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:

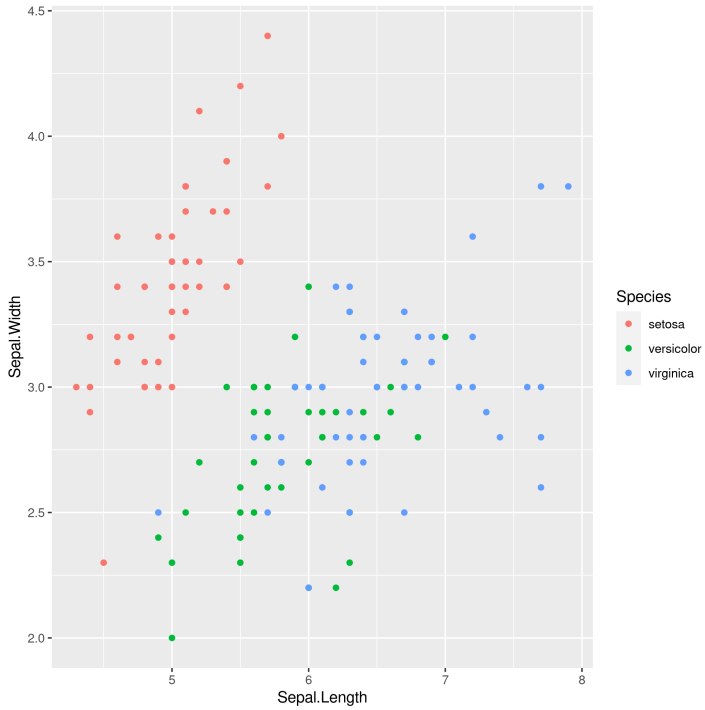


FIGURE 2.1: This is the caption of the figure!

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

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

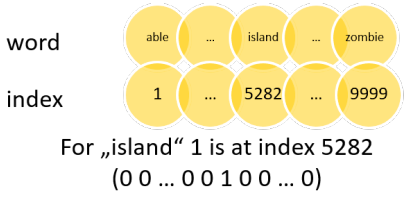
Authors: Viktoria Szabo, Marianna Plesiak, Rui Yang

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

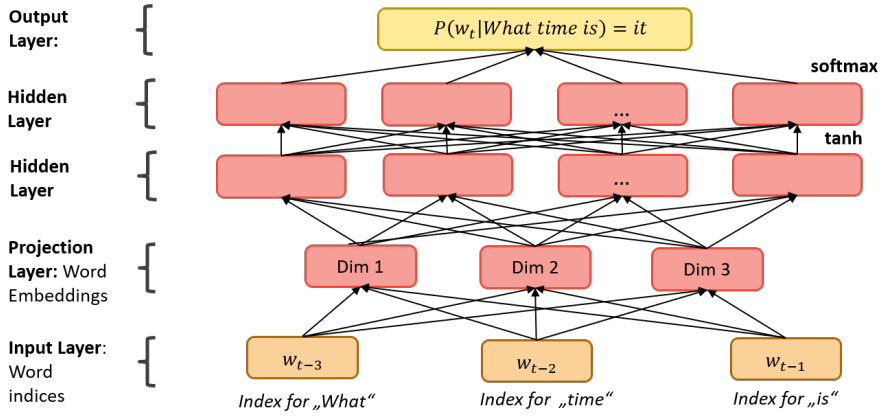


FIGURE 3.2: Feed-forward Neural Network. Source: Own figure based on Bengio et al. 2013.

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).

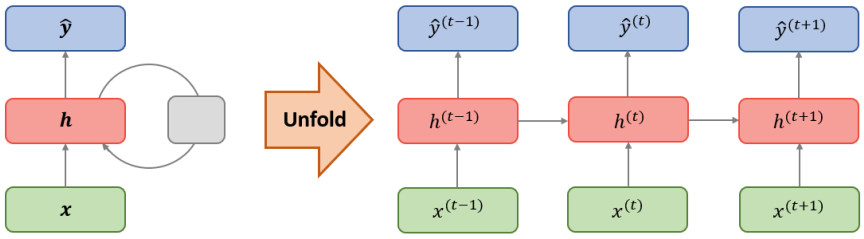


FIGURE 3.3: Right: Circuit diagram (left) and unfolded computational graph (right) of a simple RNN. Source: Own figure.

