FACULTY
OF APPLIED SCIENCES
UNIVERSITY
OF WEST BOHEMIA

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Cross-Lingual Methods for Semantic Representations

The State of the Art and the Concept of Ph.D. Thesis

Ondřej Pražák

Technical Report No. DCSE/TR-2020-06
September 2020
Distribution: public

Technical Report No. DCSE/TR-2020-06
September 2020

Cross-Lingual Methods for Semantic Representations

The State of the Art and the Concept of Ph.D. Thesis

Ondřej Pražák

Abstract

Semantic analysis is the elementary task of Natural Language Processing. Nowadays, there are many outstanding semantic models based on deep learning for English and other high-resource languages. However, for languages with less data available, these methods reach their limits. This work summarizes recent work in Deep Learning, methods for creating semantic representation and transferring these representations between languages.

This work was supported by Grant No. SGS-2019-018 Processing of heterogeneous data and its specialized applications.

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

University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
30614 Plzeň
Czech Republic

Contents

Chapter 1

Introduction

Semantic analysis (representing the meaning of texts) is one of the elementary tasks of Natural Language Processing (NLP), which is very useful across almost all NLP tasks. Generally, the task is about encoding the meaning of the language (mostly text) in a machine-readable or machine-understandable format. For creating the formal representation of semantics automatically, a high amount of labeled data is needed for training. Unfortunately, there is not enough data for supervised training for many languages. That gave the motivation to create methods that do not need any training data (unsupervised methods) and methods that can use training data in one language to train the model for a different language (Cross-lingual methods). This work focuses on cross-lingual methods of semantic representations.

1.1 Outline

The structure of this report is as follows. The second chapter describes general methods of machine learning, which are often used in Natural language processing tasks. The most attention is paid to new neural network architectures, which are used in the state of the art methods for semantic representations.

The third chapter covers semantic analysis from basic techniques to the current state-of-the-art methods. Chapter 4 deals with cross-lingual methods, the methods where we can use a single model across more languages. Chapter 5 concludes the report and presents the aims of the Ph.D. thesis.

Chapter 2

Machine Learning

This chapter describes the basic principles of some historically used methods for learning semantics and then the current state-of-the-art methods based on artificial neural networks in detail.

2.1 Feature Engineering and Simple Classifiers

Before the rise of the popularity of the deep-learning methods, machine learning in NLP had been done by feature engineering with simple classifiers such as SVM or Maximum-Entropy. That means the developer of the learning algorithm had to manually select and extract features that he considered helpful for the specific task. The classifier only learns a score for every feature (how does the feature value affect the output).

2.1.1 Supervised Machine Learning

The supervised learning is formally the function $\hat{y} = d(x, \Theta)$, where Θ is a vector of parameters to be learned, and , $\hat{y} \in Y \subset \mathbb{N}$ for classification, and $\hat{y} \in \mathbb{R}$ for regression. The learning procedure can be formalized as: $\text{argmin}_{\Theta} J(d(X, \Theta), Y)$, where J is the cost function, and Y is the vector of true classes for the examples in X.

2.1.2 Gradient-Based Optimizers

Most of the optimization techniques used in machine learning are based on the gradient descent. In the gradient descent, we initialize the model parameters randomly, and then we iteratively move the parameters in the opposite direction of the gradient of the cost function with respect to the parameters Θ.

2.1.3 Linear Regression

In the simplest case of linear regression, the task is to predict a continuous value based on another one (a single feature). We want to find a linear dependence between those two values.

A basic machine learning method is the least-squares optimization, which is probably the most common method for simple linear regression. In the least-squares optimization, we want to find the weight of each feature so that the sum of squares of the error is minimal. More formally, we want to find:

$$\underset{\Theta}{\operatorname{argmin}} \sum_{i} (x_i \Theta_i^T - y_i)^2 \tag{2.1}$$

2.1.4 Logistic Regression

In logistic regression, we change the linear regression model so that its output can be interpreted as a probability. We can then use such a model for classification. We want the output to be in $< 0, 1 >$ symmetric around functional value 0.5, and we want the value to change more near the decision boundary. Previous requirements lead to the sigmoid function for binary classification (see Figure 2.1). There is one more problem. When a datapoint is misclassified, the gradient is decreasing with increasing distance from the decision boundary, which leads to the non-convex cost function. We can fix this by using the cross-entropy cost function:

$$J(\Theta) = \sum_{i=0}^{m} cost(x_i, y_i, \Theta) \tag{2.2}$$

$$cost_\Theta(x, y, \Theta) = \begin{cases} -\log\left(\frac{1}{1+e^{-X\Theta}}\right), & \text{if } y = 1 \\ -\log\left(1 - \frac{1}{1+e^{-X\Theta}}\right) & \text{if } y = 0 \end{cases}$$

With the cross-entropy cost, the cost function with respect to the inputs and/or weights looks like it is shown in Figure 2.2, where $cost_1$ is cost value in the case there the true class is *1*, and $cost_0$ for the true class *0*.

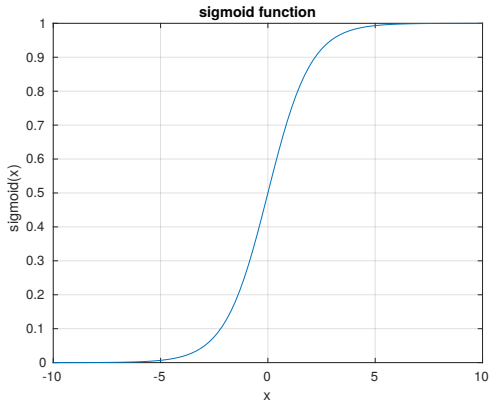


Figure 2.1: Sigmoid function

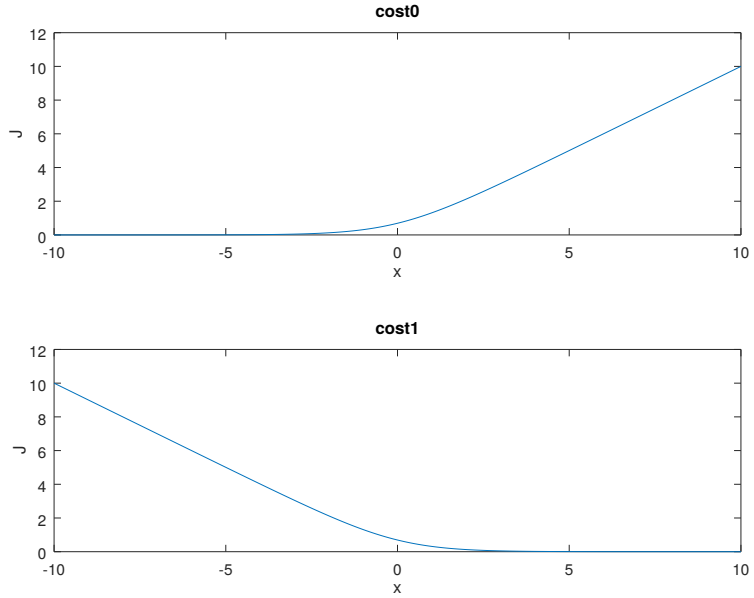


Figure 2.2: Logistic Regression Cost Function

2.2 Neural Networks

Artificial neural networks are a group of machine learning algorithms inspired by the human brain. The computation is done by a large group of neurons, which are connected with synapses. Each neuron aggregates its inputs. If the inputs aggregation exceeds a threshold, the neuron activates and sends the activation (positive value, logical true) to other neurons.

2.2.1 Mcculloch-Pitts Neuron

McCulloch and Pitts (1943) formulated the mathematical model of the neuron, which is being used (with some minor modifications) even today. The mathematical model looks as follows:

First, the inputs are aggregated, and the bias is added (or the threshold is subtracted as in the original formulation), e.g.:

$$z = \sum_{i=1}^{N} w_i \cdot x_i + b \tag{2.3}$$

Then, the non-linear activation function is applied (to incorporate decision), e.g.:

