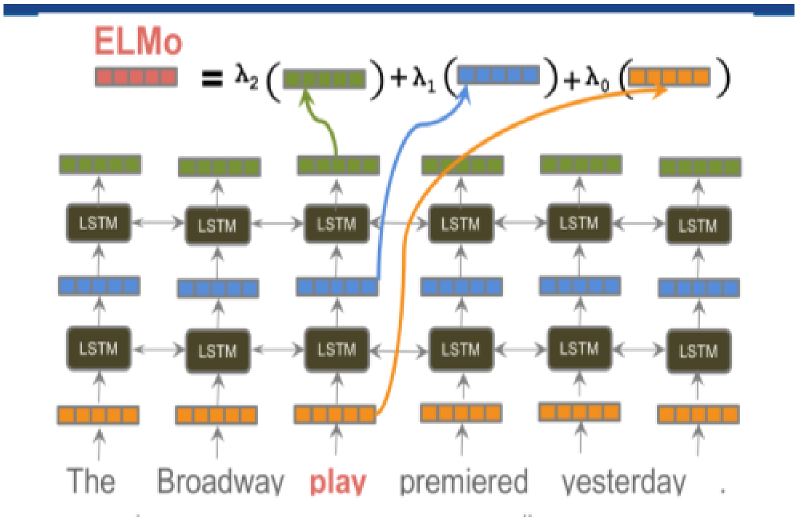
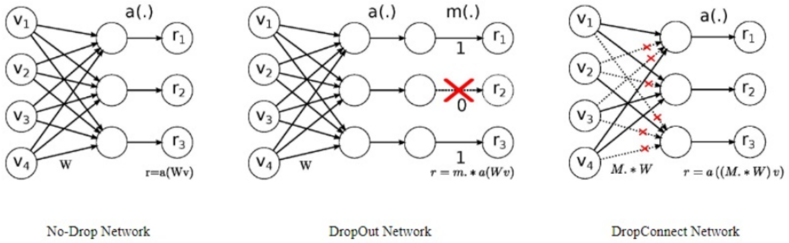


FIGURE 8.3: ELMO

Memory (LSTM, chapter ???). The Dropconnect Algorithm and the Non-monotonically Triggered ASGD (NT-ASGD) are two main improvements of this model architecture.

As in figure ?? The **Dropconnect Algorithm** (Wan 2013) regularizes the LSTM and prevents overfitting by setting the activation of units randomly to zero with a predetermined probability of p. So, only a subset of the units from the previous layer is passed to every unit. However, by using this method also long-term dependencies go lost. That is why the algorithm drops weights and not the activations in the end with the probability of $1-p$. As the weights are set to zero, the drop connect algorithm reduces the information loss, while it reduces overfitting.



No-Drop Network

DropOut Network

DropConnect Network

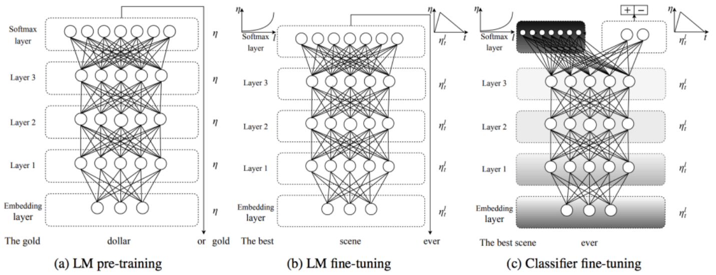
FIGURE 8.4: Dropconnect Algorithm

To improve the optimization in the AWD-LSTM further, [Merity ()] introduced the **Non-monotonically Triggered Average SGD** (or NT-ASGD), which is a new variant of Average Stochastic Gradient Descent (ASDG). The Average Stochastic gradient descent takes a gradient descent step, as the gradient descent algorithm. However, it also takes the weight of the previous iterations into account and returns the average. On the contrary, the NT-ASGD only takes the averaged previous iterations into account, if the validation metric does not improve for several steps. Subsequently, The SGD turns into an ASGD if there is no improvement for n steps.

