University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Distributional semantics using neural networks

PhD Study Report

Ing. Lukáš Svoboda

Technical Report No. DCSE/TR-2016-04
June, 2016

Technical Report No. DCSE/TR-2016-04
June 2016

Distributional semantics using neural networks

Ing. Lukáš Svoboda

Abstract

During recent years, neural networks show crucial improvement in catching semantics of words or sentences. They also show improves in Language modeling, which is crucial for many tasks among Natural Language Processing (NLP).

One of the most used architectures of Artificial Neural Networks (ANN) in NLP are Recurrent Neural Networks (RNN) that do not use limited size of context. By using recurrent connections, information can cycle in side these networks for arbitrarily long time.

Thesis summarizes the state-of-the-art approaches to distributional semantics. Thesis also focus on further use of ANN among NLP problems.

This work was supported by Grant No. SGS-2016-018 Data and Software Engineering for Advanced Applications. Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures..

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Contents

1 Introduction **1**

2 Distributional Semantics **2**

2.1 Hypotheses 2

2.1.1 Distributional Hypothesis 2

2.1.2 Bag-of-word Hypothesis 3

2.2 Word Semantic Approaches 3

2.2.1 Context Types 4

2.2.2 Model Architectures 4

2.3 Language Models 5

2.3.1 Statistical Language Models 6

2.3.2 N-gram Language Models 7

2.3.3 Class-based N-gram Models 7

3 Neural Networks **9**

3.1 Introduction 9

3.2 Machine Learning 10

3.2.1 Logistic Regression Classifier 10

3.2.2 Naive Bayes Classifier 11

3.2.3 SVM Classifier 12

3.2.4 Clustering (Word Classes) 13

3.3 Perceptron 14

3.4 Training of Neural Networks 15

3.4.1 Forward pass 15

3.4.2 Backpropagation 16
3.4.3 Regularization 19
3.5 Feed-forward Neural Networks 20
3.6 Convolutional Neural Networks 20
3.6.1 Hyperparameters 22
3.6.2 Pooling Layers 22
3.6.3 Channels 23
3.6.4 Use of CNNs in NLP 23
3.7 Recurrent Neural Networks 25
3.7.1 RNNs with Long Short-Term Memory 26
3.8 Recursive Neural Network 29
3.9 Deep Learning 29
3.9.1 Representations and Features Learning Process . . . 30

4 Distributional Semantics Using Neural Networks 32
4.0.2 Continuous Bag-of-Words 33
4.0.3 Skip-gram 34
4.0.4 Paragraph Vectors 34
4.0.5 Tree-based LSTM 35

5 Preliminary Results 36

6 Summary and Future Work 38
6.1 Overall Aims of the PhD Thesis 38

Chapter 1

Introduction

Understanding the meaning of the text is crucial among many NLP tasks. The model that improves on understanding the text may also improve the particular job where the model is used.

The Internet changed our life and the way of thinking enormously. About 50% of human population is using Internet. In the social web people have found a new way to communicate and we can find about 1 billion of web pages on the Internet. Internet became almost an infinite source of texts that can be used in models based on distributional hypothesis (see Chapter 2).

In this thesis we are going to investigate the models based on distributional semantics (see Chapter 4) which says that similarly distributed words tend to have similar meaning. Research into distributional semantics is evolving more than 20 years. Most of techniques for modeling semantics has been outperformed by neural networks based models and deep learning during recent years. These models showed crucial improvement in meaning representation of a word and sentence/document. Such models usually work with plain text data and unsupervised way of representation learning.

Thesis is organized as follows, the state-of-the-art architectures for distributional semantics are discussed in Chapter 2. Chapter 3 discusses problem of standard machine learning approaches dealing with NLP problems and describes neural networks architectures that play the current key role in modeling semantics.

Some preliminary ideas to future development that imply the aims of doctoral thesis are discussed in Chapter 5.

Chapter 2

Distributional Semantics

Distributional semantics is a research area that develops and studies theories and methods for quantifying and categorizing semantic similarities between linguistic items based on their distributional properties in large samples of language data. The basic idea of distributional semantics can be summed up in the so-called Distributional hypothesis: *"linguistic items with similar distributions have similar meanings"*.

2.1 Hypotheses

2.1.1 Distributional Hypothesis

Famous quotes:

- [1]: *"If we consider words or morphemes A and B to be more different in meaning than A and C, then we will often find that the distributions of A and B are more different than the distributions of A and C."*
- [2]: *"You shall know a word by the company it keeps."*

This principle is known as the *distributional hypothesis*. The direct implication of this hypothesis is that the word meaning is related to the context where it usually occurs and thus it is possible to compare the meanings of two words by statistical comparisons of their contexts. This implication was confirmed by empirical tests carried out on human groups in [3; 4].

The distributional semantics models typically represent the word meaning as a vector, where the vector reflects the contextual information of a word across the training corpus. Each word $w \in W$ (where W denotes the word vocabulary) is associated with a vector of real numbers $\boldsymbol{w} \in \mathbb{R}^k$. Represented

geometrically, the word meaning is a point in a high-dimensional space. The words that are closely related in meaning tend to be closer in the space.

2.1.2 Bag-of-word Hypothesis

In this context, the term *bag* means a *set* where the order has no role, however, the duplicates are allowed (the bags a, a, a, b, b, c and c, a, b, a, b, a are equivalent).

The early reference can be found in [1], but the first practical application was arguably in information retrieval. In work of [5], the documents were represented as bags-of-words and the frequencies of words in a document indicated the relevance of the document to a query. The implication is that two documents tend to be similar if they have similar distribution of similar words, no matter what is their order. This is supported by the intuition that the topic of a document will probabilistically influence the author's choice of words when writing the document.

Similarly, the words can be found related in meaning if they occur in similar documents (where document represents the word context). Thus, both hypotheses (bag-of-words hypothesis and distributional hypothesis) are related.

This intuition later expanded into many often used models for meaning extraction, such as Latent Semantic Analysis (LSA) [6], Probabilistic Latent Semantic Analysis (PLSA) [7], Latent Dirichlet Allocation (LDA) [8], and others.

2.2 Word Semantic Approaches

The models based upon distributional hypothesis are often referred to as the *distributional semantics models*.

These models typically represent the word meaning as a vector which reflects the contextual information of a word across the training corpus. Each word $w \in W$ is associated with a vector of real numbers $\boldsymbol{w} \in \mathbb{R}^k$. Represented geometrically, the meaning is a point in a k-dimensional space. The words that are closely related in meaning tend to be closer in the space. This architecture is sometimes referred to as the *Vector Space Model* (VSM) or *Semantic Space* (SS). Such methods are often called Word Embedding methods.

In last years, the extraction of meaning from a text became the backbone research area in natural language processing. It led to impressive results (see Section 4).

2.2.1 Context Types

Different types of context induce different kinds of semantic space models. [9] and [10] distinguish *context-word* and *context-region* approaches to the meaning extraction. In this thesis we use the notion *local context* and *global context*, respectively, because we think this notion describes the principle of the meaning extraction better.

- **Global context:** The models that use the global context are usually based upon bag-of-word hypothesis, assuming that the words are semantically similar if they occur in similar documents, and that the word order has no meaning. The document can be a sentence, a paragraph, or an entire text. These models are able to register long-range dependencies among words. For example, if the document is about *hockey*, it is likely to contain words like *hockey-stick* or *skates*, and these words are found to be related in meaning.
- **Local context:** The models that collect short contexts around the word using moving window to model its semantics. These methods do not require text that is naturally divided into documents or pieces of text. Thanks to the short context, these models can take the word order into account, thus they usually model semantic as well as syntactic relations among words. In contrast to the global semantics models, these models are able to find mutually substitutable words in the given context. Given the sentence: *The dog is an animal*, the word *dog* can be for example replaced by *cat*.

2.2.2 Model Architectures

There are several architectures that have been successfully used to extract meaning from raw text. In our opinion, the following four architectures are the most important:

- **Co-occurrence matrix:** The frequencies of co-occurring words (often taken as an argument of some weighting function, e.g. *Term Frequency – Inverse Document Frequency* (TF-IDF), mutual information, etc.) are recorded into a matrix. The dimension of such matrix is sometimes found to be problematic (it is proportional to the number of contexts in the text), and thus the *Singular Value Decomposition* (SVD) or different algorithm can be used for dimensionality reduction.

  GloVe (Global Vectors) [11] represents the model focusing on the global statistics of the trained data. This approach analyses log-bilinear regression models that effectively capture global statistics and

also captures word analogies. Authors propose a weighted least squares regression model that trains on global word-word co-occurrence counts. The main concept of this model is the observation that ratios of word-word co-occurrence probabilities have the potential for encoding meaning of words.

- **Topic model:** The group of methods based upon the bag-of-word hypothesis that try to discover latent (hidden) topics in the text are called *topic models*. They usually represent the meaning of the text as a vector of topics but it is also possible to use them for representing the meaning of a word. The number of topics in the text is usually set in advance.

  It is assumed that documents may vary in domain, topic and styles, which means that they also differ in the probability distribution of n-grams. This assumption is used for adapting language models to the long context (domain, topic, style of particular documents). LSA[6] (or similar methods) are used to find out, which documents are similar and which are not.