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.

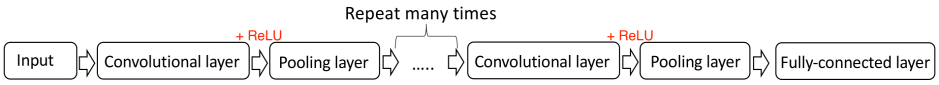


FIGURE 3.4: Basic structure of CNN. Source: Own figure.

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.

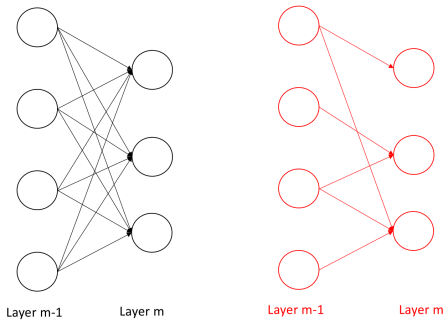


FIGURE 3.5: Comparison between the fully-connected and partial connected architecture. Source: Own figure.

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).

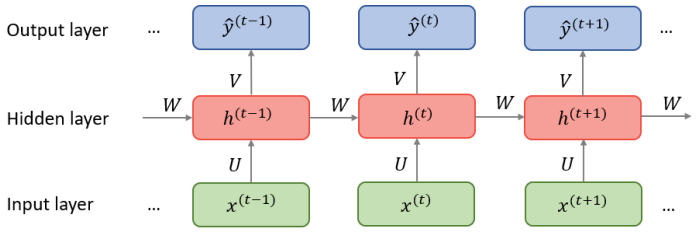


FIGURE 5.1: Unfolded computatinal graph of a RNN.

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: Carolin Becker, Joshua Wagner, Bailan He

Supervisor: Matthias Aßenmacher

As discussed in the previous chapters, natural language processing (NLP) is a very powerful tool in the field of processing human language. In recent years, there have been many proceedings and improvements in NLP to the state-of-art models like BERT. A decisive further development in the past was the way to transfer learning, but also self-attention.

In the next three chapters, various NLP models will be presented, which will be taken to a new level with the help of transfer learning in a first and a second step with self-attention and transformer-based model architectures. To understand the models in the next chapters, the idea and advantages of transfer learning are introduced. Additionally, the concept of self-attention and an overview over the most important models will be established

7.1 What is Transfer Learning?

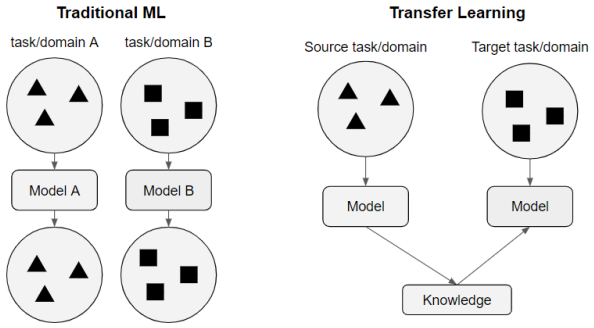


FIGURE 7.1: Classic Machine Learning and Transfer Learning

In figure 7.1 the difference between classical machine learning and transfer learning is shown.

For classical machine learning a model is trained for every special task or domain. Transfer learning allows us to deal with the learning of a task by using the existing labeled data of some related tasks or domains. Tasks are the objective of the model. e.g. the sentiment of a sentence, whereas the domain is where data comes from. e.g. all sentences are selected from Reddit. In the example above, knowledge gained in task A for source domain A is stored and applied to the problem of interest (domain B).

Generally, transfer learning has several advantages over classical machine learning: saving time for model training, mostly better performance, and not a need for a lot of training data in the target domain.