$$a = \sigma(z) \tag{2.4}$$

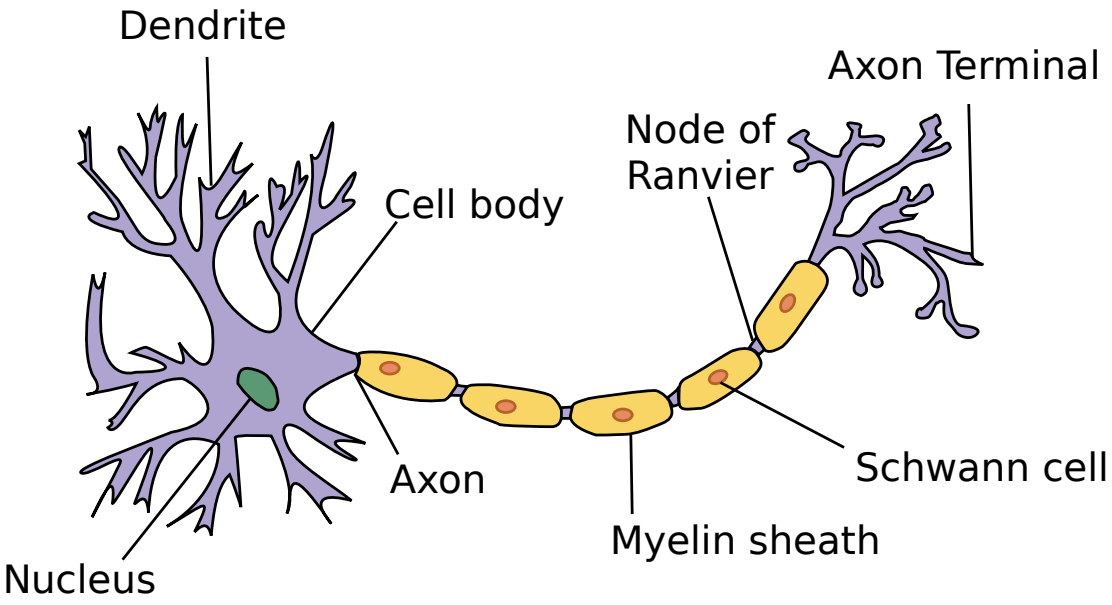


Figure 2.3: Biological neuron

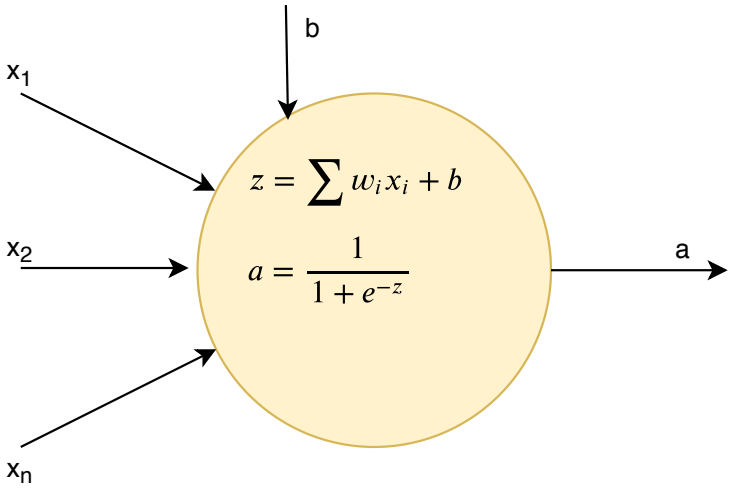


Figure 2.4: McCulloch-Pitts artificial neuron

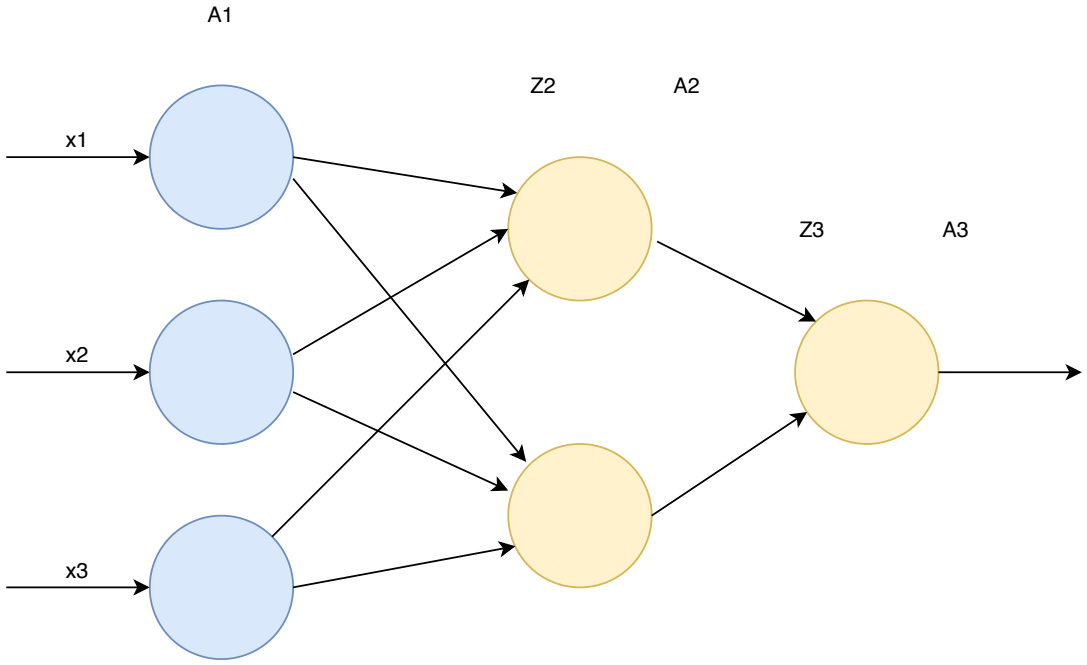


Figure 2.5: Feedforward neural network

The same model is being used today, however there are many different activation functions and several aggregation functions which are described later in the text.

2.2.2 Feed-Forward Neural Network

Most of the current state-of-the-art neural network architectures are based on simple feed-forward neural network (FFNN)[1]. In this network, the neurons (Mcculloch-Pitts neurons with various activation and aggregation functions) are arranged into a layered network where each neuron can be directly connected only with neurons in surrounding layers. The feed-forward network architecture is shown in Figures 2.4 and 2.5.

2.2.3 Forward Propagation and Loss

The forward propagation is a series of aggregation and activation functions. If the aggregation function is weighted sum and the activation is sigmoid, the layer is equivalent to the logistic regression.

The loss function is dependent on the concrete task, but the most common loss function for classification tasks is cross-entropy with the softmax activation function on the output layer:

$$softmax(X_i) = \frac{e_i^x}{\sum_{j=0}^{len(X)} e_j^x} \tag{2.5}$$

[1]sometimes referred to as the multi-layer perceptron

$$J = \sum_{i=0}^{m} y_i \cdot log(a_i^{last}) \tag{2.6}$$

2.2.4 Backpropagation Algorithm

The Backpropagation algorithm is a way of computing partial derivatives with respect to the weights based on the chain rule (eq. 2.7). If we have partial derivatives $\frac{\partial J}{\partial W}$, we can train the model with standard gradient-based optimizers. The backpropagation algorithm works as follows:

After computing the forward propagation and the loss value, we compute for each layer starting at the top:

1. Rate of parameters change. Formally this is the $\frac{\partial J}{\partial W_l}$
2. Backpropagation error δ_l. Formally the derivative with respect to the layer input $\frac{\partial J}{\partial Z_l}$

$$\frac{\mathrm{d}f(g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(g)}{\mathrm{d}g} \cdot \frac{\mathrm{d}g(x)}{\mathrm{d}x} \tag{2.7}$$

Intuitively according to the chain rule:

$$\frac{\partial J}{\partial W_2} = \frac{\partial J}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial W_2} = A_2 \cdot \frac{\partial J}{\partial Z_3} \tag{2.8}$$

$$\frac{\partial J}{\partial Z_2} = \frac{\partial J}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial Z_2} \tag{2.9}$$

$$\frac{\partial Z_3}{\partial Z_2} = \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \tag{2.10}$$

$$\frac{\partial Z_3}{\partial A_2} = W_2 \tag{2.11}$$

And this sums up to:

$$\frac{\partial J}{\partial Z_2} = W_2 \cdot \frac{\partial \sigma(Z_2)}{\partial Z_2} \cdot \frac{\partial J}{\partial Z_3} \tag{2.12}$$

and same as for the succeeding layer (eq. 2.8)

$$\frac{\partial J}{\partial W_1} = \frac{\partial J}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_1} = A_1 \cdot \frac{\partial J}{\partial Z_2} \tag{2.13}$$

Usually, we combine output activation and cost so that:

$$\frac{\partial J}{\partial Z_3} = A_3 - Y \tag{2.14}$$

because we want to have a linear gradient with respect to the error on the output layer. This is true for both the linear activation with least-squares cost (standard linear regression model) and the softmax activation with cross-entropy cost (also used in logistic regression).

2.2.5 Deep Neural Network

Recently (in the past ten years), as we have much more computational power (mainly GPUs), deep neural networks (DNNs) became very popular. Defining deep learning is not an easy task. The deep neural network is sometimes defined as an artificial neural network where we have more hidden layers (in opposite to the standard feed-forward network where we have only one hidden layer). Another definition is that with DNNs, we do not need to extract interesting features manually. The network accepts raw inputs (e. g. pixels in case of an image), and it extracts the interesting features itself. There are two basic approaches typically used in deep learning; Convolutional neural networks (DCNN) and recurrent neural networks (RNN). More recently, attentional networks based on the transformer architecture appeared, and they became very successful in various tasks.

2.2.6 Activation Functions in Deep Learning

This section summarizes various activation functions for deep learning. In classical neural networks, the activations mostly used were *sigmoid* and *tanh*.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

The range of the sigmoid is $(0, 1)$, which makes its output interpretable as a probability, but it is shifted to 0.5 ($\sigma(0) = 0.5$), so it shifts the mean value, which complicates gradient-based training. The advantage of *tanh* is its symmetricity around 0 (it preserves the mean). The disadvantage of both functions is that their gradient is decreasing very quickly on both sides.

Softsign

$$f(x) = \frac{x}{1 + |x|} \tag{2.16}$$

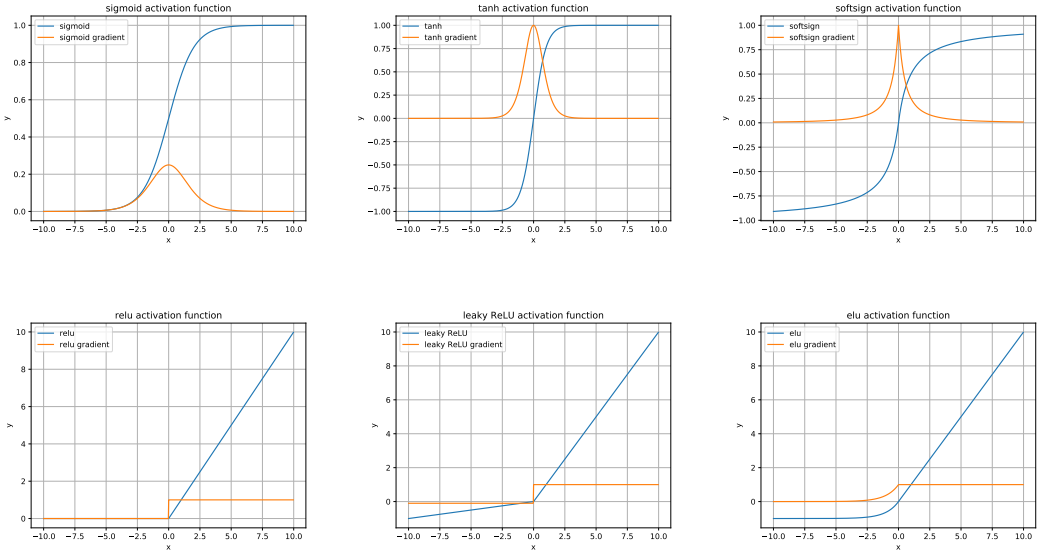


Figure 2.6: Activation Function Examples

ReLU

the rectified linear unit (ReLU) (Nair and Hinton, 2010) has been proposed to solve the *vanishing gradient problem*[2]. It copies the idea from logistic regression to have a decreasing gradient only on one side (Figure 2.2). In many cases, improper training with *sigmoid* or *tanh* activation is caused by a decreasing gradient with increasing distance from the optimum. *ReLU* tries to solve this problem.

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.17}$$

ReLU is the simplest non-linear function with a single point of non-linearity.

Leaky-ReLU

Although it is super effective in practical use, the ReLU activation function may be problematic to train due to its zero gradient in the negative domain. This way, the network cannot learn from deactivated neurons, and it activates them only by updating the weights of other neurons with some random chance. Leaky ReLU (Maas et al., 2013) solves this problem by setting the value in the negative domain to linear with a small negative slope. So there is some gradient towards the decision point.

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ Cx, C \ll 1 & \text{otherwise} \end{cases} \tag{2.18}$$

[2]Details can be found in Section 2.2.7

Maas et al. (2013) set C to 0.01.