- **Random indexing:** Such models usually start by creating random high-dimensional sparse vectors[12] that are unlikely to overlap and are assumed to be nearly orthogonal. Co-occurrence of these vectors are then recorded into the final matrix representing the meanings. These method take advantage from setting the dimension at the beginning.

- **Neural network:** In the last years, these models have become very popular. It is the human brain that defines semantics, so it is natural to use a neural network for the meaning extraction. The principles of the meaning extraction differ with the architecture of a neural network. Much work on improving the learning of word representations with using Neural Networks has been done, from feed-forward networks [13] to hierarchical models [14; 15] and recently recurrent neural networks [16]. In papers [17; 18] Mikolov examined existing word embeddings and showed that these representations already captured meaningful syntactic and semantic regularities such as the singular and plural relation that vectors $\boldsymbol{orange} - \boldsymbol{oranges} = \boldsymbol{plane} - \boldsymbol{planes}$. Read more in Section 4

2.3 Language Models

Language models are crucial in NLP, the backbone principle of language modeling is often used in distributional semantics models. The goal of language model is very simple, to estimate probability of any word occurrence

possible in the language. Even the task looks very easy, the satisfactory solution for natural language is very complicated.

2.3.1 Statistical Language Models

Let W denotes the word vocabulary. The $W^{\mathbb{N}}$ is the set of all combination of word sequences possible to create from the vocabulary W. Let

$$\mathcal{L} \subseteq W^{\mathbb{N}} \tag{2.1}$$

is a set of all possible word sequences in a language.

The sequence of words (i.e. sentence) can be expressed as

$$S = w_1^k = w_1, \cdots, w_k, \qquad S \in \mathcal{L}. \tag{2.2}$$

The language model tries to capture the regularities of a natural language by giving constraints on sequences S. These constraints can be either *deterministic* (some sequences are possible, some not) or *probabilistic* (some sequences are more probable than others).

Reason why we are talking about Language modeling is simple, the better the model represent language the better results we usually achieve solving our NLP problem (such as semantic understanding). Currently there is a massive research invested in language modeling, but often this time invested into bringing new representation is being outperformed by simple n-gram model and recently by simple recurrent neural network model [16]. In Chapter 3 we will show, that standard n-grams and many others language models with big mathematical background can be outperformed by Recursive Neural Network with memory.

Through the nature of problems mentioned above it is clear that we can only estimate words probabilities from the as large training data as possible. The probability estimation of $P(S)$ will be referred to $\tilde{P}(S)$ in the following text

$$P(S) \approx \tilde{P}(S). \tag{2.3}$$

By application of *chain rule* the probability $P(S)$ can be decomposed into the product of conditional probabilities

$$P(S) = P(w_1^k) = P(w_1)P(w_2|w_1)\cdots P(w_k|w_1^{k-1}) = \prod_{i=1}^{k} P\left(w_i|w_1^{i-1}\right). \tag{2.4}$$

2.3.2 N-gram Language Models

In previous Section the formula 2.4 shows that the probability of each word w_i is conditioned by complete history of words w_1^{i-1}. However, the problem is still the same. There is no way how to process all possible histories of words with all possible lengths k. The number of training parameters needed to be estimated rises exponentially with extending the history.

According to all problems mentioned above, truncating the word history is done to decrease the number of training parameters. It means, that the probability of word w_i is estimated only by $n-1$ preceding words (not by complete history)

$$P(S) = P(w_1^k) \approx \prod_{i=1}^{k} \tilde{P}\left(w_i | w_{i-n+1}^{i-1}\right). \tag{2.5}$$

These models are referred to as the *n-gram language models*. N-gram language models have been the most often used architecture for language modeling since a long time. N-grams, where $n = 1$, are called *unigrams*. The most often used are, however, *bigrams* ($n = 2$) and *trigrams* ($n = 3$).

Note that even for the trigram model of natural language, the number of training parameters is still enormous. For example, in the case of 65,536 words vocabulary, there is 2.8×10^{14} potential parameters to train. There will never be enough data for training all these parameters. Moreover, even if it could ever be, the storage for parameters and probability estimate retrieval time will not be satisfactory.

The lack of training data is sometimes referred to as the *data sparsity problem*.

2.3.3 Class-based N-gram Models

Class-based modeling is the most popular technique used for reducing the huge vocabulary-related sparseness of statistical language models [19]. Individual words are clustered into a much smaller number of classes. As a result, less data are required to train a robust class-based language model. Both manual and automatic word-clustering techniques are being used. Standalone class-based models usually perform poorly, which is the reason why they are usually combined with other models.

Let W denotes the set of possible words (word vocabulary) and C denotes a class vocabulary. Then we can define a mapping function $m : W \to C$, which maps every word $w_i \in W$ to some class $c_i \in C$.

The probability estimation of word w_i conditioned by its history w_{i-n+1}^{i-1}

(where n is the length of the n-gram) is given by the following formula

$$\tilde{P}\left(w_i|w_{i-n+1}^{i-1}\right) = \tilde{P}\left(w_i|c_i\right) \cdot \tilde{P}\left(c_i|c_{i-n+1}^{i-1}\right). \tag{2.6}$$

The probability estimate of the word occurrence given by its class is calculated as follows

$$\tilde{P}\left(w_i|c_i\right) = \frac{c\left(w_i, c_i\right)}{c\left(c_i\right)}, \tag{2.7}$$

where $c(w_i, c_i)$ is the number of times the word w_i is mapped to the class c_i over the frequency of class c_i.

Chapter 3

Neural Networks

Neural networks is a biologically-inspired programming paradigm which enables a computer to learn from observational data.

The simplest definition of a neural network, respective 'artificial' neural network (ANN), is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

3.1 Introduction

Architecture of neural networks is composed from neurons, layers and connections. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural nets adaptive to inputs and capable of learning. As an activation function of neurons that converts a neuron's weighted input to its output activation it is being commonly used *sigmoid* or *tanh* function, similarly to logistic regression (see section 3.2). More information about neurons (respective perceptrons) can be found in Section 3.3.

The main motivation is to simply come up with more precise way how to represent and model words, documents and language than the basic machine learning approaches. Like other machine learning methods – systems that learn from data – neural networks have been used to solve a wide variety of tasks, in this thesis we will however focus on NLP problems. There is nothing that neural networks can do in NLP that the basic machine learning techniques completely fail at, but in general neural networks and deep

learning currently provide the best solutions to many problems in NLP. We can benefit from those gains and see it as an evolution in machine learning.

3.2 Machine Learning

Machine learning explores the study and construction of algorithms that can learn from input data and make predictions on data. Such algorithms operate by building a model from the example data during a training phase. Inputs comes into algorithm in order to make data-driven predictions or decisions expressed as outputs. This is achieved by observing the properties from labeled training data, this learning technique is called supervised learning. Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from unlabeled data. Creating manually annotated dataset is generally a hard and time-consuming task. However most of current NLP problems are being solved based on annotated data sets which has been annotated by humans. Often such datasets have lack of data and together with features developed for NLP task often tend to be over-tuned for specific data set and fail to generalize in real approach.

While supervised learning, the acquired knowledge is later applied to determine the best category for the unseen testing dataset. For unsupervised learning there is no error or reward signal to evaluate potential solution, the goal is to model input data. Commonly used unsupervised learning algorithms are artificial neural network models, about which we will talk in next Chapter 3.

Machine learning techniques applied to NLP often uses n-gram language models introduced in previous Chapter, word clustering and basic bag-of-words representations as basic feature representation and further infer more complicated features. Bag-of-words model assumes that the document is a collection of words where the word order has no significance.

Basic machine learning technique how to perform classification is logistic regression (also commonly referred as Maximum Entropy classifier).

3.2.1 Logistic Regression Classifier

The Logistic regression classifier is based on the maximum entropy principle. The principle says that we are looking for a model which will satisfy all our constraints and at the same time resembles uniform distribution as much as possible. The logistic regression is a probabilistic model for binomial cases. Input is a vector of features, output is usually binary. Logistic classifier can be trained by stochastic gradient descent. The Maximum Entropy (MaxEnt) generalizes the same principle for multinomial cases.

We want a conditional probability:

$$p(y|x), \tag{3.1}$$

where y is the target class and x is vector of features.

The logistic regression follows binomial distribution. Thus, we can write following probability mass function:

$$p(y,x) = \begin{cases} h_\Theta(x), & \text{if y = 1,} \\ 1 - h_\Theta(x), & \text{if y= 0,} \end{cases} \tag{3.2}$$

where Θ is the vector of parameters $h_\Theta(x)$ is the hypothesis:

$$h_\Theta(x) = \frac{1}{1 + \exp(-\Theta^T x)} \tag{3.3}$$

The probability mass function can be rewritten as follows:

$$p(y|x) = (h_\Theta(x))^y (1 - h_\Theta(x))^{1-y} \tag{3.4}$$

We use maximum log-likelihood for N observations to estimate parameters:

$$\begin{aligned} l(\Theta) &= log \left[\prod_{n=1}^{N} (h_\Theta(x_n))^{y_n} (1 - h_\Theta(x_n))^{1-y_n} \right] \\ &= \sum_{n=1}^{N} [y_n \ log \ h_\Theta(x_n) + (1 - y_n) \ log \ (1 - h_\Theta(x_n))] \end{aligned} \tag{3.5}$$

Another basic concept is Naive Bayes classifier.