It is an especially important topic in NLP problems, as there is a lot of knowledge about many texts, but normally the training data only contains a small piece of it. A classical NLP model captures and learns a variety of linguistic phenomena, such as long-term dependencies and negation, from a large-scale corpus. This knowledge can be transferred to initialize another model to perform well on a specific NLP task, such as sentiment analysis. (Malte and Ratadiya, 2019)

7.2 (Self-)attention

The most common models for language modeling and machine translation were, and still are to some extent, recurrent neural networks with long short-term memory (Hochreiter and Schmidhuber, 1997) or gated recurrent units (Chung et al., 2014). These models commonly use an encoder and a decoder archictecture. Advanced models use attention, either based on Bahdanau's attention (Bahdanau et al., 2014) or Loung's attention (Luong et al., 2015).

Vaswani et al. (2017) introduced a new form of attention, self-attention, and with it a new class of models, the *Transformers*. A Transformer still consists of the typical encoder-decoder setup but uses a novel new architecture for both. The encoder consists of 6 Layers with 2 sublayers each. The newly developed self-attention in the first sublayer allows a transformer model to process all input words at once and model the relationships between all words in a sentence. This allows transformers to model long-range dependencies in a sentence faster than RNN and CNN based models. The speed improvement and the fact that "individual attention heads clearly learn to perform different tasks" Vaswani et al. (2017) lead to the eventual development of **B**idirectional **E**ncoder **R**epresentations from **T**ransformers by Devlin et al. (2018). **BERT** and its successors are, at the time of writing, the state-of-the-art models used

for transfer learning in NLP. The concepts attention and self-attention will be further discussed in the **"Chapter 9: Attention and Self-Attention for NLP"**.

7.3 Overview over important NLP models

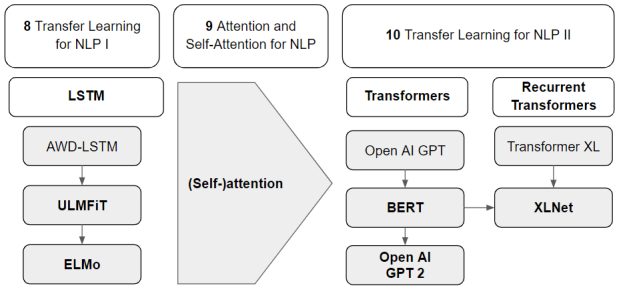


FIGURE 7.2: Overview of the most important models for transfer learning

The models in figure 7.2 will be presented in the next three chapters.

First, the two model architectures ELMo and ULMFit will be presented, which are mainly based on transfer learning and LSTMs, in **Chapter 8: "Transfer Learning for NLP I"**:

- **ELMo** (Embeddings from Language Models) first published in Peters et al. (2018) uses a deep, bi-directional LSTM model to create word representations. This method goes beyond traditional embedding methods, as it analyses the words within the context
- **ULMFiT** (Universal Language Model Fine-tuning for Text Classification) consists of three steps: first, there is a general pre-training of the LM on a general domain (like WikiText-103 dataset), second, the LM is finetuned on the target task and the last step is the multilabel classifier fine tuning where the model provides a status for every input sentence.

In the **"Chapter 10: Transfer Learning for NLP II"** models like BERT, GTP2 and XLNet will be introduced as they include transfer learning in combination with self-attention:

- **BERT** (Bidirectional Encoder Representations from Transformers Devlin et al. (2018)) is published by researchers at Google AI Language group. It is regarded as a milestone in the NLP community by proposing a bidirectional Language model based on Transformer. BERT uses the Transformer Encoder as the structure of the pre-train model and addresses the unidirectional

constraints by proposing new pre-training objectives: the "masked language model"(MLM) and a "next sentence prediction"(NSP) task. BERT advances state-of-the-art performance for eleven NLP tasks and its improved variants **Albert** Lan et al. (2019) and **Roberta** Liu et al. (2019) also reach great success.

- **GPT2** (Generative Pre-Training-2, Radford et al. (2019)) is proposed by researchers at OpenAI. GPT-2 is a tremendous multilayer Transformer Decoder and the largest version includes 1.543 billion parameters. Researchers create a new dataset "WebText" to train GPT-2 and it achieves state-of-the-art results on 7 out of 8 tested datasets in a zero-shot setting but still underfits "WebText".
- **XLNet** is proposed by researchers at Google Brain and CMU(Yang et al., 2019). It borrows ideas from autoregressive language modeling (e.g., Transformer-XL Dai et al. (2019)) and autoencoding (e.g., BERT) while avoiding their limitations. By using a permutation operation during training, bidirectional contexts can be captured and make it a generalized order-aware autoregressive language model. Empirically, XLNet outperforms BERT on 20 tasks and achieves state-of-the-art results on 18 tasks.