PReLU

He et al. (2015) proposed a generalization of ReLU called Parametric ReLU, which is simply the leaky Relu where C is the parameter trained by a network as well. With $PReLU$, negative values can still affect the training (because there can be non-zero gradient). But for $PReLU$ it is harder to model a decision (discrete) because all negative examples are still affecting the activation.

ELU

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha \cdot (e^x - 1), & \text{otherwise} \end{cases} \tag{2.19}$$

ELU is very similar to $ReLU$, but it has a smooth gradient decrease in the negative part. It is sort of a compromise between $ReLU$ and $PReLU$. The gradient in the negative part is non-zero, but it is decreasing very quickly.

2.2.7 Initializing Weights

With standard activation functions used in simple FFNN (sigmoid or tanh), especially very deep neural networks face the *vanishing gradient* problem because the first derivative of those functions goes down very quickly in both directions away from the decision boundary. This is a big problem, especially for weights initialization, because if we initialize weights in a wrong way, we may never converge due to small gradients. Glorot and Bengio (2010) proposed a method for weights initiation with standard activation functions so that the mean and variance of the data do not change through the layers of the network (or the mean and variance change will be the least possible). In this way, all the neurons activation will be at a similar distance from its critical value (decision boundary), and the derivatives will be on the same scale.

Suppose we have input data centered around zero with unit variance. We want to have the same mean and variance on the succeeding layer. If the variance of the weights $var(W^l) = 1$, then the variance of the output would be:

$$var(z_i^l) = \sum_j var(a_j^{l-1}) \cdot var(w_{i,j}^{l-1})$$

if the mean of the input and weights is equal to zero. Now, if the variance of the weights is equal to one:

$$var(z_i^l) = var(a^{l-1}) \cdot n^l)$$

where n^l is the number of inputs of the layer l. So if we want $var(z^l) = var(a^{l-1})$, we need the $var(w^l) = \frac{1}{n^l}$ for the forward case and $var(w^l) = \frac{1}{n^{l+1}}$ for the backward case. To compromise between these two constraints, Glorot and Bengio (2010) suggest initializing the weights with the average of these two.

$$var(w^l) = \frac{2}{n^l + n^{l+1}}$$

If we use the uniform distribution, we initialize the weights according to:

$$W = U(-\frac{\sqrt{6}}{\sqrt{n^l + n^{n+1}}}, \frac{\sqrt{6}}{\sqrt{n^l + n^{n+1}}})$$

because $var(U) = \frac{1}{12} \cdot (b-a)^2$

Later He et al. (2015) introduced a method for initializing weights for ReLU activation in the same manner. The inference is very similar, and it leads to this initialization rule:

$$var(W^l) = \frac{2}{n^l}$$

The weights can be drawn from a uniform or a normal distribution with zero mean and derived variance for both initializers.

2.2.8 Batch Normalization

Batch normalization is another technique to avoid very different gradient scales in different layers. Ioffe and Szegedy (2015) proposed a normalization schema as a part of the training where it can be applied on the input of each layer. Normalized inputs on each layer should speed up training due to similar gradient scales (same as in standard feature normalization). It also solves the problem of non-proper weights initialization because the layers do not change mean and variance anymore. Ioffe and Szegedy (2015) also address the problem of restrictive properties of such a transformation (fixed mean and variance). They introduce two trainable parameters $\beta = \{\beta^{(1)}, ...\beta^{(n)}\}$ and $\gamma = \{\gamma^{(1)}, ...\gamma^{(n)}\}$ so that:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{2.20}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{2.21}$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.22}$$

$$y_i = \gamma \hat{x_i} + \beta \tag{2.23}$$

where m is the size of the batch, n is the number the the parameters of the lazer to be batch-normalized, and ϵ is a small constant added for numerical stability. In this way, the network can learn the optimal mean and variance of the inputs

for each layer. The problematic terms here are μ_B and σ_B^2 whose gradients depend on the whole batch, which is computationally expensive. The backpropagation inference can be found in Ioffe and Szegedy (2015).

2.2.9 Regularization

Regularization is a standard machine learning technique to prevent over-fitting. It makes the model to generalize better. The standard regularization technique used in machine learning is L_2 parameter penalization (referred to as L_2 regularization). It alters the loss function to penalize too complex hypotheses. The formal definition of L_2 regularization is defined by Equation 2.24.

$$\hat{J} = J + \sum_{i=1}^{N} w_i^2 \tag{2.24}$$

With deep neural networks, many different regularization techniques have been proposed. Nowadays, the most common one is the dropout. Dropout (Srivastava et al., 2014) prevents overfilling by deactivating some neurons in each learning step. This way, the neurons with the biggest information value are sometimes turned off, and so they give a chance to others to affect the resulting model.

Another regularization technique used with DNN is gradient noise (Neelakantan et al., 2015). In this method, the gradient used for optimization is the linear combination of the actual gradient computed with backprop and the random noise. Several different approaches have been proposed. Neelakantan et al. (2015), inspired by Welling and Teh (2011), proposed adding the Gaussian gradient noise decreasing with training time with the variance equal to:

$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma} \tag{2.25}$$

$$\hat{g}_t = g + N(0, \sigma_t^2) \tag{2.26}$$

Graves (2011) and Blundell et al. (2015) try to make the network to learn hyper-parameters of the distribution for the noise generation itself. This way, they get closer to the Bayesian posterior from the ML hypothesis, which is chosen by standard neural networks. This method is a Bayesian alternative to regularization (dropout).

2.2.10 Parameter Sharing Relaxation

Parameter sharing relaxation. (Kaiser and Sutskever, 2015) is another technique used to boost the convergence of optimization methods in DNN. In a recurrent neural network with parameter sharing relaxation, the parameters are not shared across all timestamps, but we use r independent sets of parameters. Next, we add a term into the loss function, which is proportional to the distance between

these parameter sets. This term is then multiplied by the scalar weight called *relaxation pull*. At the beginning of the training procedure, the *relaxation pull* is set to 0, so we have r independent sets of parameters, and the network can learn more different hypotheses. As the training continues, the *relaxation pull* is being increased linearly, so the network converges to a single set of parameters.

2.2.11 Convolutional Neural Network

Convolutional networks were introduced on image classification (LeCun et al., 1998). The basic idea is to learn an interesting pattern that should be detected in the image. Their presence or absence in the image should be the discriminative attributes for the classification. Convolutional networks are organized into deep structures, where each successive layer should detect more complex patterns by combining the patterns found by the previous layer. After each convolutional layer, there is a pooling layer, which reduces the dimensionality.

Mathematical Notation

In mathematics, convolution is defined as an integral of the product of two functions where one is reversed and shifted:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau. \tag{2.27}$$

In machine learning, we use its discrete variant:

$$(f * g)_k = \sum_{i=-\infty}^{\infty} f_i \cdot g_{k-i} = \sum_{i=-\infty}^{\infty} f_{k-i} \cdot g_i \tag{2.28}$$

Convolutional Layer

In the convolutional layer, we define the trainable set of convolution kernels, which are moved across the whole input space (for example, whole image) searching for matches. The layer computes the convolutions between the input and all the trainable filters.

Pooling Layer

Pooling reduces the dimensionality of the input by applying reductional operation on a region of the size given by the hyper-parameter. There are several types of reductional operations, but max and average pooling are used the most.

Other Layers

After a sequence of convolutional and max-pooling layers, we map the output of the convolution to a set of classes by a fully connected layer(s) same as in

other types of neural networks, and then we backpropagate the error (by standard backpropagation algorithm) through all the layers updating the weights.

Convolution on Text

Textual data can also be processed with a convolutional neural network as a sequence of words or characters (or any other tokens). In this case, we have 1D convolution (only one dimension of the data is sequential).

For example, Kalchbrenner et al. (2014) used deep convolutional network for sentiment classification. In many cases, convolutional networks are used as character-based models for adding syntactic information into the models.

The neural networks, very similar to convolutional networks, where the sequential dimension expresses time, have been called *time-delayed networks*. In more recent years, these two terms have been practically merged, and we use the term convolutional network even in case of time-dependent data.

When we have single-layer CNN with filters of size n (which is quite standard on the text), the network is learning to find important n-grams, and it cannot handle longer dependencies than n.

2.2.12 Recurrent Neural Network

For the sequential data, the recurrent neural networks are now used widely. There are many different architectures, but the basic concept is always the same. The RNN model is the sequence of RNN cells where the output of each cell depends on current inputs and the previous cell state. Some non-linear transformations have to be performed to model a decision. Elman (1990) defined recurrent neural network as follows:

$$h_t = \sigma_h(W_h * x_t + U_h * h_{t-1} + b_h) \tag{2.29}$$

$$y_t = \sigma_y(W_y * h_t + b_y) \tag{2.30}$$

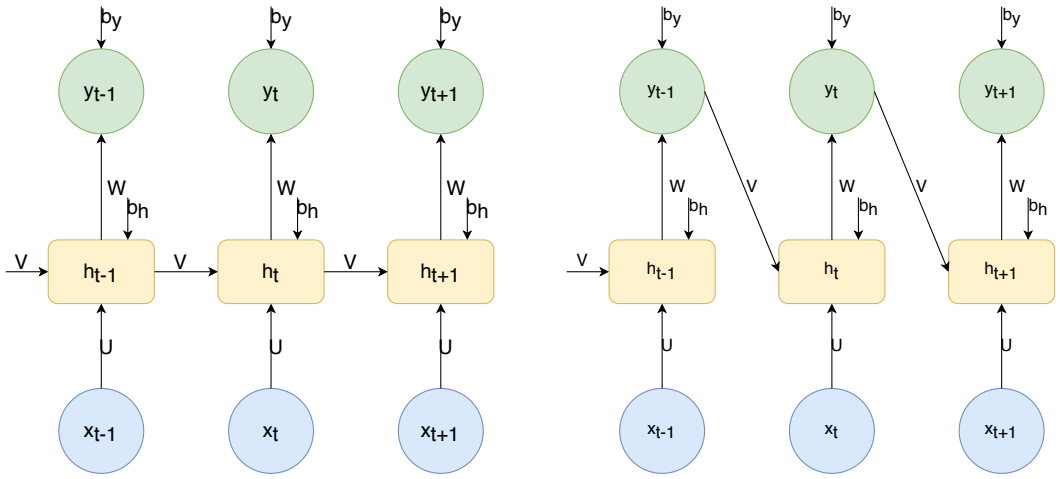
Jordan's definition is slightly different:

$$h_t = \sigma_h(W_h * x_t + U_h * y_{t-1} + b_h) \tag{2.31}$$

$$y_t = \sigma_y(W_y * h_t + b_y) \tag{2.32}$$

Here the input to the next timestamp is the current output, whereas in Elman's definition, the next timestamp depends on the current hidden state. This type of sequential network has significant limitations. The biggest one the input is weighted independently of the previous state. This way, the network cannot control properly what to store. Consequently, it suffers from the *vanishing/exploding*

gradient problem. Many different approaches on how to solve these have been developed. The most important are different activation functions and more advanced RNN cells. Basic RNN architectures are shown in Figure 2.7.