3.2.2 Naive Bayes Classifier

Naive Bayes (NB) classifier is a simple classifier commonly used as a baseline for many tasks. The model computes the posterior probability of a class based on the distribution of words in the given document as shown in equation 3.6, where s is the output label and x is the given document:

$$P(s|x) = \frac{P(x|s)P(s)}{P(x)} \tag{3.6}$$

$$\hat{s} = \underset{s \in S}{\text{argmax}} \ P(s) \prod_{x=1}^{n} P(x|s) \tag{3.7}$$

The NB classifier is described by equation 3.7, where $\hat{s}$ is the assigned output label. The NB classifier makes the decision based on the maximum a posteriori rule. In other words it picks the label that is the most probable.

3.2.3 SVM Classifier

Support vector machines till recent time was one of the most used classifier, is very similar to logistic regression. It is a vector space based machine learning method where the goal is to find a decision boundary between two classes that represents the maximum margin of separation in the training data [20].

SVM can construct a non-linear decision surface in the original feature space by mapping the data instances non-linearly to an inner product space where the classes can by separated linearly with a hyperplane.

Support Vector Machines

Following the original description [21] we describe the basic principle. We will assume only binary classifier for classes $y = -1, 1$ and linearly separable training set $\{(x_i, y_i)\}$, so that the conditions 3.8 are met.

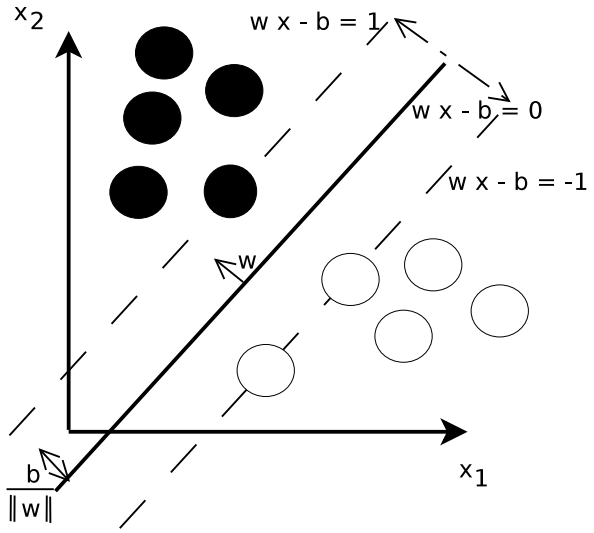


Figure 3.1: Optimal (and suboptimal) hyperplane.

$$\begin{array}{rcrrrr} \mathbf{w} \cdot \mathbf{x_i} + b & \leq & -1 & \text{if} & y_i = & -1 \\ \mathbf{w} \cdot \mathbf{x_i} + b & \geq & 1 & \text{if} & y_i = & 1 \end{array} \tag{3.8}$$

Equation 3.9 combines the conditions 3.8 into one set of inequalities.

$$y_i \cdot (\mathbf{w}_0 \cdot \mathbf{x} + b_0) \geq 1, \quad \forall i \tag{3.9}$$

SVM search the optimal hyperplane (equation 3.10) that separates both classes with the maximal margin. The formula 3.11 measures the distance between the classes in the direction given by $\mathbf{w}$:

$$\mathbf{w}_0 \cdot \mathbf{x} + b_0 = 0, \tag{3.10}$$

where $\mathbf{w}_0$ and b_0 are parameters of optimal hyperplane in feature space.

$$d(\mathbf{w}, b) = \min_{x;y=1} \frac{\mathbf{x} \cdot \mathbf{w}}{|\mathbf{w}|} - \max_{x;y=-1} \frac{\mathbf{x} \cdot \mathbf{w}}{|\mathbf{w}|} \tag{3.11}$$

The optimal hyperplane, expressed in equation 3.12, maximizes the distance $d(\mathbf{w}, b)$. Therefore the parameters $\mathbf{w}_0$ and b_0 can be found by maximizing $|\mathbf{w}_0|$. For better understanding see the optimal and suboptimal hyperplanes on figure 3.1.

$$d(\mathbf{w}_0, b_0) = \frac{2}{|\mathbf{w}_0|} \tag{3.12}$$

The classification is then just a simple decision on which side of the hyperplane the object is. Mathematically written as (3.13).

$$label(\mathbf{x}) = \text{sign}(\mathbf{w}_0 \cdot \mathbf{x} + b_0) \tag{3.13}$$

3.2.4 Clustering (Word Classes)

The goal of clustering is simple; to find an optimal grouping in a set of unlabeled data. In other words, similar words should share parameters which leads to generalization.

Example:

$$\begin{aligned} Class_1 &= \{yellow, green, blue, red\} \\ Class_2 &= \{Italy, Germany, France, Spain\} \end{aligned} \tag{3.14}$$

There are many ways how to compute the classes - usually, it is assumed that similar words appear in similar context. There are, however, two problems. Firstly, the optimality criterion must be defined. This criterion depends on the task that is being solved. The second problem is the complexity of the problem. The number of possible partitioning rises exponentially[1] with the number of elements in the set. It is therefore impossible to examine every

[1] To be exact, the number of possible partitioning of a n-element set is given by the Bell number, which is defined recursively: $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$.

possible partitioning of even a decently large set. The task is then to find a computationally feasible algorithm that would be as close to the optimal partitioning as possible.

Maximum Mutual Information Clustering

In [19] the MMI[2] clustering algorithm was introduced. The algorithm is based upon the principle of merging a pair of words into one class according to the minimal mutual information loss principle.

The algorithm gives very satisfactory results and it is completely unsupervised. This method of word clustering is possible only on very small corpora and is not suitable for large vocabulary applications.

K-means Clustering

K-means clustering [22] is a method commonly used for word clustering. It proceeds by selecting k initial cluster centers and then iteratively refining them as follows:

- Each instance d_i is assigned to its closest cluster center.
- Each cluster center C_j is updated to be the mean of its constituent instances.

The algorithm converges when there is no further change in assignment of instances to clusters.

3.3 Perceptron

The perceptron is a mathematical model of a biological neuron. While in actual neurons that receives electrical signals from the axons of other neurons, in the perceptron these electrical signals are represented as numerical values. We can think of perceptron model in artificial computer neural network as non-linear projections of data.

Without the non-linearity, it is not possible to model certain combinations of features such as Boolean XOR function 3.2.

Rosenblatt [23] proposed a simple rule to compute the output. He introduced weights, w1,w2,..., $w_i \in \mathbb{R}$ expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether

[2]Maximum Mutual Information

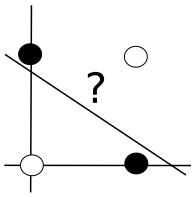


Figure 3.2: Boolean XOR function.

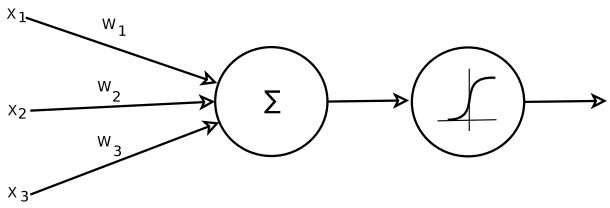


Figure 3.3: Neuron with three input synapses $\boldsymbol{x} = (x_1, x_2, x_3)$, $\boldsymbol{w}$ denotes to input weights. Sigmoid function is used as an non-linear activation function.

the weighted sum $\sum_j w_j x_j$ is less than or greater than some threshold value. Just like the weights, the $threshold \in \mathbb{R}$ is a parameter of the neuron. More formally:

$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq threshold. \\ 1, & \text{if } \sum_j w_j x_j > threshold. \end{cases} \tag{3.15}$$

3.4 Training of Neural Networks

The goal of any supervised learning algorithm is to find a function that best maps a set of inputs to its correct output. There are many ways how to train neural networks [24][25][26]. However, the most widely used and successful in practice is Stochastic Gradient Descent (SGD) [27].

Training of neural networks involves two stages, at first so called *forward pass* (respective forward propagation) it is being done:

3.4.1 Forward pass

- Input vector are presented at first in input layer.
- Forward propagation of a training takes input feature vector through

the neural network in order to generate the propagation's output activations. The target vector presents the desired output vector.

- While training we change weights that in another cycle, where the same input vector is presented, the output vector will be closer to the target vector.

In second stage it is so called *backpropagation* (respective "backward propagation of errors") performed. Backpropagation takes output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

3.4.2 Backpropagation

The backpropagation algorithm was originally introduced in the 1970s [28], but its importance was not fully appreciated for use in artificial neural networks until 1986 [27]. That paper describes neural networks where backpropagation works far faster than earlier approaches to learning and making it possible to use artificial neural networks to solve problems which were not solvable before.