[formula for ASGD]

In addition to the enhancements above the AWD-LSTM, the authors of the paper propose several other regularization and data efficiency methods: Variable Length Backpropaation Sequences (BPTT), Variational Dropout, Embedding Dropout, Reduction in Embedding Size, Activation Regularization, Temporal Activation Regularization. For further information, the paper is a great way to start. [Mehr hier???]

8.3.2.2 The three steps of ULMFiT



(a) LM pre-training (b) LM fine-tuning (c) Classifier fine-tuning

FIGURE 8.5: Three steps of ULMFiT

ULMFiT follows three steps to achieve these notable transfer learning results:

1. **LM pre-training**: The AWD-LSTM (language model) is trained on general-domain data like the Wikipedia data set.
2. **LM fine-tuning**: The model is fine-tuned on the tasks' dataset. For this purposed [Howard and Ruder(2018)] proposed two training techniques to stabilize the fine-tuning process:
    - **Discriminative fine-tuning** Considering distinctive layers of LM capture distinct types of information, ULMFiT proposed to tune each layer with different learning rates.
    - **Slanted triangular learning rates** (STLR) STLR is a particular learning rate scheduling that first linearly increases the learning rate, and then gradually declines after a cut. That leads to an abrupt increase and a more extensive decay like in figure ??:

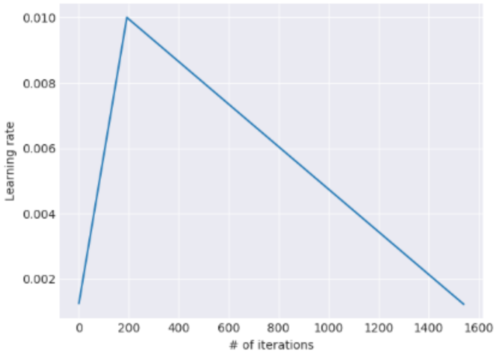


FIGURE 8.6: Slanted triangular learning rates

The learning rates η_t are calculated by the number of iterations T:

$$\begin{aligned} cut &= \lfloor T \cdot cut\_frac \rfloor \\ p &= \begin{cases} t/cut, & \text{if } t < cut \\ 1 - \frac{t - cut}{cut \cdot (1/cut\_frac - 1)}, & \text{otherwise} \end{cases} \\ \eta_t &= \eta_{\max} \cdot \frac{1 + p \cdot (ratio - 1)}{\text{ratio}}, \end{aligned}$$

where cut_frac is the increasing learning rate factor, cut the iteration where the decreasing is started, p the fraction of the number of iterations that are increased or decreased, $ratio$ is the ratio the difference between the lowest and highest learning rate

3. **Classifier fine-tuning**: In the last step, the LM is expanded with two common feed-forward layers and a softmax normalization at the end to predict a target label distribution. Again, two new techniques are submitted:
   - **Concat pooling** Concat pooling elicits “max-pooling and mean-pooling over the history of hidden states and concatenates them with the final hidden state.”

[formula for concat pooling]

```
  + **Gradual unfreezing**
```

A common problem of retraining the model is losing information about the general data (Wikipedia), which is called “catastrophic forgetting.” Hence, gradual unfreezing the model will be trained step by step, starting from the last layer. So first, all layers are “frozen” except the last layer (not tuned). In every step, one additional layer is “unfrozen.”

8.3.3 GTP - First step towards transformers

[Radford-et-al(2018)] from Open AI published **Generative Pre-Training** (GPT). GPT's idea is very similar to ELMO, as it expands the unsupervised LM to a much larger scale, as it uses pre-trained models and transfer learning.

Following the similar idea of ELMo, OpenAI GPT expands the unsupervised language model to a much larger scale by training on a giant collection of free text corpora.

Despite the similarity, GPT has three significant differences to ELMo:

First, ELMO is based on word embeddings, whereas GPT is based on fine-tuning like ULMFiT.

Second, GTP uses a different model architecture. Instead of the multi-layer LSTM, GTP is a multi-layer decoder. A model architecture will be explained in the upcoming chapters, as it is a significant step towards the state-of-the-art NLP models.