(a) Elman RNN (b) Jordan RNN

Figure 2.7: Basic RNN architectures

LSTM

Hochreiter and Schmidhuber (1997) proposed the Long short-term memory (LSTM). It is the standard recurrent neural network with a different cell. The LSTM cell operation works like this:

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{2.33}$$

Where:

- x_t is the current input,
- h_t is the current output vector,
- f_t is the forget gate,
- i_t is the input gate,
- o_t is the output gate,
- c_t is the hidden cell state,
- U, W, b parameter matrices and vector.

- **Forget gate** controls what part of the previous hidden state should be copied without any change based on input,
- **input gate** controls what part of the input should be added to the current hidden state,
- **output gate** controls what part of the hidden state should be passed to the output.

GRU

Cho et al. (2014) simplified the gating mechanism and proposed the *Gated Recurrent Unit (GRU)*. *GRU* cell operations are shown in the following equations.

$$\begin{aligned} z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\ h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h) \end{aligned} \tag{2.34}$$

Where:

- x_t is the current input,
- h_t is the current output vector,
- z_t is the update gate,
- r_t is the reset gate,
- U, W, b are The parameter matrices and the bias vector.
- **update gate** controls the mixture of the previous unchanged state and new state based on the current input,
- **forget gate** controls what part of the previous state should be used to produce the next state based on input and previous state.

Both LSTM and GRU are designed to control better what to remember and what to forget when. The gates are dependent on the current input and on the previous state, so the network can learn, that some information is no longer useful based on the state (the information is already in the hidden state, or some information in the hidden state became irrelevant based on inputs).

Recurrent neural networks can be used for creating representations of whole sequences or for contextualized representations of individual tokens. For token representations, we use h_t of each timestep, whereas for the whole sequence representation, the last state is used. Especially for token representations, it is important to capture the context from both sides. For this purpose, the bidirectional RNNs are often used. In the bidirectional RNN, we process the same sequence by two RNNs, the first processes the input from left to right and the second from right to left. In the end, we concatenate (or sum) both representations.

Another important approach is stacking the RNNs. In this model we stack more recurrent layers, so that output of the first layer at each timestep is fed as input to the second layer at the same timestep. The motivation is the same as for building deep feed-forward networks. A single-layer model with enough capacity can probably learn the same decisions, but it has been empirically shown, that deeper models are easier to train Pascanu et al. (2013).

When we stack the bidirectional RNNs, the upper layers have the information about the whole sequence, and they probably can learn more than single layer RNNs. For example, it has been shown to improve the performance of speech recognition in Graves et al. (2013).

2.2.13 Encoder-Decoder

The encoder-decoder architecture has been proposed to generate sequences (seq2seq model) with (recurrent) neural networks. It is quite a general concept where the whole input is encoded into a single hidden state at the first step, and then the output is sequentially generated from that state. In NLP, most neural machine translation models are based on the encoder-decoder architecture. In neural machine translation, the source sentence is at first projected into a hidden state which is the vector in the embedding space shared between languages ("interlingua"). The sentence in the target language is then generated sequentially from that shared space.

Figure 2.8 shows standard encoder-decoder architecture on the example of machine translation with RNN. When decoding the output, we first feed to the network a special token that signalizes the start of the decoding stage (NULL in the Figure, or EOS is sometimes used). In the next steps, we feed into the network previously generated tokens (can be embedded as well). The decoder modifies the state according to generated tokens. At the training time, we feed into the decoder the expected outputs no matter what the network actually generates. At the inference time, we need to use the previously generated token.

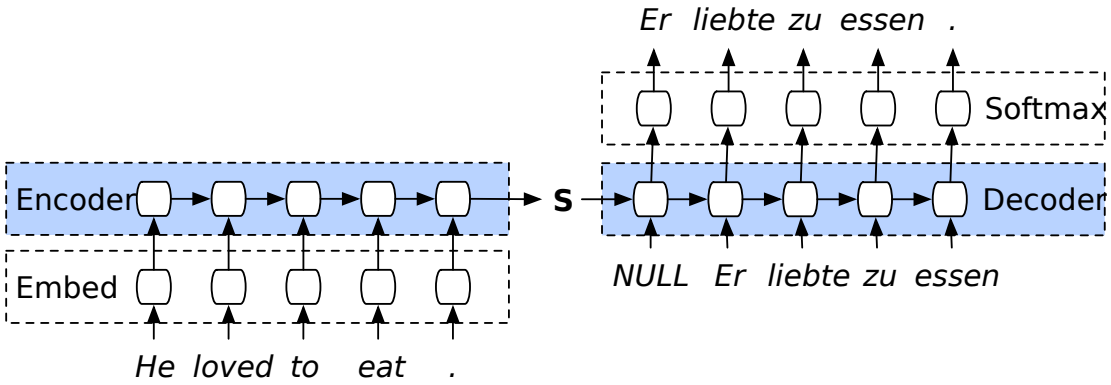


Figure 2.8: Encoder-Decoder Architecture

A special case of the *encoder-decoder* is autoencoder, where the expected output is the same as the input. Autoencoder thus first encodes the input into the

hidden state, and then it reconstructs the original input from the hidden state. So it generally compresses the data.

2.2.14 Attention-Based Networks

For the encoder-decoder architecture, long-distance dependencies are quite hard to capture since the whole sequence is stored in a single state and decoded then. The attention mechanism has been proposed to mitigate this problem.

The attention mechanism takes into account specific inputs when generating the output. The network learns how important is the specific input position when it needs to generate the output on the concrete position. Formally, the attention is the function of the encoder input and the decoder state producing the attention score (importance, relevance).

Luong et al. (2015a) presented attentional encoder-decoder architecture in neural machine translation. They proposed several modifications. In the simplest case, the attention mechanism works like this:

1. The output of the decoder in timestamp t is concatenated with *context vector* c_t:

$$\bar{h_t} = tanh(W_c \cdot [c_t; h_t]) \tag{2.35}$$

$$o = softmax(W_s \cdot \bar{h_t}) \tag{2.36}$$

2. The *context vector* c_t is given by the weighted sum of the encoder hidden states where the weights are given by the attention scores (attention vector a_t):

$$c_t = \sum_S a_t^S \cdot \bar{h_S} \tag{2.37}$$

3. Attention vector a_t is given by softmax over the attention scores:

$$a_t = \frac{exp(score(h_t, \bar{h_S}))}{\sum_{\bar{h_S}} exp(score(h_t, \bar{h_S}))} \tag{2.38}$$

   There are many slightly different approaches on how to compute the score, for example:

$$score(h_t, \bar{h_S}) = \begin{cases} h_t \bar{h_S} \\ h_t W_a \bar{h_S} \\ W_a[h_t; \bar{h_S}] \end{cases} \tag{2.39}$$

and many others.

To reduce computational complexity, Luong et al. (2015a) also propose to use the *local attention*. In their *local attention* every word can attend only to a small subset of the closest surrounding words.

Later the attention concept has been generalized in the way that it does not have to be between encoder and decoder, but it can be between arbitrary layers. The attention between two layers of the same part of the model is called self-attention or sometimes intra-attention.

Parikh et al. (2016) used attention in the natural language understanding model.

Transformer

Vaswani et al. (2017) proposed a learning method based mainly on attention. They show that attention in combination with feed-forward layer has at least the same (for some tasks even better) computational power as RNNs. The proposed architecture, called the *Transformer*, is widely used in recent models. The authors proposed multi-head attention as a crucial learning mechanism.

They formalized attention as the operation of *query*, *key*, and *value* as follows:

$$A(Q, K, V) = softmax(Q \cdot K) \cdot V \tag{2.40}$$

In the standard encoder-decoder attention, the *key* comes from the previous layer of the decoder and both *query* and *value* come from the encoder. In self-attention, all *query*, *key*, and *value* come from the previous layer of the same stack. This is now the most common formalism to describe attention.

Vaswani et al. (2017) further proposed *scaled dot product attention*:

$$A(Q, K, V) = softmax(\frac{Q \cdot K}{\sqrt{d_k}}) \cdot V \tag{2.41}$$

where d_k is the dimension of *query* and *key*. The motivation behind scaled dot-product attention is to preserve variance in a deep attention-based model. When the input has zero mean and unit variance and the output of dot-product attention will have zero mean but variance equal to d_k (because of summing d_k elements with unit variance). When we scale the output by the factor of $\frac{1}{\sqrt{d_k}}$ the output also has unit variance.

Multi-head attention is another generalization of attention mechanism where we compute several attentions with the different projection matrices, and then we join the results with another linear projection:

$$A_{multiHead}(Q, K, V) = Concat(head_1, head_2, ..., head_N) \cdot W^O \tag{2.42}$$

where

$$head_i = A(QW_i^Q, KW_i^K, VW_i^V) \tag{2.43}$$

So we compute N different attentions, we concatenate them and then project them onto a result vector. In the *Transformer* model, multi-head attention is used as the standard inter-attention (or encoder-decoder attention) and as a self-attention between both encoder and decoder layers. There is no recurrent network used in the *Transformer* architecture. The architecture of the *Transformer* for machine translation is depicted in Figure 2.9. It is a standard encoder-decoder architecture where both encoder and decoder are based mainly on intra-attention (or self-attention), and there is inter-attention between encoder and decoder. After every attention layer, there is a feed-forward layer. It consists of two fully connected layers with $ReLU$ activation. The same weights of the FFNN are used for all the positions. All the layers can be skipped through the residual connection.

The advantage of the *Transformer* over the standard feed-forward network is that in the attentional layer, the input is not changed with non-linear operations and the network can preserve the original input value and its variations and linear combinations. In the attention layer, the only non-linear operation is *softmax* used when computing transformation matrix. The self-attention can be formalized as the transformation $Y = T \cdot X$, where $T = softmax(\frac{Q \cdot K}{\sqrt{d_k}})$, so the only operation performed with the input is multiplying with the transformation matrix. This is also true for the multi-head attention because all the additional operations with heads are linear transformations.

In other words, with FFNN the output of the layer are the decisions made according to input, whereas in *Transformer* the output is the modification of the input according to the decisions made.