Algorithm

Before we dive into a mathematical background and full explanation of backpropagation algorithm, will briefly introduce an algorithm. If we are going to use software library which will train our Neural Network, we will get familiar with α parameter which is necessary to set to train the network using SGD. To train the network, it is possible to compute gradient of the error. The gradients are sent back using the same weights that were used in the forward pass. Learning rate α controls how much we change the weights. For every iteration, making better prediction of our current neuron, we slightly change the weights and ensure that our change gives us a smaller error. However, if there is too little α value, it will result in long training time, too high value will erase previously learned patterns. With reasonably little α we are guaranteed to get a good approximation of some local minimum (though it might not be a global one). Usually there is being set a high learning rate at the beginning of training and reduce it during training.

Several training epochs over the training data are often performed, the training is usually finished when performance on held-out (development) data does not improve anymore. Held-out data are usually a small portion of training data and are used only for verification of the network performance - the network is not trained on these examples.

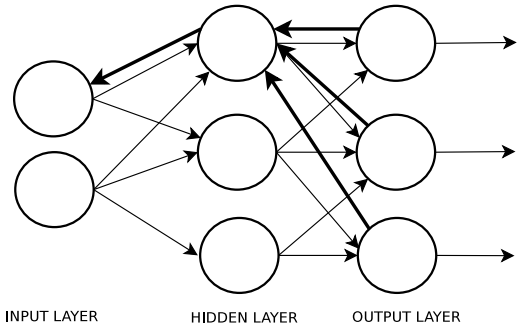


Figure 3.4: Backpropagation

Mathematical Background

Let us show an easy example of single neuron N (fig. 3.3), where $\boldsymbol{w} = (w_0, \ldots, w_k)$ and an input $\boldsymbol{x} = (x_0, \ldots, x_k)$. At the moment we will not consider the activation function σ of neuron N, f thus computes only the summation part of input weights and data $f = \sum_n w_i x_i$. Given set of training inputs x_j with labels y_j, we can compute the error of our neuron with following standard equation:

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_j (y_j - f(\boldsymbol{x_j}))^2 \tag{3.16}$$

E is a function of the weights $w_i \in \mathbb{R}$, which specifies the behavior of single neuron and denotes to jth input vector. We would like to find a global minimum of the error function E, for that we can use already mentioned standard gradient-descent algorithm [27]. We compute the gradient ∇E of E for our current set of weights:

$$\boldsymbol{w}_{current} = \boldsymbol{w}_{current} - \alpha \nabla E(\boldsymbol{w}_{current}), \tag{3.17}$$

where α is fixed parameter between 0 and 1 representing the "learning rate".

This gives vector which points in direction of steepest ascent of the function E. Error function will always have values greather than zero. If we substract some sufficiently small multiple of this vector from our current weight vector, we will be closer to a minimum of the error function:

$$\nabla E = \left(\frac{\partial E}{\partial w_0}, \ldots, \frac{\partial E}{\partial w_n} \right) \tag{3.18}$$

In each partial $\frac{\partial E}{\partial w_i}$ we consider each other variable beside w_i to be constant, and combining this with the chain rule gives:

$$\begin{aligned}\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_j (y_j - f(\boldsymbol{x_j}))^2 \\ &= \sum_j (y_j - f(\boldsymbol{x_j}))\frac{\partial}{\partial w_i}(y_j - f(\boldsymbol{x_j})) \\ &= \sum_j (y_j - f(\boldsymbol{x_j}))\left(-\frac{\partial f}{\partial w_i}\right) \\ &= -\sum_j (y_j - f(\boldsymbol{x_j}))\frac{\partial f}{\partial w_i}\end{aligned} \tag{3.19}$$

Because in the summation formula for f the w_i variable only shows up in the product $\frac{w_i}{x_{j,i}}$ (where $x_{j,i}$ is the i-th term of the vector $\boldsymbol{x}_j$), the last part expands as $x_{j,i}$. We also add our update rule, i.e. we have:

$$w_i = w_i + \alpha \sum_j (y_j - f(\boldsymbol{x_j}))x_{j,i} \tag{3.20}$$

There is an alternative form of an update rule that allows one to update the weights after each individual input is tested (as opposed to checking the outputs of the entire training set). This is called the stochastic update rule, and is given identically as above but without summing over all j:

$$w = w + \alpha(y_i - f(\boldsymbol{x_j}))\boldsymbol{x_j} \tag{3.21}$$

In last part we also add the activation function σ (we assume sigmoid function $\sigma = \frac{1}{1+e^{-x}}$) to the equation, sigmoid function satisfies the identity:

$$\sigma^{'}(x) = \sigma(x)(1 - \sigma(x)) \tag{3.22}$$

So instead of ∂f in the formula 3.19 above we need $\partial(\sigma \circ f)/\partial w_i$ and this requires the chain rule:

$$\frac{\partial E}{\partial w_i} = \sum_j (y_j + \sigma(f(\boldsymbol{x_j}))\sigma^{'}(f(\boldsymbol{x_j}))x_{j,i} \tag{3.23}$$

And using the identity for $\sigma^{'}$ gives us:

$$\frac{\partial E}{\partial w_i} = \sum_j (y_j + \sigma(f(\boldsymbol{x_j}))\sigma(f(\boldsymbol{x_j})(1 - \sigma(f(\boldsymbol{x_j})))x_{j,i} \tag{3.24}$$

There is a problem in training more than one layer neural network using the update rule above. We do not know what the "expected" output of any of the internal neurons in the graph are. In order to compute the error we need to know what the correct output should be, but we do not immediately have this information. We showed, how to backpropagate errors for one layer network, in reality we have two or more layer network and we have to backpropagate errors of inner neurons with proportion to all weighs [29], we end up with equation:

$$\boldsymbol{w} = \boldsymbol{w} + \alpha o_j(1 - o_j) \left(\sum_i w_{j,i}(y_i - o_i) \right) \boldsymbol{z} \tag{3.25}$$

where by $\boldsymbol{z}$ we denote the vector of inputs to the neuron – these may be the original input $\boldsymbol{x}$ if this neuron is the first in the network and all of the inputs are connected to it, or it may be the outputs of other neurons feeding into it, o_i is the output value $\sigma(f(\boldsymbol{x_i}))$.

3.4.3 Regularization

While network is being trained, it often overfits the training data, so it has good performance during training, but fails to generalize on *Test-data*. In section 3.4.2 we briefly talked about *Held-out data*, but we did not say, why to use them. Simple answer is that we are using them to setup the hyper-parameters - such as α, regularization parameters, cache for RNN and others. To understand why, consider that when setting hyper-parameters we are going to try many different choices for the hyper-parameters. If we set the hyper-parameters based on evaluations of the *Test-data* it is possible that we will end up overfitting our hyper-parameters to the *Test-data*. That is, we may end up finding hyper-parameters which fit particular *Test-data*, but where the performance of the network will not generalize to other data sets. We guard against that by figuring out the hyper-parameters using the *Held-out data*.

The network itself "memorizes" the training data, after training is finished, it will contain high weights that are used to model only some small subset of data.

We can try to force the weights to stay small during training to reduce this problem.

3.5 Feed-forward Neural Networks

A feedforward neural network is an artificial neural network where connections between the units do not form a cycle and the network can be seen on figure 3.5. This is different from recurrent neural networks introduced in followed section 3.7.

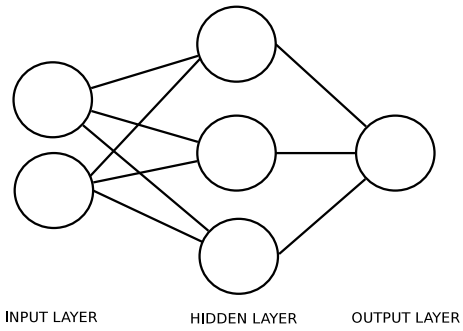


Figure 3.5: Feed-forward Neural Network

3.6 Convolutional Neural Networks

When we hear about Convolutional Neural Network (CNN), we typically think of Computer Vision. CNNs were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems [30; 31] today, from Facebook's automated photo tagging [32] to self-driving cars [33].

More recently we've also started to apply CNNs to problems in Natural Language Processing and gotten some interesting results.

Before we dive into exploring of convolutional neural networks in NLP, we will show how they works on simple image demonstration (see 3.6). What happens in parts of the image that are smooth, where a pixel color equals - the additions cancel and the resulting value approaches to 0, or black? If there's a sharp edge in intensity, a transition from white to black for example, you get a large difference and a resulting white value. This behavior can be used to detect object edges in an image, like it is on figure 3.7 using Sobel filter 3.26.

$$Mx = \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix} \quad \text{and} \quad My = \begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} \tag{3.26}$$

CNNs are basically just several layers of convolutions with nonlinear activa-

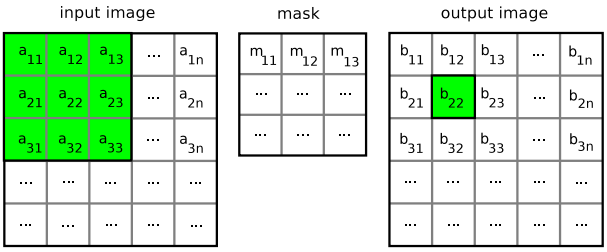


Figure 3.6: Convolution example using 3x3 filter, multiply its values element-wise with the original matrix, sum them up and slide the filter window over the whole matrix. $b_{22} = (a_{11} * m_{11}) + (a_{12} * m_{12}) + (a_{13} * m_{13}) + (a_{21} * m_{21}) + (a_{22} * m_{22}) + \ldots)$

(a) Input image.