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

Natural Language Generation

Author: Haris Jabbar

Supervisor: Matthias Aßenmacher

15.1 Introduction

Machine learning systems can be differentiated into two types : Discriminative and Generative. While discriminative systems like classification, regression, clustering are the more well known type, it's the Generative systems that hold greater promise of achieving Artificial General Intelligence. In essence, a Generative system is expected to produce images, text or audio that would be meaningful to the users. That's a much harder problem than just identifying a horse in a picture or fitting a line through a set of datapoints.

In this chapter, I will tackle this generative process in Natural Language Processing domain. Understandably, it's called Natural Language Generation (NLG).

15.2 Definition

Reiter and Dale (2000) defined Natural Language Generation (NLG) as "the sub-field of artificial intelligence and computational linguistics that is concerned with the construction of computer systems than can produce understandable texts in English or other human languages from some underlying non-linguistic representation of information"

15.3 Domains of NLG Systems

NLG systems have a wide variety of application areas.

1. Text-to-Text
   - Machine Translation
   - Text Summarization
2. Data-to-Text
   - Image Captioning
   - Business Intelligence
3. Ideas-to-Text
   - Poetry/Song Generation
   - Mimic an artist
   - Fake News
4. Dialog Systems (Chatbots)
   - Goal Oriented
   - Open ended conversations

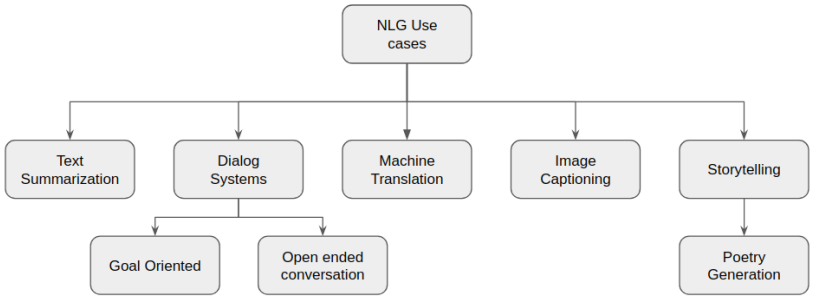


FIGURE 15.1: NLG Use Cases

15.4 Commercial Activity

1. https://www.narrativescience.com
2. https://automatedinsights.com
3. http://www.arria.com

16

The Architectures

There are many architectures that are peculiar to an NLG system. While some are used in other domains as well, in the following sections I will explain them with a focus on language generation.

16.1 Encoder-Decoder Architecture

The most ubiquitous architecture for NLG is the encoder-decoder architecture, and especially the decoder part of it. Hence I will explain it in some detail. The architecture is shown in following figures:

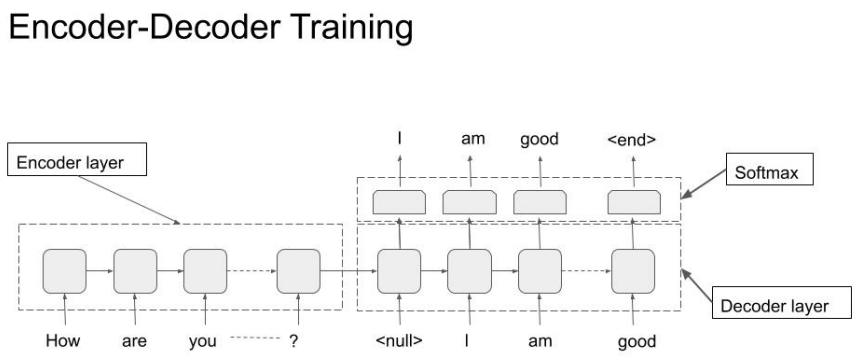


FIGURE 16.1: Encoder-Decoder (Training)

The architecture can be seen as conditional probability P(y/x) with 'y' being the decoder and it is conditioned on 'x' (the encoder). Hence the NLG task becomes generating text through decoder conditioned on some input, coming from the encoder.