Positional Encoding Since the attention layer has no information about the word order (as the recurrent networks have), we need to add positional information into the model somehow. For this purpose, the positional encoding can be used. Vaswani et al. (2017) tried two approaches to positional encoding. The first one is straightforward. They take the absolute position of the word in the sentence and use it as an index into a trained embedding matrix (position embeddings). The second way tries to capture relative position directly in its encoding. This is achieved by representing the position with the sines and cosines of the absolute position with different frequencies. Positional encoding is summed up with corresponding word embedding.

The authors used the *Transformer* for machine translation in the encoder-decoder architecture, as it is shown in Figure 2.9. In many other models (for example, various sentence encoders), only the encoder part is used, and then the encoded representation is projected directly to the output.

Probably the most significant advantage of the *Transformer* over recurrent networks is that the *Transformer* is much more computationally efficient (it can be computed for whole sequential dimension in parallel).

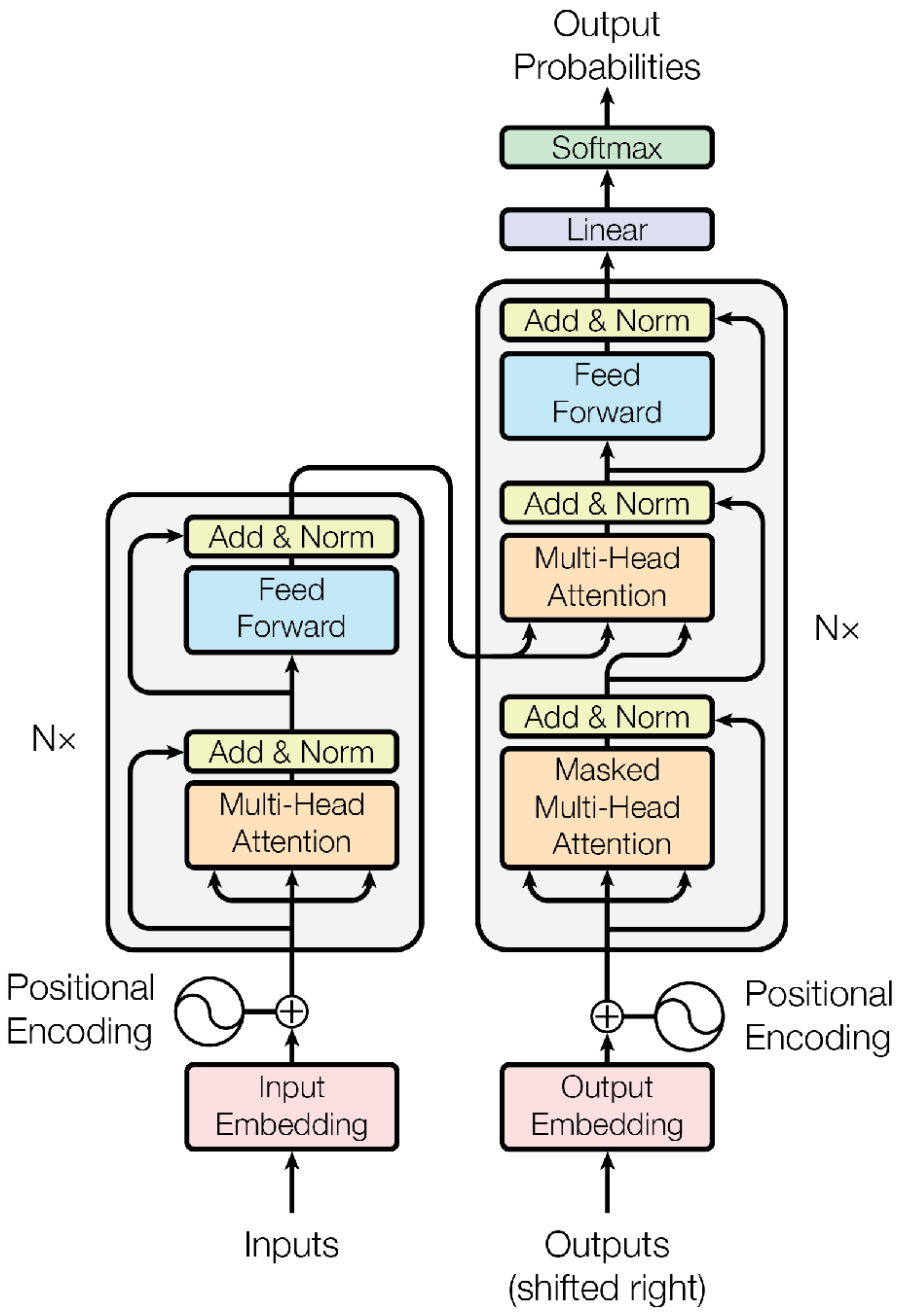


Figure 2.9: Transformer model architecture from Vaswani et al. (2017).

2.2.15 Tree-Structured Networks

In Natural Language Processing, many models have a tree structure. For example, formal representations of both syntax and semantics are formed in a tree. In recent years many researchers were trying to find the best way how to process trees in neural networks. There are at least three different approaches:

1. The simplest to implement is to linearize the tree (for example, by one of the tree search strategies), and then we can use standard recurrent neural networks to make a tree representation.

(a) Scaled dot-product attention.

(b) Multi-heead attention.

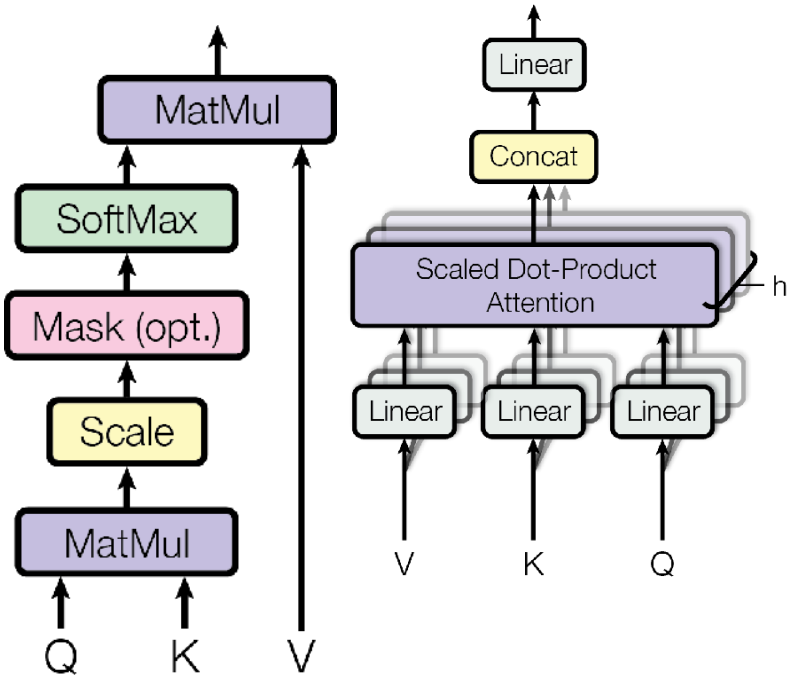


Figure 2.10: Types of attention from Vaswani et al. (2017).

2. Socher et al. (2013) proposed the tree-recursive neural network. The network itself is formed into a tree structure.

3. There are many tree modifications of standard approaches for modeling sequences. For example, tree LSTM (Tai et al., 2015) and tree-based convolution (Mou et al., 2015).

Recently the classic RNNs have been proved (empirically) to have superior performance over the tree-structured networks. Li et al. (2015) compare different methods in various tasks.

2.3 Multi-Task Learning

Most of the machine learning techniques solve the task in a completely isolated environment. They do not model the surrounding world or some related decisions. The idea of multi-task learning is that knowledge of one task can help to solve the other. (Caruana, 1997), Thus, in multi-task learning, we combine objectives of more tasks because we suppose that knowledge from one task can help the other. There are several possible reasons why to use multi-task learning.

1. *The natural case* – When we want to solve a complicated task composed of several subtasks, neural networks with multi-task learning are very successful tool to solve it end-to-end.

2. *Auxiliary tasks* – While solving a single task, it has been found beneficial in many cases to add some auxiliary tasks, which add more features that can help the system to make a decision.

2.3.1 Neural Networks for Multi-Task Learning

In recent years with the new wave of popularity of neural networks, many multi-task learning systems based on them have been developed. (Ruder, 2017) In neural networks, multi-task learning can be handled by just sharing some of the layers between tasks and their objectives.

2.3.2 Hard Parameter Sharing

In hard parameter sharing, some parameters are simply used and trained in both (all) tasks.

2.3.3 Soft Parameter Sharing

In soft parameter sharing, we have a different set of parameters for both tasks, but we add the term to the cost function, which makes the two sets to have similar values. For example, Euclidean distance can be used as the similarity measure for parameter sets:

$$J(\Theta) = J(\Theta) + \sum_i (\Theta_i^1 - \Theta_i^2)^2 \tag{2.44}$$

Chapter 3

Semantics

Generally, the task of semantic analysis is to capture the meaning of the text.

Understanding the meaning of words and texts is a crucial task for many natural language processing applications. Many ways how to represent texts have been developed. The simplest way how to represent a word is a one-hot vector. In this way, the words are represented as vectors in a high dimensional space where every word is orthogonal to each other. Therefore, there is no semantic information in this representation (only lexical) because there exists no relation between words encoded in this representation. The approaches how to represent the meaning of words, sentences or even longer texts can be divided into two categories: *formal* and *distributional*.

3.1 Lexical Databases and Ontologies

In past decades people have created many resources of semantic knowledge. When we study the meaning of the words, the most important hand-created resource is probably Wordnet.

3.1.1 Wordnet

Wordnet (Miller, 1998) is the lexical database which groups the words according to their meaning into *synsets*. The synsets are linked with semantic and lexical relations with the form of an ontology.

The backbone structure of *Wordnet* is the acyclic graph of the hypernym/hyponym relations. It links more general synsets like (furniture, piece_of_furniture) to increasingly specific ones like (bed) and (bunkbed).

The meaning of a word can be represented by its position in the resulting graph.

We study *Wordnet* based semantic methods more deeply in (Konopík and Pražák, 2015). Here we also compare these methods to another group of methods based on distributional semantics, and we study if they can complement each other.

3.2 Distributed Representations

The distributional hypothesis (Harris, 1954) says that if two words appear in the same or similar context frequently, they tend to be similar in their meaning.

According to the distributional hypothesis, we can represent a word by the context in which the word is likely to appear. In this way, the words that appear in similar contexts have similar representations.