(b) Output gray-scale image after convolution with Sobel mask.

Figure 3.7: Edge detection with Sobel mask.

tion functions like *tanh* applied to the results. In a traditional Feedforward Neural Network we connect each input neuron to each output neuron in the next layer. That is called a fully connected layer, or affine layer. In CNNs we do not do that. Instead, we use convolutions over the input layer to compute the output. This results in local connections, where each region of the input is connected to a neuron in the output. Each layer applies different filters, typically hundreds or thousands, and combines their results.

There is also something technique called pooling (subsampling) layers (see Section 3.6.2). During the training phase, a CNN automatically learns the values of its filters based on the task you want to perform. For example, in image classification a CNN may learn to detect edges from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features.

There are two aspects of this computation: Location Invariance and Compositionality. Because of sliding filters over the whole image we do not really care where our output object sits on image. In practice, pooling also gives invariance to translation, rotation and scaling. The second key aspect is (local) compositionality. Each filter composes a local patch of lower-level

features into higher-level representation. That is why CNNs are so powerful in Computer Vision. It makes intuitive sense that you build edges from pixels, shapes from edges, and more complex objects from shapes.

3.6.1 Hyperparameters

Architectures of Neural Networks other than Feed-forward Neural Network usually introduce additional parameters than just α parameter for training the network. We will explain traditional hyperparameters for CNN:

- *Narrow* or *wide convolution* (respective *zero-padding*). While sliding the filter, we add zero for values outside of a scope for our input matrix - means when we are with the center of the filter on edges of input matrix, that is called *wide convolution* and without using zero elements it is called *narrow convolution* (less computations).
- *Stride size* is another hyperparameter of CNN. Defining by how much you want to shift your filter at each step. In all the examples above the stride size was 1, and consecutive applications of the filter overlapped. A larger stride size leads to fewer applications of the filter and a smaller output size.
- *Pooling* parameter distinguish between max or average pooling (see 3.6.2).

3.6.2 Pooling Layers

Pooling layers are typically applied after the convolutional layers. *Pooling layers* subsample their input. The most common way to do pooling is to apply a *max* operation to the result of each filter. We do not necessarily need to pool over the complete matrix, we could also pool over a window. For example, the following figure 3.8 shows max pooling for a 2×2 window (in NLP we typically apply pooling over the complete output, yielding just a single number for each filter).

There are a couple of reasons to use pooling. One property of pooling is that it provides a fixed size of output matrix regardless of the size of used filters, which typically is required for classification. This allows us in NLP to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier.

Pooling also reduces the output dimensionality keeps the most "useful" information. We can think of each filter as detecting a specific feature, such as detecting if the sentence contains a negation like "not amazing" for example

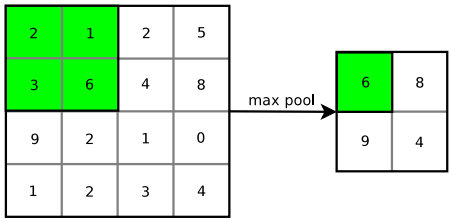


Figure 3.8: Max pooling with 2×2 filters and stride 2.

of sentiment analysis. If this phrase occurs somewhere in the sentence, the result of applying the filter to that region will yield a large value, but a small value in other regions. By performing the max operation you are keeping information about whether or not the feature appeared in the sentence, but you are losing information about where exactly it appeared. It is similar to what a bag-of-words model is doing. We are losing global information about locality (where in a sentence something happens), but we are keeping local information captured by your filters, like "not amazing" being very different from "amazing not". We distinguish between two types of pooling - max-pooling which takes maximum value as it is on figure 3.8, or avg-pooling which takes average number.

3.6.3 Channels

Channels are different "views" of the input data. For example, in image recognition you typically have RGB (red, green, blue) channels. We can apply convolutions across channels, either with different or equal weights. In NLP we could imagine having various channels as well: Separate channels for different word embeddings, or we could have a channel for the same sentence represented in different languages, or phrased in different ways.

3.6.4 Use of CNNs in NLP

Location Invariance and local Compositionality made intuitive sense for images, but not so much for NLP. We do care a lot where in the sentence a word appears. Pixels close to each other are likely to be semantically related (part of the same object), but the same is not always true for words. Clearly, words compose in some ways, like an adjective modifying a noun, but how exactly this works what higher level representations actually "mean" is not as obvious as in the Computer Vision case.

However, it turns out that CNNs applied to NLP problems perform quite well. The simple bag-of-words model is an obvious oversimplification with incorrect assumptions, but has nonetheless been the standard approach for

years and lead to pretty good results. Especially during recent years there is being dedicated a lot of attention to CNNs. A good start is [34] where authors evaluate different hyper parameter settings on various NLP problem. Article [35] evaluates CNNs on various classification NLP problems. In [36] they train CNN from scratch, without need for pre-trained word embeddings. Another use case of CNNs in NLP from Microsoft Research lab can be found in [37] and [38]. They describe how to learn semantically meaningful representations of sentences that can be used for Information Retrieval. The example given in the papers includes recommending potentially interesting documents to users based on what they are currently reading.

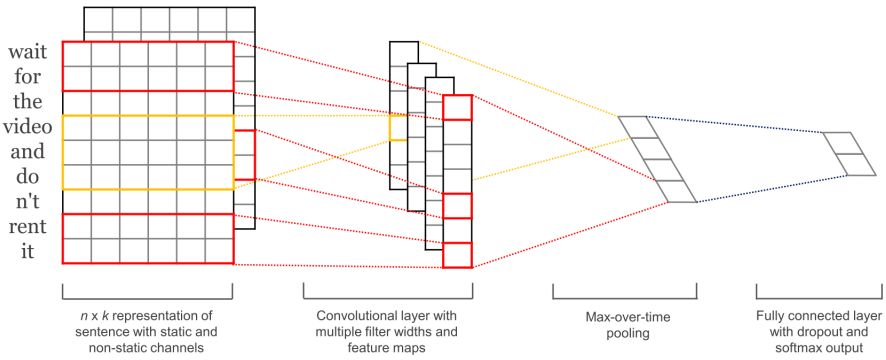


Figure 3.9: CNN for Sentence Classification, picture is taken from [35].

The input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix corresponds to one token, typically a word represented by word embeddings computed by Word2Vec/GloVe, but it could be a character. That is, each row is vector that represents a word. Typically, these vectors are word embeddings (low-dimensional representations) like Word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary. For a 10 word sentence using a 100-dimensional embedding we would have a 10x100 matrix as our input. That is our "image". In NLP we typically use filters that slide over full rows of the matrix (words). Thus, the "width" of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical.

A big argument for use of CNNs is that they are fast to train. CNNs are also efficient in terms of representation. With a large vocabulary, computing anything more than 3-grams can quickly become expensive. Even Google does not provide anything beyond 5-grams. Convolutional Filters learn good representations automatically, without needing to represent the whole vocabulary. It's completely reasonable to have filters of size larger

than 5.

3.7 Recurrent Neural Networks

Popular in NLP due to their capability for processing arbitrary length sequences. The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that is an ideal idea, especially in NLP tasks. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps, because it is also often claimed that learning long-term dependencies by stochastic gradient descent can be difficult [39].

For Language modeling [16] there is being used a so called *simple recurrent neural network* (see figure 3.10) or Elman network [40].

Notation:

- $\boldsymbol{x_t}$ is the input at time step t. For example for language modeling, $\boldsymbol{x_1}$ could be seen as a vector corresponding to the second word of a sentence.
- $\boldsymbol{hs_t}$ is the hidden state at time step t. It is the networks "memory" (captures information about what happened in all the previous time steps0) and it is calculated based on the previous hidden state and the input at the current step: $hs_t = f(Ux_t + Whs_{t-1})$, where the f is usually our well known nonlinearity function such as tanh. hs_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.
- $\boldsymbol{y_t}$ is the output at step t. For example, if we wanted to predict the next word in a sentence, it would be a vector of probabilities across our vocabulary, $\boldsymbol{y_t} = softmax(Vhs_t)$. Output is calculated based on the memory at time t, but it is more complicated in practice, because hs_t can not capture information from too many time steps ago (explanation in section 3.7.1). $Softmax$ regression is a probabilistic method with function similar to the Logistic regresion, we use the softmax function to map inputs to the the predictions (can be multinomial).

- U and W are parameters of RNN that are shared across the whole network and are not different at each layer as it is for example in Feed-forward Neural Networks and its weight parameters.