Third, in contrast to ELMO, that works character-wise, GPT uses tokens (subwords) from the words.

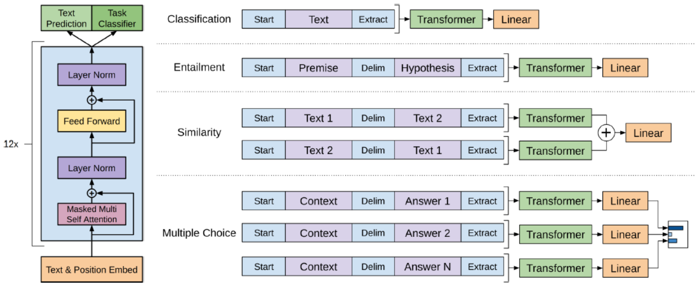


FIGURE 8.7: GPT

As shown in figure `\@ref(fig:ch21-figure04)` GPT is a uni-directional transformer with 12 layers which have two sublayers connected by a feed-forward network: First, the model is pre-trained by thousand books from Google books. Then, the parameters of the model are adopted on the specific task.

As the basic idea of transformers is discussed in the following chapters, further explanations of the functionality of the transformer model architectures will follow.

8.4 Summary

In 2018, a new generation of NLP models had been published, as transfer learning mainly pushed further enhancements from computer vision. The **main advances** of these models are

- due to the use of **transfer learning** the training for the target task needs less time and less target specific data,
- **ELMO** adds the contextualization to word embeddings,
- **ULMFiT** introduces many ideas like fine-tuning, which undoubtedly lowered the error rate notable, and
- **GTP** uses first the transformer model architecture, which cutting-edge NLP models use.

Besides, many features of these models show high **potential for improvements**:

- All models are **not genuinely bi-directional**, as ULMFiT and GTP are one-directional, and ELMo is a concatenation of a right-to-left and left-to-right LSTM. Bidirectional models can even have more precise word representations, as the human language understanding is bidirectional.
- ELMo uses character-based **model input**, and **ULMFit** uses word-based **model input**. **GPT** and following transformer-based models use **tokenized** words (subwords), which is take advantage of both other model inputs.
- ULMFiT and GPT use **fine-tuning**, which needs no further adjustments for every target task.
- ULMFiT and ELMO are **based on LSTMs**, whereas the transformer-based model architecture of GPT has many advantages like parallelization and subsequent performance improvements.

In the next chapter, the main idea behind transformers, self-attention, is explained. More popular state-of-art models based on the idea like BERT are presented in chapter 10 [??].

8.5 Was noch fehlt:

- inserting the literature correctly with Bibtex
- language and typos
- Bilder verbessern
- groß und kleinschreibung von fachbegriffen
- fett drucken von ichtigen wörtern??

8.6 Literature:

Pan, S. J., and Yang, Q. (2010). A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359.

Evolution of Transfer Learning in Natural Language Processing Aditya Malte, Pratik Ratadiya (2019) https://arxiv.org/pdf/1910.07370.pdf

Stephen Merity, Nitish Shirish Keskar, Richard Socher," Regularizing and Optimizing LSTM Language Models," arXiv:1708.02182 [cs.CL]

Wan et al. 2013 "Regularization of Neural Networks using DropConnect" http://proceedings.mlr.press/v28/wan13.html

2018-peters-et-al_ELMO.pdf Aus https://moodle.lmu.de/pluginfile.php/450729/mod_folder/content/0/2018-peters-et-al_ELMO.pdf?forcedownload=1

2018-Howard-ruder_ulmfit.pdf Aus https://moodle.lmu.de/pluginfile.php/450729/mod_folder/content/0/2018-howard-ruder_ulmfit.pdf?forcedownload=1

2018-Radford-et-al_openAIgpt.pdf Aus https://moodle.lmu.de/pluginfile.php/450729/mod_folder/content/0/2018-radford-et-al_openAIgpt.pdf?forcedownload=1

Peter et al. 2019 https://arxiv.org/abs/1903.05987

(McCloskey & Cohen, 1989; French, 1999

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