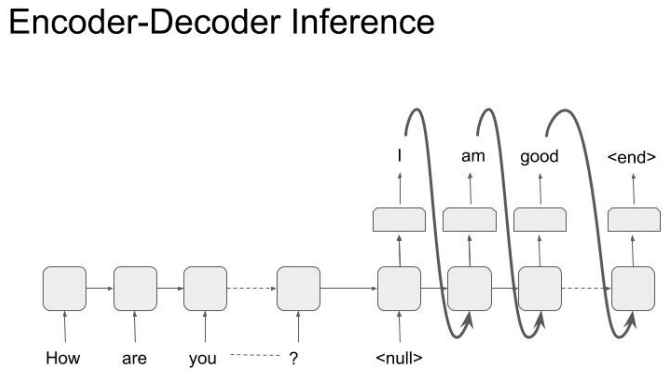


FIGURE 16.2: Encoder-Decoder (Inference)

16.1.1 Encoder :

As stated above, the purpose of this part of the network is to provide conditional information on which the decoder generates text. As such, this part can have ANY architecture. It can be a convolutional neural network for image based conditioning, or RNN/LSTM/Transformer architecture for text or audio based conditioning. But for the purpose of illustration we are using an RNN/LSTM and text as input condition.

The thing to note here is that the richer the feature vector going from encoder to decoder, the more information decoder would have to generate better output. This was the motivation to move from single feature vector (\cite) to multiple vectors and subsequently to a transformer based architecture.

16.1.2 Decoder :

The decoder is the most distinctive part of an NLG system. Almost all decoders have the same form as shown in the figures above. The purpose is to generate text tokens (\cite) one after the other until a terminating criteria is met. This termination is usually a termination token (<end> in the figures).

During training, we are given an input (text/image/audio) and the 'gold label text' that we want the system to learn to generate for that particular input. In the given example, the input is text "How are you?" and the gold label is "I am good". The input goes through the encoder and produces a feature vector that is used as the input to decoder. The decoder than generates tokens one by one and the loss is calculated after the softmax layer from the generated

token and the gold label token. Note the inclusion of an extra token '<null>' as the first token. The last token of the gold label should produce the '<end>' token.

During inference, we don't have the gold label, so the output of one step is used as input to next step, as shown in the figure. Note that it matches with the setup during training. The generator stops when '<end>' token is emitted; thus completing the inference.

16.2 Attention Architecture

The attention architecture is introduced in detail in section (\cite). Here I will briefly mention its use in NLG systems. Looking at the picture below, we can see that the attention is from decoder to encoder.

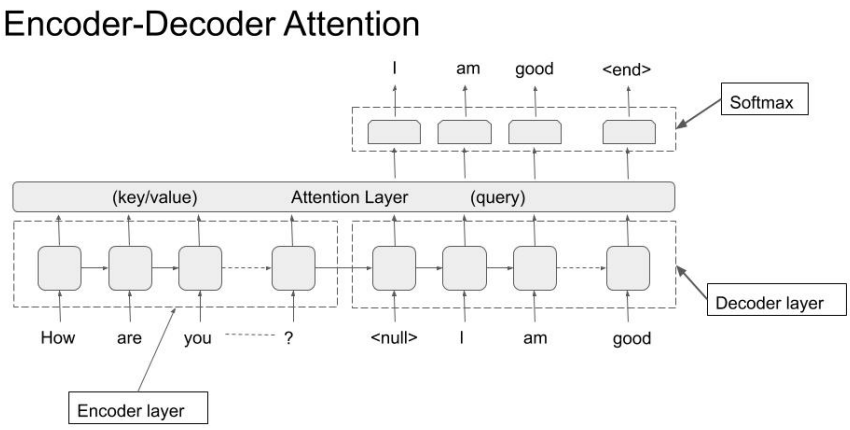


FIGURE 16.3: Encoder-Decoder (Inference)

In other words, before generating each token, the decoder attends to all tokens in the encoder, as shown. The query is the decoder token and key/values are all encoder token. That way the decoder has much richer information to base its output on.

16.3 Decoding Algorithm at Inference

Now I will explain the decoding algorithms that are used to generate text from the softmax layer.