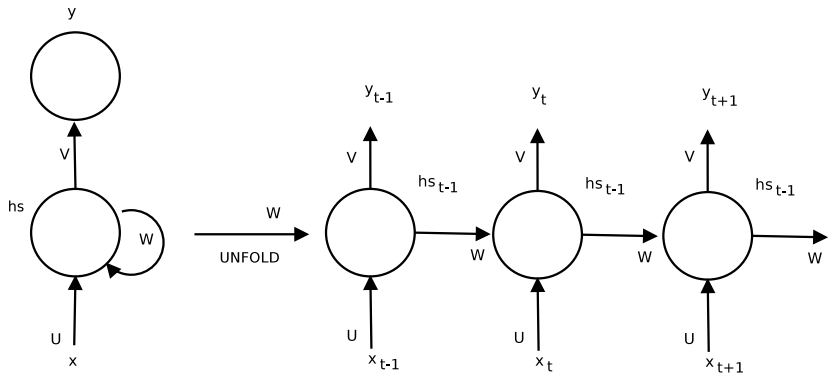


Figure 3.10: Picture shows a RNN being unrolled (or unfolded) into a full network. By unrolling we simply mean that we write out the network for the complete sequence. For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.

3.7.1 RNNs with Long Short-Term Memory

Long Short-Term Memory (LSTM) units [41] have re-emerged as a popular architecture due to their representational power and effectiveness at capturing long-term dependencies. LSTMs do not have a fundamentally different architecture from RNNs, but they use a different function to compute the hidden state.

The memory in LSTMs are called cells and they take as input the previous state hs_{t-1} and current input x_t. Internaly these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input.

In a traditional RNN, during the gradient back-propagation phase, the gradient signal can end up being multiplied a large number of times (as many as the number of timesteps) by the weight matrix associated with the connections between the neurons of the recurrent hidden layer. This means that the magnitude of weights in the transition matrix can have a strong impact on the learning process.

When the weights in this matrix are small (if the leading eigenvalue of the weight matrix is smaller than 1), it can lead to a situation called vanishing gradients [39] where the gradient signal gets so small that learning either

becomes very slow or stops working altogether. It can also make more difficult the task of learning long-term dependencies in the data. Conversely, if the weights in this matrix are large (or, again, more formally, if the leading eigenvalue of the weight matrix is larger than 1), it can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as exploding gradients.

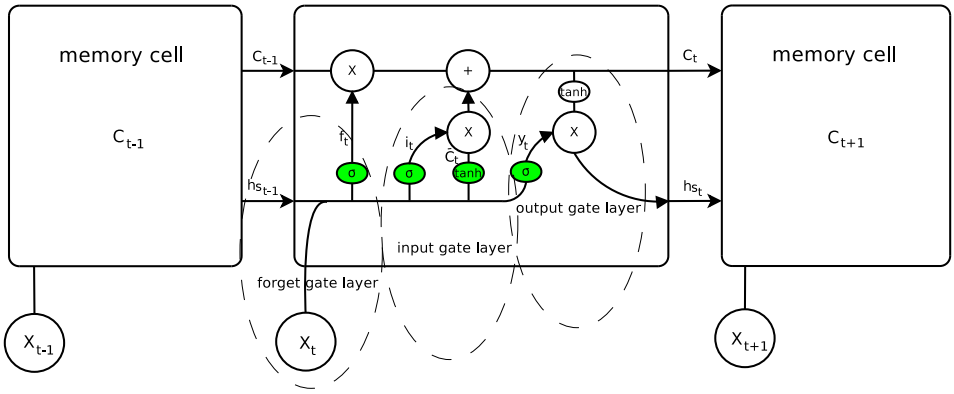


Figure 3.11: LSTM memory cell. Green boxes represent learned neural network layers, while circles inside a cel represents pointwise operations.

These issues are the main motivation behind the LSTM model which introduces a new structure called a *memory cell* (fig. 3.11). Cells take as input the previous state h_{t-1} and current input x_t. Internally these cells decide what to keep in (and what to erase from) memory. They then combine the previous state, the current memory, and the input. Forget gate makes decision what information we are going to throw away from the cell state. Input gate layer decides which values we will update (which information we keep). It has turns out that these types of units are very efficient at capturing long-term dependencies.

Mathematical Background

Notation:

- $\boldsymbol{x_t}$ is the input to the memory cell layer at time t
- $\boldsymbol{hs_t}$ is the network state at time t
- W_i, W_f, W_c, W_o are weight matrices
- b_i, b_f, b_c and b_o are bias vectors

First, we compute the value for f_t, the activation of the memory cells forget gates at time t, we need to decide what information we are going to throw

away from the cell state. It looks at hs_{t-1} and x_t, outputs a number between 0 and 1[3] for each number in the cell state C_{t-1}. For instance of the language model which is trying to predict next word based on all the previous ones, cell state might include the gender of a present subject, so that the correct pronouns can be used. But after few words in a sentence we might get to subordinate clause talking about a new object and want to forget the gender of the old one:

$$f_t = \sigma(W_f \cdot [hs_{t-1}, x_t] + b_f) \tag{3.27}$$

Second, we compute the values for i_t, the input gate, and $\widetilde{C_t}$ the candidate value for the states of the memory cells that could be added at time t. We want to decide what new information we are going to store in the cell state. *Input gate layer* decides which values we will update. In the next step, we will combine these two to create update to the state. For instance of the language model again, we would like to add the gender of the new subject to the cell state replacing the old one.

$$i_t = \sigma(W_i \cdot [hs_{t-1}, x_t] + b_i) \tag{3.28}$$

$$\widetilde{C_t} = tanh(W_c \cdot [hs_{t-1}, x_t] + b_c) \tag{3.29}$$

Given the value of the input gate activation i_t, the forget gate activation f_t and the candidate state value $\widetilde{C_t}$, we can compute C_t the memory cells' new state at time t. We multiply the old state by f_t, forgetting the things we decided to forget and we add $i_t \cdot \widetilde{C_t}$. For the case of the language model, this is where we actually drop the information about subject gender and add the new information.

$$C_t = i_t \cdot \widetilde{C_t} + f_t \cdot C_{t-1} \tag{3.30}$$

With the new state of the memory cells, we can compute the value of their output gates and their outputs. For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case it is coming next. For example it might output whether the subject is singular or plural, so that we knot what for of a verb should be following.

$$y_t = \sigma(W_o \cdot [hs_{t-1}, x_t] + b_o) \tag{3.31}$$

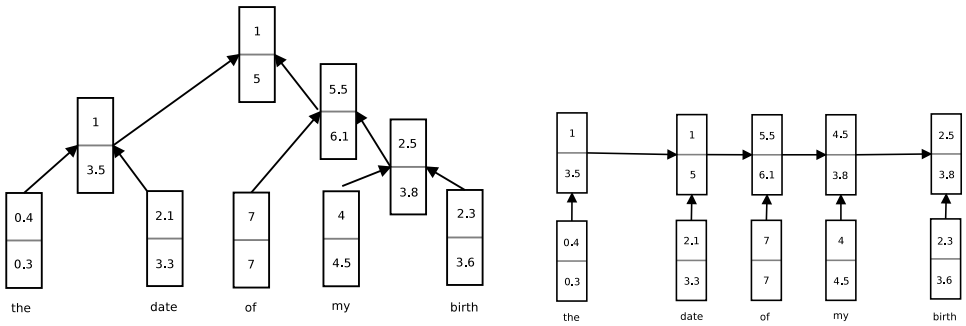
$$h_t = y_t \cdot tanh(C_t) \tag{3.32}$$

[3]One for strongly remember, zero to not remembering the information.

There exists a lot of variants of LSTM models [42; 43; 44], for instance one of recent variation called Gated Recurrent Unit [43] combines the forget and input gates into a single "update gate". It also merges the cell state and hidden state, and makes some other changes. However the differences are minor and as has been investigated, usually does not perform better [45; 46].

3.8 Recursive Neural Network

There is another special type of Reccurent Neural Network, it is called *Recursive Neural Network*. A Recursive Neural Network is more like a hierarchical network where there is really no time aspect to the input sequence, but the input has to be processed hierarchically in a tree fashion. Example of how a recursive neural network looks can be seen at figure 3.12. It shows the way to learn a parse tree of a sentence by recursively taking the output of the operation performed on a smaller chunk of the text. Recursive Neural Networks operate on any hierarchical structure, combining child representations into parent representations, Recurrent Neural Networks operate on the linear progression of time, combining the previous time step and a hidden representation into the representation for a current time step.



(a) Recursive Neural Network

(b) Recurrent Neural Network

Figure 3.12: Sentence representation with Recurrent and Recursive Neural Network.

3.9 Deep Learning

Deep learning algorithms attempt to learn multiple levels of representation of increasing complexity (respective abstraction of the problem). Most current machine learning techniques requires human-designed representations and input features. Standard Machine learning approaches just optimize the

weights to produce best final prediction. Machine Learning methods are heavily dependent on quality of input features created by human.

Deep belief networks (DBNs), Markov Random Fields with multiple layers, various types of multiple-layer neural networks are techniques which has more than one hidden layer and are able to model complex non-linear problems. Deep architectures can represent certain families of functions more efficiently (and with better scaling properties) than shallow ones, but the associated loss functions are almost always non convex. Deep learning is practically putting back together representation learning with machine learning. It tries to learn a good features, across multiple levels of increasing complexity and abstraction (hidden layers) of the problem [47].