We can split distributional methods into two categories:

- **Global context** (or document context) methods model words to be more similar if they appear in the same or similar document. The documents are often represented as bag-of-words. It means that the word order in the document is not encoded in the representation.
- **Local context** methods consider the words semantically similar if their contexts of a few surrounding words contain the same or similar words.

The representations of words created according to the distributional hypothesis are called *word embeddings*.

3.2.1 LDA

Blei et al. (2003) proposed a generative Bayesian model for finding hidden (or latent) topics in the set of documents. LDA is the bag-of-words generative model. The distributions trained for this generative process can be used to represent the meaning of words and documents. Both documents and words can be represented as the distribution over the hidden topics. The generative process words as follows:

1. Choose $\Theta_i \tilde{D}ir(\alpha)$, where $i \in \{1, \ldots, M\}$ and $Dir(\alpha)$ is a Dirichlet distribution with a symmetric parameter α which typically is sparse ($\alpha < 1$).
2. Choose $\varphi_k \sim \mathrm{Dir}(\beta)$, where $k \in \{1, \ldots, K\}$ and β typically is sparse.
3. For each of the word positions i, j, where $i \in \{1, \ldots, M\}$, and $j \in \{1, \ldots, N_i\}$
   (a) Choose a topic $z_{i,j} \sim \mathrm{Multinomial}(\theta_i)$.
   (b) Choose a word $w_{i,j} \sim \mathrm{Multinomial}(\varphi_{z_{i,j}})$.

When the training is finished, we can represent words by their probabilities in the topics, and we can represent documents by their topics probability distribution.

3.2.2 LSA

Latent Semantic Analysis is one of the simplest global context methods. It uses the *term-document matrix*, which is the matrix where rows represent words and columns represent documents. Each entry contains the count of occurrences of

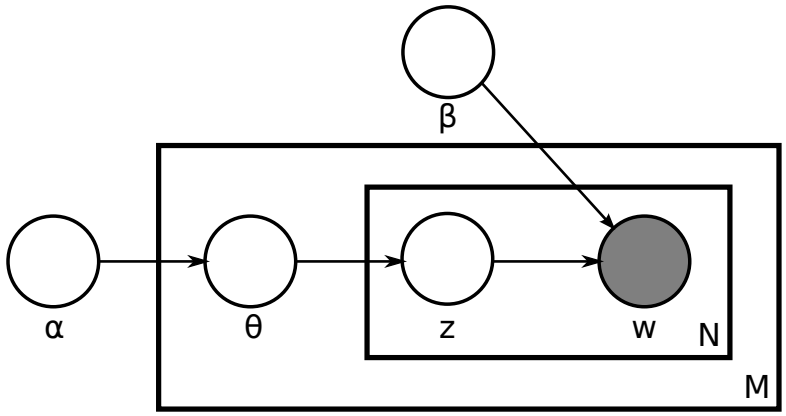


Figure 3.1: LDA Graphical Model Representation

ith word in jth document. The counts are typically weighted with *inverse document frequency (IDF)*[1]. Such a matrix is very sparse, and we need to reduce its dimensionality. LSA uses singular value decomposition (SVD) for dimensionality reduction. SVD decomposes the matrix $A = U\Sigma V^T$ where U is the matrix of left-singular vectors, Σ is the diagonal matrix of singular values, and V is the matrix of right singular vectors. Both U and V are orthogonal. *SVD* is the generalization of the eigenvalue decomposition and it can be simply derived from eigenvalue decomposition. If:

$$A = U\Sigma V^T$$

then

$$A^T A = V\Sigma^T U^T U\Sigma V^T = V^T \Sigma^T \Sigma V = V^T S V$$

and

$$AA^T = U\Sigma V^T V\Sigma^T U^T = U\Sigma^T \Sigma U^T = USU^T$$

because $U^T U = VV^T = I$ (orthogonality). $AA^T = USU^T$ is eigendecomposition of AA^T so V is the matrix of eigenvectors of AA^T and U is the matrix of eigenvectors of $A^T A$. The transformation is independent of the order of the columns, but by convention, they are sorted in descending order according to singular values. Upper submatrices are then optimal low-rank approximations of the original matrix. The higher the eigenvalue is, the more of the original variance it captures.

In *LSA*, words can be represented with rows of U (can be weighted with Σ), which can be interpreted as the weights for the linear combination of representatives of words (sort of word clusters, created from co-occurrences in documents). The documents can be represented with rows of V (can be weighted with Σ), which can be interpreted as weights of the linear combination of eigendocuments (the most representative documents as a word mixtures).

[1]The words that occurred in less document are more informative thus they are more important.

HAL

Hyperspace Analogue to Language (HAL) is the simplest method for local-context-based distributional semantics. It slides the window on a given input while counting the word co-occurrences. The algorithm works like this: Increase the count if the word i is in the left context of size k of word j. The counts can be weighted by distance. In this way, we get a matrix which rows contain right context counts, and columns contain left context counts. A word is then represented with the concatenation of the corresponding row and column. The big disadvantage of this method is the high dimensionality of the word representations.

This problem can be partially solved with *Random Indexing*. *RI* is the modification of *HAL* where in *HAL*, we sum the one-hot representations of the context words, whereas in *RI*, we create for each word a random vector of rank in thousands with a few randomly selected $+1$ and -1 in this way, the vectors are nearly orthogonal although they are much smaller than in case of *HAL*. We can set arbitrary dimensionality. The rest of the algorithm is the same as *HAL*.

3.2.3 Neural Networks' Hidden States

Textual representation from neural networks can be divided into two categories: *feature-based approaches* and *fine-tuning approaches*.

- **Feature-based Approaches** train neural network on unsupervised task (language model) and then use previously trained weights as features in a different neural network for the downstream task (supervised).
- **Fine-tuning Approaches** first pre-train the network on the unsupervised task (language model) and then use the same model for downstream tasks. In this way, only a few parameters are learned from scratch (only the projection layer), but all the parameters are typically fine-tuned for the downstream task.

Skip-Gram and CBOW

Mikolov et al. (2013a) Created two simple neural network models to create semantic representation. The basic idea is to take the hidden state of the neural language model to represent the meaning of words. The neural network learns to predict words according to their context (or it learns the context from central words), so the hidden state naturally captures the contextual information of the words. The basic idea to represent words with a neural network's hidden state comes from Bengio et al. (2003). Standard neural network for language modeling from Bengio et al. (2003) has four layers:

1. **Input** – Words are fed into the network in the one-hot representation.[2]

[2]The word is represented by a vector of the length equal to the dictionary size, where each element of the vector represents one word. Only one element of the vector is non-zero.

2. **Projection** – Input vector goes through the fully-connected layer, so the internal word representations are created.

3. **Hidden** – Creates representation of the context.

4. **Output** – Softmax layer which computes the probability of the current word according to the context.

Figure 3.2 shows the architecture of the basic language model based on the feed-forward neural network.

(Mikolov et al., 2013a) simplified this network by removing the hidden layer, and the context word representations are only summed up. In the Skip-gram architecture, the context is predicted from the central word, and in *CBOW* the central word is predicted from the context. Those models are also referred as *Word2Vec*. The architectures of skip-gram and CBOW are shown in Figure 3.3. For Skip-Gram, the context word probability is computed as:

$$p(w_o|w_c) = \frac{exp(W_c^{(0)} \cdot W_o^{(1)}))}{\sum_{w_c}^{V} exp(W_c^{(0)} \cdot W_o^{(1)})} \tag{3.1}$$

We can then use the standard cross-entropy cost:

$$J = \sum_{w_c}^{C} \sum_{w_o}^{N(w_c)} -log(p(w_o|w_c)), \tag{3.2}$$

where C is the sequence of all words in the corpus and $N(w_c)$ is the neighborhood function, which returns all the words in the context window of w_c.

Another simplification of *Word2Vec* is the negative sampling. Since we do not need to use the network as the language model and because *softmax* is a very expensive operation, we can replace it by approximation of the probability with sigmoid with few negative samples. Instead of predicting probabilities for each word on the output (classify into V classes), we put into the network the central and the context word, and we want to predict the probability of their co-occurrence.

$$p(w_o|w_c) = \sigma(W_c^{(0)} \cdot W_o^{(1)}) \tag{3.3}$$

We need to include negative examples in the cost since there is no *softmax* anymore.

$$J = \sum_{w_c}^{C} (\sum_{w_o}^{N(w_c)} -log(p(w_o|w_c)) - \sum_{w_o}^{U(k)} log(1 - p(w_o|w_c))) \tag{3.4}$$

where $U(k)$ generates k samples from the uniform distribution over the vocabulary.