As explained above, during inference, the tokens are generated sequentially. In a vanilla version of decoding, at each step of the sequence, the token with highest probability in the softmax layer is generated. This is called 'greedy decoding', but it has been shown to produce suboptimal text. There are few improvements over this greedy approach.

16.3.1 Beam Search

In greedy decoder we simply output the maximum probability token at each step. But if we track multiple words at each step and then output the sequence formed by highest probability combination, we get beam search. The number of tokens we keep track of is the length of beam (k). The algorithm than goes as follows:

1. Select k-tokens with highest probability at step 0.
2. Use these k-tokens to generate k softmax vectors at step 1.
3. Keep the k highest scoring combinations.
4. Repeat steps 2 and 3 till <end> token is generated, or a predefined max is reached
5. At each step, we have only k-hypothesis, which is the length of beam search.

While beam search tends to improve the quality of generated output, it has its own issues. Chiefly among them is that it tends to produce shorter sequences. Although it can be controlled by the max parameter (of step 4), it's another hyperparameter to be reckoned with.

16.3.2 Pure Sampling Decoder

Here, at each step, instead of taking the token with maximum probability like in greedy search, the token is sampled from the whole vocabulary according to the probability distribution predicted by the softmax layer.

16.3.3 K-sampling Decoder

It's like pure sampling decoder, but instead of sampling from whole vocabulary, the token is sampled only from the k-highest probability tokens.

16.4 Memory Networks

Memory Networks is another architecture that is quite useful in language generation tasks. The basic premise is that LSTMs/RNNs and even Transformer architecture stores all the information only in the weights of the network. When we want to generate text that should include information from a large knowledge base, this 'storage' of network weights is insufficient. So the network employs an external storage (the memory) that it can use during language generation.

16.5 Language Models

Language models are probably the most important ingredient of generating text. As the name implies, they model how the langauge probability distribution. More concretely, they model the conditional probability distribution P(w_t/w_{t-1}). This can then be used to generate text.

17

Question-Answer Systems

The question-answer systems attempt to extract or generate answers from a given question and either a fixed or open ended context. The context here is the corpus from which the answer needs to be generated. For example in SQUAD dataset (\cite) the context is a given paragraph from wikipedia and the question is asked from that paragraph. In open ended contexts, the whole wikipedia (or other corpus) can be the contexts.

17.1 Datasets

- msmarco
- google natural questions
- multiple choice
  - swag/trivia qa
- conversational qa
  - coqa, wizard of wikipedia, quac.ai
- many others e.g. visual qa, KB qa etc.

17.2 Types

- Question Answer Systems
  - Structured knowledge source The sources can be e.g. Freebase, Wikidata, DBpedia or RDBMS systems
  - Unstructured knowledge soure Free text e.g. Wikipedia
  - FAQs Extract answers for similar questions to the given in a corpus of question-answer pairs.

17.3 Architectures

- Context
- Question

Five conceptual levels - Token level features (embeddings) - Context and question encoder - Attention from context to question or vice versa - Modeling layer - Output layer

- Pointer Networks
- Open Domain QA
  - DrQA
  - Distant Supervision

17.4 Evaluation Metrics

There are generally two metrics commonly used in QA systems. The exact match (EM) and F1 score.

18

Dialog Systems

These are the systems where an agent chats with a human being either with a specific pupose (goal oriented) or it is a general open ended chat. The examples of goal oriented chats include tasks like booking an appointment or a flight ticket. Open ended chats can be talking about a general topic which may or may not include a 'personality' for the chatbot.

18.1 Types

- Chatbots
  - Open domain
  - Goal Oriented

18.2 Architectures

- Information Retrieval
  - Inbuilt in the model weights
  - External Source
    - Memory Networks
    - API calls (?)
- Text Generation
  - Encoder-Decoder

19

Conclusion

Natural Language Generation (NLG) has huge potential to be not only an academic domain for research but to affect and improve our daily lives. In this chapter I have talked about only two of its manifestations.

20

Use-Case II

Authors: Author 1, Author 2

Supervisor: Supervisor

21

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

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

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.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

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

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

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

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*.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Malte, A. and Ratadiya, P. (2019). Evolution of transfer learning in natural language processing.

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.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations.

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

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

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.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

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*.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-

Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 5753–5763. Curran Associates, Inc.

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