Hidden layer represents learned non-linear combination of input features. With hidden layer, we can solve non-linear problem (such as XOR):

- Some neurons in the hidden layer will activate only for some combination of input features.
- The output layer can represent combination of the activations of the hidden neurons.

Neural network with one hidden layer is *universal approximator*. The *universal approximator* theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated arbitrarily closely by a multi-layer perceptron with just one hidden layer. However, not all functions can be represented efficiently with a single hidden layer. That is why deep learning architectures can achieve better accuracy for complex problems.

In recent work, there are being used deep LSTM networks, often bidirectional deep recurrent (LSTM) network [48]. Bidirectional RNNs are based on the idea that the output at time t may not only depend on the previous elements in the sequence, but also future elements. For example, to predict a missing word in a sequence you want to look at both the left and the right context. Bidirectional RNNs are quite simple. They are just two RNNs stacked on top of each other. The output is then computed based on the hidden state of both RNNs. Deep (Bidirectional) RNNs are similar to Bidirectional RNNs, only that we now have multiple layers per time step. In practice this gives already mentioned higher learning capacity (but we also need a lot of training data).

3.9.1 Representations and Features Learning Process

Development of good features is hard and time-consuming process. Features are eventually over-specified and incomplete anyway. In NLP research we

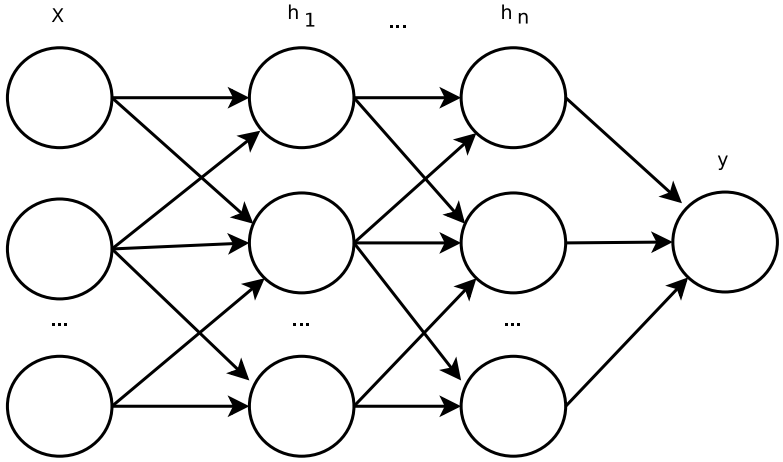


Figure 3.13: Deep neural network. X represents the input layer, $h_1, h_2, \ldots h_n$ represents hidden layers and y denotes to output layer

usually after some time can find and tune features for manually annotated corpus dealing with some NLP problem. However problem is, that in real situation we will often find that developed features were over specified for concrete corpus and fail in generalization during a real usage.

If machine learning could learn features automatically, the learning process could be automated more easily and more task could be solved and there deep learning provides one way of automated feature learning.

Chapter 4

Distributional Semantics Using Neural Networks

Many models in NLP are based on counts over words such as Probabilistic Context Free Grammars (PCFG) [49]. In those approaches it can hurt generalization performance when specific words during testing were not present in the training set. Because an index vector over a large vocabulary is very sparse, models tends to overfit to the training data. The classical solutions to the problem involve in already mentioned time consuming manual engineering of complex features. Deep learning models of language usually use distributed representation (see 2.1.1). Methods for learning word representations where meaning of words or phrases is represented by vectors of real numbers, where the vector reflects the contextual information of a word across the training corpus.

It was shown that the word vectors can be used for significant improving and simplifying of many NLP applications [50; 51]. There are also NLP applications, where word embeddings does not help much [52].

There has been introduced several methods based on the feed-forward NNLP (Neural Network Language Model) in recent studies. One of the Neural Network based models for word vector representation which outperforms previous methods on word similarity tasks was introduced in [53]. The word representations computed using NNLP are interesting, because trained vectors encode many linguistic properties and those properties can be expressed as linear combinations of such vectors.

Nowadays, word embedding methods Word2Vec [17] and GloVe [11] significantly outperform other methods for word embeddings. Word representations made by these methods have been successfully adapted on variety of core NLP task such as Named Entity Recognition [54; 55], Part-of-speech Tagging [56], Sentiment Analysis [57], and others.

There are also neural translation-based models for word embeddings [43; 58] that generates an appropriate sentence in target language given sentence in source language, while they learn distinct sets of embeddings for the vocabularies in both languages. Comparison between monolingual and translation-based models can be found in [59].

We will shortly introduce current state-of-the-art Word Embedding methods called Word2Vec[17] and other methods for sentence representation in following sections.

Vector Similarity Metrics

The distance (similarity) between two words can be calculated by a vector similarity function. Let $\boldsymbol{a}$ and $\boldsymbol{b}$ denote the two vectors to be compared and $S(\boldsymbol{a}, \boldsymbol{ab})$ denote their similarity measure. Such a metric needs to be symmetric: $S(\boldsymbol{a}, \boldsymbol{b}) = S(\boldsymbol{b}, \boldsymbol{a})$.

There are many methods to compare two vectors in a multi-dimensional vector space. Probably the simplest vector similarity metrics are the familiar Euclidean ($r = 2$) and city-block ($r = 1$) metrics

$$S_{mink}(\boldsymbol{a}, \boldsymbol{b}) = \sqrt[r]{\sum |a_i - b_i|^r}, \tag{4.1}$$

that come from the Minkowski family of distance metrics.

Another often used metric characterizes the similarity between two vectors as the cosine of the angle between them. The cosine similarity is defined as follows:

$$S_{\cos}(\boldsymbol{a}, \boldsymbol{b}) = \cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\|} = \frac{\sum a_i b_i}{\sqrt{\sum a_i^2 \sum b_i^2}}. \tag{4.2}$$

4.0.2 Continuous Bag-of-Words

Continuous Bag-of-Words (CBOW) [17] tries to predict the current word according to the small context window around the word. The architecture is similar to the feed-forward Neural Network Language Model (NNLM) which has been proposed in [60]. The NNLM is computationally expensive between the projection and the hidden layer. Thus, CBOW proposed architecture, where the (non-linear) hidden layer is removed (or in reality is just linear) and projection layer is shared between all words. The word order in the context does not influence the projection (see Figure 4.1a). This architecture also proved low computational complexity.

(a) CBOW (b) Skip-gram

Figure 4.1: Nerual network models architectures.

4.0.3 Skip-gram

Skip-gram architecture is similar to CBOW. Although instead of predicting the current word based on the context, it tries to predict a words context based on the word itself [18]. Thus, intention of the Skip-gram model is to find word patterns that are useful for predicting the surrounding words within a certain range in a sentence (see Figure 4.1b). Skip-gram model estimates the syntactic properties of words slightly worse than the CBOW model, but it is much better for modeling the word semantics on English test set [17; 18]. Training of the Skipgram model does not involve dense matrix multiplications 4.1b and that makes training also extremely efficient [18].

4.0.4 Paragraph Vectors

It does not make sense to treat phrases such as Czech Republic, or longer sentences as separate words. Paragraph vectors were proposed in [61] as an unsupervised method of learning text representation. Article shows the way how to compute vector for the whole paragraphs, documents or sentences. Resulting feature vector has fixed dimension while the input text can be of any length. The paragraph vectors and word vectors are concatenated to predict the next word in a context. The paragraph token acts as a memory that remembers what information is missing from the current context.

The sentence representations can be further used in classifiers (logistic regression, SVM or NN).

4.0.5 Tree-based LSTM

Tree-structured representation of LSTM was presented in [48].Tree model represents the sentence structure. Dependency parsing is being used as typical sentence tree structure representation [62]. LSTM processes input sentences of variable length via recursive application of a transition function on a hidden state vector hs_t. Such model was used for sentiment analysis or given these representations, model predicts the similarity score using a neural network considering distance and angle between vectors representing sentences - i.e. for each sentence pair it creates sentence representations h_L and h_R using Tree-LSTM model.

Chapter 5

Preliminary Results

This Chapter describes preliminary ideas for future work that imply the aims of PhD thesis. The three main directions are indicated:

Much has been investigated about word embeddings of English words and phrases, but only little attention has been dedicated to other languages. In [63] we explore the behavior of state-of-the-art word embedding methods on Czech that is a representative of Slavic languages with rich word morphology. These languages are highly inflected and have a relatively free word order. Czech has seven cases and three genders. The word order is very variable from the syntactic point of view: words in a sentence can usually be ordered in several ways, each carrying a slightly different meaning. All these properties complicate the learning of word embeddings. We introduced new corpus for word analogy task that inspects syntactic, morphosyntactic and semantic properties of Czech words and phrases. We experimented with Word2Vec and GloVe algorithms and discussed the results on this corpus. We showed that while current methods can capture semantics on English at similar corpus with 76% of accuracy, there is still room for improvement of current methods on highly inflected languages where the models work on less than 38%.