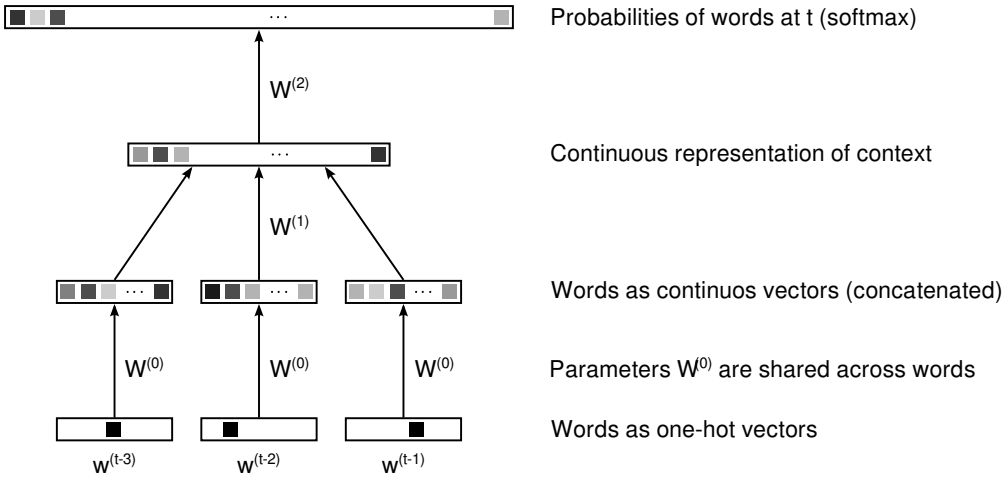


Figure 3.2: Basic neural network language model architecture

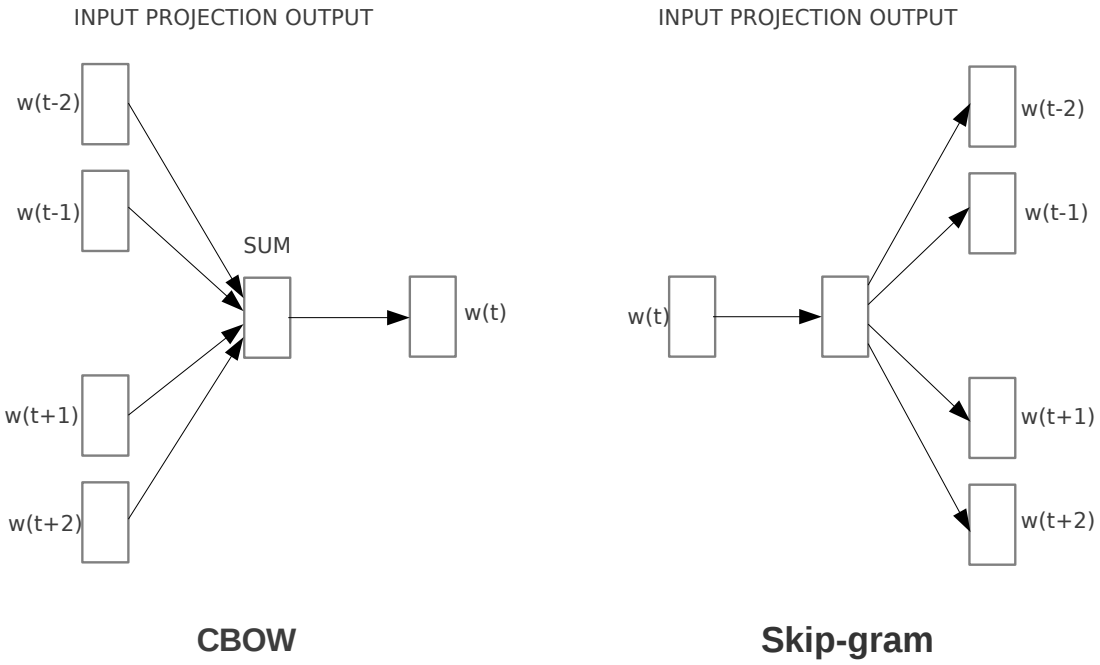


Figure 3.3: Architecture of *Word2Vec* Models (from Mikolov et al. (2013a))

3.2.4 Sentence Embeddings and Contextualized Word Embeddings

In this section, various contextual word representations and sentence representations are described. In recent years many of such models have been developed, and they all are very similar. That is why only the most popular ones will be described here chronologically. They are also summarized in Table 3.1

method	**dataset**	**model**	**params**	**input**	**tasks**	**GLUE score**
Skip-thoughts	Books Corpus	GRU	x	word	L2R-LM	61.3
ELMo	Word Benchmark	CNN+LSTM	x	word	L2R-LM	71.0
GPT	Books Corpus	Transformer	117M	BPE	L2R-LM	75.1
BERT	Books Corpus	Transformer	762M	WordPiece	MLM+NSP	82.1
ALBERT	Books Corpus	Transformer	235M	SentPiece	MLM+SOP	89.4
RoBERTa	Books,News,WebText,...	Transformer	$\approx 782M$	SentPiece	MLM	88.5
USE	Wiki,news,QA,SNLI	Transformer, DAN	x	word	L2R-LM+SNLI	-
GPT 2	WebText	Transformer	1542M	BPE	L2R-LM	-

Table 3.1: List of Recent Contextualized Models of Semantics

Input Representation

Current models for unsupervised pre-training of contextualized embeddings use different input representations (tokenizations). Some of them use standard word-level tokenization, and others use subword tokenizations to reduce effective vocabulary and deal with morphology in a better way. There are two commonly used subword tokenizations:

1. **Byte Pair Encoding (BPE)** (Sennrich et al., 2016) – Originally, BPE was used for data compression. It is a straightforward algorithm where it iteratively replaces the most frequent byte pair with a new single byte up to target vocabulary size or until there is no reoccurring byte pair. Sennrich et al. (2016) modified this algorithm for tokenization in NLP applications. First, they tokenize the text on the character level,[3] and then they iteratively merge the most frequent pair of tokens until they reach a target vocabulary size. The algorithm does not consider token pairs that cross the word boundaries.

2. **WordPiece Tokenization** (Wu et al., 2016) – Was originally used for Japanese segmentation. The algorithm is very similar to BPE. They first tokenize the text on the character level, and then they use a greedy algorithm to maximize the likelihood of the data obtained from a language model by merging the tokens. They basically iteratively merge a pair of tokens which increases the likelihood the most.

Skip-Thoughts

Kiros et al. (2015) proposed the *Skip-Thoughts* model for semantic representation of sentences. It can be described as a generalization of *Skip-Gram*, where instead of predicting surrounding words, we predict the previous and the next sentence given the actual sentence. For this purpose, the authors use a single-layer GRU-based encoder-decoder in the standard way (see section 2.2.13). The network is composed of one encoder and two decoders (one for the previous sentence and one for the next sentence). The encoder part is then used as a general-purpose encoder for capturing the meaning of the sentences.

The authors propose a vocabulary expansion method based on the linear transformation of vector spaces. Because the model is much more complex than *Word2Vec* and mostly because of the softmax on the top of the encoder-decoder, it is much harder to learn infrequent words. During the training, the vocabulary of the RNN was limited to the 20 000 most frequent words. The authors also trained a standard *skip-gram* with a large vocabulary. After the training, both word vector spaces W_{w2v} and W_{RNN} are taken, and W_{w2v} is transformed so that:

$$\hat{W}_{RNN} = \Theta \cdot W_{w2v} \tag{3.5}$$

[3] Adding special end-of-word character to be able to restore the original tokenization.

$$\Theta = \underset{\Theta}{\operatorname{argmin}} \left(\sum_{i}^{V} (\Theta \cdot W^{i}_{w2v} - W^{i}_{RNN})^2\right) \tag{3.6}$$

where the vocabulary $V = V_{w2v} \cap V_{RNN}$

In this way, the unknown words are first projected from *Word2Vec* embeddings, and the resulting embeddings can be fed into RNN to encode the sentence.

ELMo

ELMo (Peters et al., 2018) has a similar learning procedure to *Skip-thoughts*, but the model is more complex. The authors follow the neural language model by Jozefowicz et al. (2016). It consists of convolutional layer for processing characters and two-layer LSTM-based encoder on the word level. First, the character sequences are processed by a convolutional layer with 2 048 filters. Then the representations are projected to state with 512 elements. These are the word embeddings trained from scratch from character sequences. Next, there are two LSTM layers (bidirectional, but both directions are processed separately). On the top LSTM encoder, there is a softmax (or its approximation) to predict the next word. For semantic models like *ELMo* simple sampling algorithm like negative sampling is efficient enough since we do not need the inference step of the language model. For the actual language model, more advanced techniques are used (see Jozefowicz et al. (2016) for details). The crucial idea of *ELMo* is that the lower layers of the language model can help the transfer task a lot. So for each task, the task-specific weights for the layers are trained. More formally:

$$ELMo^{task}_{k} = \gamma^{task} \sum_{j=0}^{L} s^{task}_{j} h^{LM}_{k,j} \tag{3.7}$$

where s^{task} are softmax normalized weights, γ^{task} is a global scaling parameter, and L is the number of layers.

GPT

The GPT model (Radford et al., 2018) uses the transformer decoder (see section 2.2.14) for left-to-right language model. The model is pre-trained on the Books corpus dataset (Zhu et al., 2015), and then used for various NLU tasks fine-tuned with a single additional projection layer. During fine-tuning the language model is added as an auxiliary task (see section 2.3). The model is pre-trained on single-sentence inputs, but some of the transfer tasks process structured inputs (typically sentence pairs), so during the fine-tuning, multiple inputs are concatenated using special separator tokens. The motivation for this approach is to add the least possible number of additional parameters during fine-tuning and make better use of those pre-trained. In this way, we add a few additional tokens, which should be better than changing the architecture and train many additional parameters.

During training, the positional embeddings are trained rather than using the sine, cosine approach from Vaswani et al. (2017).

BERT

BERT (Devlin et al., 2018) stands for Bidirectional Encoder Representations from Transformers. It is one of the current state-of-the-art methods for semantic representation. The architecture is based on the *Transformer*. It belongs to fine-tuning approaches for semantic representations. The training objective consists of two tasks:

1. **Masked language model** In order to capture dependencies from both sides (not only left-to-right or right-to-left model) in masked LM we hide a certain number of random words, and the task is to predict them from the rest of the sentence. At the top of the encoder, there is the softmax classification of all the masked-out tokens. Thus, *BERT* is not the encoder-decoder model, but only the encoder with a dense layer and softmax on the top. In this way, the model can capture dependencies from both sides, whereas in standard encoder-decoder we cannot use both contexts at the same time, because the model would be able to see the word, which it needs to predict directly on higher layers.

2. **Next sentence prediction** Given two sentences, the task is to determine if one sentence follows the other. With this task, the model learns to capture relationships between two sentences.

BERT architecture is shown in Figure 3.4. The main difference between BERT and GPT is that BERT is pre-trained on the sentence pairs and the classification task of the next sentence prediction, so it learns to capture some relationships between sentences during training. Another significant difference is that BERT is using a bidirectional encoder thanks to masked LM, which should capture the context in a better way.

The pre-training of *BERT* is done on the concatenation of the books corpus and English Wikipedia. The multilingual version uses the Wikipedias of all 104 languages. Since no cross/lingual Wikipedia links have been used, the BERT model is multilingual, but its cross-lingual capabilities are limited. Details about cross-lingual properties of *BERT* are discussed later.