In [64] we present our UWB[1] system for Semantic Textual Similarity (STS). Given two sentences, the system estimates the degree of their semantic similarity. In the monolingual task, our system achieve mean Pearson correlation 75.7% compared with human annotators. Our system is ranked second among 119 systems. In the cross-lingual task, our system has correlation 86.3% and is ranked first among 26 systems. It was shown, how good can simple Tree LSTM Neural Network architecture perform representing a meaning and was compared with complex state-of-the-art algorithms for the meaning representation. We also experimented with Paragraphs vector

[1] University of West Bohemia

models and linear combination of word vectors (CBOW model) representing the sentence.

Clustering of word vectors and Paragraphs vector models has showed significance improvement in Sentiment analysis at SemEval2016 competition in [65] and also in recent work targeted on Czech language [66]. Neural network based Word Embedding models has helped with accuracy of previous model originally developed for SemEval2014 competition [67] to get into first position on several tasks during competition of year 2016.

Chapter 6

Summary and Future Work

This thesis presents overview of the current state-of-the-art approaches for distributional semantics.

The performance of semantic representation models has rapidly improved during recent years with use of neural network and deep learning. As a future work we choose to reach a hard target. We will try to outperform current state-of-the-art word embedding methods. We are going to experiment with use of external sources of information (such as part-of-speech tags, NER, or stemming and character n-grams) during training process of current state-of-the-art neural network based word embedding methods and explore results on highly inflected languages. We will explore further use of neural network in solving NLP problems where the semantic knowledge is crucial to solve the actual problem. Currently, we are working on article for Paraphrase detection where we are going to use neural network based approach. This thesis is a theoretical preparation for the future development.

6.1 Overall Aims of the PhD Thesis

The goal of doctoral thesis is to explore state-of-the-art distributional semantics models based on neural networks and to propose novel approaches to improve these models on inflectional languages. The work will be focused on the following research tasks:

- Study the influence of rich morphology on the quality of meaning representation.
- Propose of novel approaches based on neural networks for improving the meaning representation of inflectional languages.
- Use of distributional semantic models for improving NLP tasks.

Bibliography

[1] Z. Harris, "Distributional structure," *Word*, vol. 10, no. 23, pp. 146–162, 1954.

[2] J. R. Firth, "A Synopsis of Linguistic Theory, 1930-1955," *Studies in Linguistic Analysis*, pp. 1–32, 1957.

[3] H. Rubenstein and J. B. Goodenough, "Contextual correlates of synonymy," *Communications of the ACM*, vol. 8, pp. 627–633, Oct. 1965.

[4] W. G. Charles, "Contextual correlates of meaning," *Applied Psycholinguistics*, vol. 21, no. 04, pp. 505–524, 2000.

[5] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[6] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis.," *Journal of the American Society for Information Science 41*, pp. 391–407, 1990.

[7] T. Hofmann, "Probabilistic latent semantic analysis," in *Proceedings of 15th Conference on Uncertainty in Artificial Intelligence*, pp. 289–296, 1999.

[8] D. M. Blei, A. Y. Ng, M. I. Jordan, and J. Lafferty, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, p. 2003, 2003.

[9] B. Riordan and M. N. Jones, "Redundancy in perceptual and linguistic experience: Comparing feature-based and distributional models of semantic representation," *Topics in Cognitive Science*, vol. 3, no. 2, pp. 303–345, 2011.

[10] D. S. McNamara, "Computational methods to extract meaning from text and advance theories of human cognition," *Topics in Cognitive Science*, vol. 3, no. 1, pp. 3–17, 2011.

[11] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[12] M. Sahlgren, "An Introduction to Random Indexing," *Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, 2005.

[13] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, pp. 1137–1155, Mar. 2003.

[14] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model.," in *Aistats*, vol. 5, pp. 246–252, Citeseer, 2005.

[15] A. Mnih and G. E. Hinton, "A scalable hierarchical distributed language model," in *Advances in neural information processing systems*, pp. 1081–1088, 2009.

[16] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent neural network based language model.," in *INTERSPEECH*, vol. 2, p. 3, 2010.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, pp. 3111–3119, 2013.

[19] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational Linguistics*, vol. 18, pp. 467–479, 1992.

[20] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[21] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, pp. 273–297, Sept. 1995.

[22] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, (Berkeley, Calif.), pp. 281–297, University of California Press, 1967.

[23] J. Rosenblatt, "Testing approximate hypotheses in the composite case," *Ann. Math. Statist.*, vol. 33, pp. 1356–1364, 12 1962.

[24] R. S. Scalero and N. Tepedelenlioglu, "A fast new algorithm for training feedforward neural networks," *Signal Processing, IEEE Transactions on*, vol. 40, no. 1, pp. 202–210, 1992.

[25] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 989–993, 1994.

[26] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms.," in *IJCAI*, vol. 89, pp. 762–767, 1989.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[28] H. J. Kelley, "Gradient theory of optimal flight paths," *Ars Journal*, vol. 30, no. 10, pp. 947–954, 1960.

[29] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks, 1989. IJCNN., International Joint Conference on*, pp. 593–605, IEEE, 1989.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[31] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *Neural Networks, IEEE Transactions on*, vol. 8, no. 1, pp. 98–113, 1997.

[32] S. S. Farfade, M. J. Saberian, and L. Li, "Multi-view face detection using deep convolutional neural networks," *CoRR*, vol. abs/1502.02766, 2015.

[33] Y. Bengio, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

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

[35] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014.

[36] R. Johnson and T. Zhang, "Effective use of word order for text categorization with convolutional neural networks," *CoRR*, vol. abs/1412.1058, 2014.

[37] J. Gao, L. Deng, M. Gamon, X. He, and P. Pantel, "Modeling interestingness with deep neural networks," Dec. 17 2015. US Patent 20,150,363,688.

[38] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pp. 101–110, ACM, 2014.

[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[40] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[41] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[42] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3, pp. 189–194, IEEE, 2000.

[43] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[44] K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, "Depth-gated lstm," *arXiv preprint arXiv:1508.03790*, 2015.

[45] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *arXiv preprint arXiv:1503.04069*, 2015.

[46] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342–2350, 2015.

[47] Y. Bengio, Y. LeCun, *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, 2007.

[48] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks," *CoRR*, vol. abs/1503.00075, 2015.

[49] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*, vol. 999. MIT Press, 1999.

[50] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," 2008.

[51] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *CoRR*, vol. abs/1103.0398, 2011.

[52] J. Andreas and D. Klein, "How much do word embeddings encode about syntax?," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, (Baltimore, Maryland), pp. 822–827, Association for Computational Linguistics, June 2014.

[53] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, (Stroudsburg, PA, USA), pp. 873–882, Association for Computational Linguistics, 2012.

[54] S. K. Siencnik, "Adapting word2vec to named entity recognition," in *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, pp. 239–243, 2015.

[55] H. Demir and A. Ozgur, "Improving named entity recognition for morphologically rich languages using word embeddings," in *Machine Learning and Applications (ICMLA), 2014 13th International Conference on*, pp. 117–122, IEEE, 2014.

[56] R. Al-Rfou, B. Perozzi, and S. Skiena, "Polyglot: Distributed word representations for multilingual nlp," 2013.

[57] M. Pontiki, D. Galanis, H. Papageorgiou, S. Manandhar, and I. Androutsopoulos, "Semeval-2015 task 12: Aspect based sentiment analysis," in *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015), Association for Computational Linguistics, Denver, Colorado*, pp. 486–495, 2015.

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

[59] F. Hill, K. Cho, S. Jean, C. Devin, and Y. Bengio, "Not all neural embeddings are born equal," *CoRR*, vol. abs/1410.0718, 2014.

[60] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain, "Neural probabilistic language models," in *Innovations in Machine Learning*, pp. 137–186, Springer, 2006.

[61] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pp. 1188–1196, 2014.

[62] M.-C. De Marneffe, B. MacCartney, C. D. Manning, *et al.*, "Generating typed dependency parses from phrase structure parses," in *Proceedings of LREC*, vol. 6, pp. 449–454, 2006.

[63] L. Svoboda and T. Brychcín, *New word analogy corpus for exploring embeddings of Czech words*. Cham: Springer International Publishing, 2016.

[64] T. Brychcín and L. Svoboda, "Uwb at semeval-2016 task 1: Semantic textual similarity using lexical, syntactic, and semantic information.," *In Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016), San Diego, California, June*, vol. 16, 2016.

[65] T. Hercig, T. Brychcín, L. Svoboda, and M. Konkol, "Uwb at semeval-2016 task 5: Aspect based sentiment analysis," in *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, (San Diego, California), pp. 354–361, Association for Computational Linguistics, June 2016.

[66] T. Hercig, T. Brychcín, L. Svoboda, M. Konkol, and J. Steinberger, "Unsupervised methods to improve aspect-based sentiment analysis in czech," *Computación y Sistemas*, in press.

[67] T. Brychcın, M. Konkol, and J. Steinberger, "Uwb: machine learning approach to aspect-based sentiment analysis," in *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pp. 817–822, 2014.