Universal Sentence Encoder

Universal sentence encoder is a multi-task learning approach for creating semantic representations. The model is trained primarily on the standard language model task in the same way as *Skip-Thoughts* Kiros et al. (2015) and conversational data via response suggestion task Henderson et al. (2017). In addition, *SNLI* (Bowman et al., 2015) is included as an auxiliary task during pre-training. The pre-training data includes Wikipedia pages and news for unstructured textual data, question-answer pages and discussion forums for conversational data, and SNLI dataset as

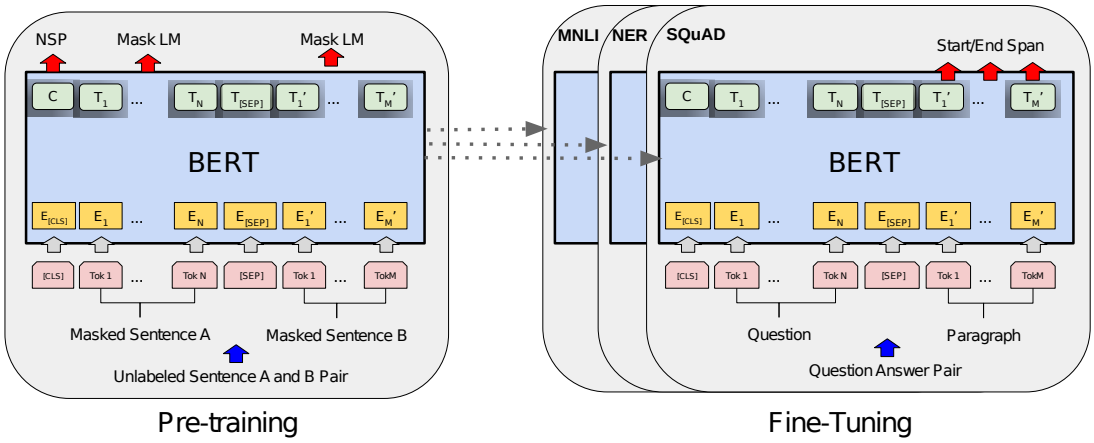


Figure 3.4: BERT pretraining and fine-tuning (from Devlin et al. (2018))

a general discriminative task data. *USE* is meant as a feature-based approach for creating universal sentence representation. The model is not supposed to be fine-tuned as it is. *USE* includes two encoder models Transformer, which is used in the standard way, and Deep Averaging Network (DAN). Deep Averaging Network is a very simple and fast model, where first, the embeddings for all the words in the sentence are summed up and then processed by a deep feed-forward network.

RoBERTa

RoBERTa (Liu et al., 2019) uses almost the same model as *BERT* with a few modifications:

1. They use much larger batches (8k).
2. They omit the next sentence prediction task.
3. They feed into MLM the maximum number of sentences to fit the maximal sequence length (512) separated by *SEP* token.
4. They use a larger subword vocabulary (50k).

ALBERT

In *ALBERT* (Lan et al., 2019) the authors propose several improvements of the *BERT* model. First they argue that the hidden size is unnecessarily large for word embeddings and they use lower dimensional projection layer to reduce the number of parameters. Instead of having $V \times H$ parameters for word embeddings they have $V \times E + E \times H$ parameters, where $E \ll H$.

Next they propose to use sentence ordering prediction (SOP) instead of the next sentence prediction. The task of SOP is to determine if two given sentences are in the right order. This task is significantly harder than NSP. In NSP the negative examples are random sentences from whole corpus and they are usually

very different from the positive ones. However, in SOP the model has to understand the semantics of the sentences much better in order to determine the order.

The last modification is to reduce the number of parameters by sharing all the parameters between transformer layers.

These modifications result into a model which has only 18M parameters in the same setting as BERT-large[4].

GPT 2

GPT 2 Radford et al. (2019) basically follows the *GPT* architecture; The only significant difference is in the dataset used for training. The authors created a new corpus called *WebText*, by crawling various web pages similarly to the *Common Crawl corpus*, but the authors state that *WebText* is much better is the quality of the text. The version of *WebText* used in *GPT 2* has around 40GB of text. It is crawled from the common web starting on *Reddit*. The Wikipedia pages are discarded to avoid overlap with common datasets. Since the dataset is much bigger and much more open-domain than the datasets used for previous models, the *WebText* is much harder to overfit, and thus much more parameters should be trained. *GPT* model has 117M parameters, whereas *GPT 2* has 1542M parameters. The authors state that it still under-fits the training dataset.

Additionally, the authors evaluate the ability of this general-purpose language model to learn various NLP tasks in the zero-shot setting. If we have a left-to-right language model which is learning to maximize the probability of the training text given the previous part of the sequence:

$$p(x) = \prod_{i=1}^{n} p(s_i|s_1, ..., s_{i-1}) \tag{3.8}$$

for any supervised task, learning can be expressed as estimating the probability $P(output|input)$. The probability also depends on the task so it can be formalized as estimating $P(output|input, task)$. The supervised objective is the same as the language model objective but evaluated only on the subset of the sequence. Therefore the global optimum of the language model is (in theory) also the global optimum of any supervised task. So a good language model should learn (itself) to perform on supervised tasks. Radford et al. (2019) evaluate zero-shot performance on supervised tasks by estimating $P(output|input, task)$ with the *GPT 2* model. For example, to make the model do translation, we can compute $P(English\_sent|French\_sent, few\_translation\_examples)$ by conditioning on a few examples we are describing the task.

3.2.5 Document Embeddings

The main problem of aforementioned Transformer-based models lays in the maximum length of the inputs. During pre-training, all the models use the maximum

[4]With the same hidden size, number of layers and number of heads as BERT-large. For comparison, BERT-large has 334M parameters.

length of 512 sub-word tokens due to quadratic time and memory complexity of the attention. It is long enough for word-level, sentence-level, and sentence pair tasks but it is far from enough for document-level tasks like document classification or some types of question answering. There are several ways how to deal with longer inputs in transformer-based models. Most of them are based on the pruning of the attention matrix (for example, with local attention [5]).

Longformer

Beltagy et al. (2020) propose several sparse attention mechanisms to reduce the complexity of the *Transformer* model. They use three types of sparse attention patterns:

1. First, they use the **sliding window** (same as standard local attention), where each token attends to w surrounding tokens.

2. **Dilated sliding window** – To further increase the context length without increasing computational complexity, the sliding window can be dilated. With d dilated sliding window, each word attends only to the words with relative distance divisible by d.

3. **Global attention** – To enable modeling of long-distance dependencies while having the linear complexity, the authors propose task-specific global attention patterns (rule-based). For example, in the case of sentence classification, all the tokens attend to the *CLS* token, and the *CLS* token attends to all other tokens.

Other Pre-Training Objectives

Another problem of standard BERT-like models comes from the masked language modeling task used for their pre-training. When we want to predict a word or a subword token based on the rest of the sentence, it can be determined from the small context in most cases, so BERT does not learn to model many long-distance dependencies between the words. For example, for the information retrieval task, many other pre-training objectives have been used.

Chang et al. (2020) propose different pre-training objectives for large-scale document retrieval. Beside more global information, their model benefits from sentence (document) relationship-based tasks. In large-scale document retrieval, a system cannot use query-document inter-attention during inference. There are too many documents to compare to, and the document representations need to be pre-computed.

Chang et al. (2020) use three global pre-training objectives:

- **Inverse Cloze Task (ICT)** – The task is to determine if a given context (paragraph) is surrounding a given sentence.

[5] described in Section 2.2.14

- **Body First Selection (BFS)** – Given a sentence from the first section of a Wikipedia page and a random passage from a Wikipedia page, the task is to determine if both come from the same page.
- **Wiki Link Prediction (WLP)** – Given a sentence from the first section of a Wikipedia page and a random passage from another page, the task is to determine if there is a link between those two pages.

3.3 Semantic Role Labeling

Semantic role labeling (Gildea and Jurafsky, 2002) is the task of shallow semantic parsing, where given a sentence, the task is to:

1. First, identify predicates (actions, events, etc.),
2. identify arguments of the predicates,
3. determine argument types (active entity, passive entity, other entities, and modifiers - time, place, etc.).

Figure 3.5 shows an example of an SRL annotation.

(1) [He]$_{\text{AGENT|A0}}$ believes [in what he plays] $_{\text{THEME|A1}}$.

(2) Can [you] $_{\text{AGENT|A0}}$ cook [the dinner] $_{\text{PATIENT|A1}}$?

(3) [The nation's] $_{\text{AGENT|AM-LOC}}$ largest [pension]$_{\text{THEME|A1}}$ fund,

Figure 3.5: Three SRL annotation examples

3.3.1 Feature Engineering

At the beginning of this task, with standard learning and feature engineering, many features for SRL have been developed. They are well summarized in Moschitti et al. (2008). Lang and Lapata (2011) proposed simple syntactic rules for argument identification and proved that syntactic features are suitable enough for this subtask. Semantic role labeling can be divided into four separate machine learning problems:

1. Predicate identification,
2. argument identification,
3. role labeling,
4. global optimization.

Standard features used in these approaches can be divided into several categories:

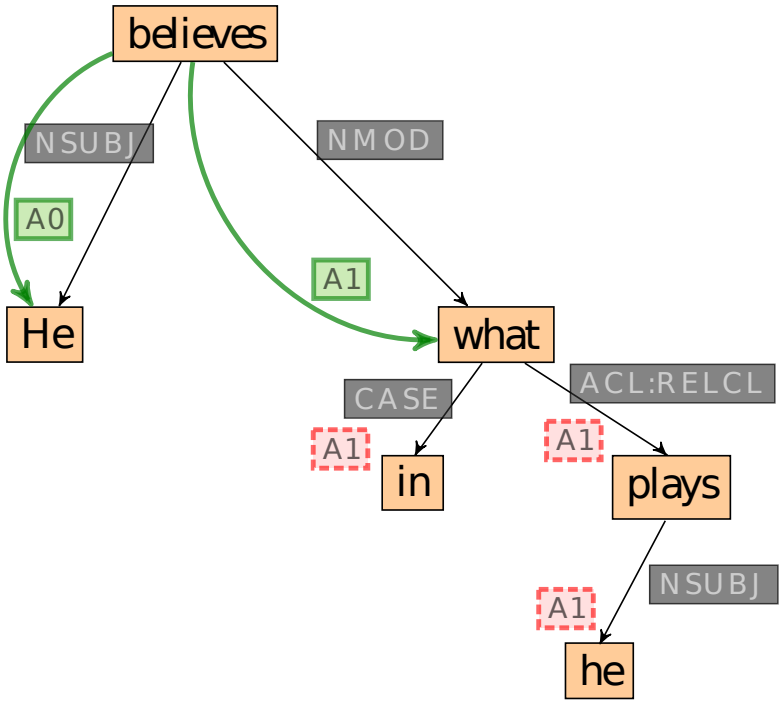


Figure 3.6: Tree visualization of SRL annotation

- **Syntactic Features** – part-of-speech tags of both predicate and argument, a position in a dependency tree (for example as a directed path from predicate to argument), a dependency relation of the argument, voice (active/passive) etc.
- **Lexical Features** – a lemma or a sense of both predicate and argument (or whole subtree).
- **Semantic Features** – Earlier mainly semantic clusters, nowadays word embeddings.

Argument Identification From the machine learning perspective, argument identification is a binary classification or tagging task to decide what subtrees are the argument of the predicate.

Role Labeling Role labeling can be formalized as a multi-class classification problem to determine the type of a semantic relation. The main problem here is that every predicate have quite different arguments. For example, *A2* role label of one predicate can have completely different semantic meaning from *A2* for other predicates. But if we treat every predicate completely independent, the data would be quite sparse.

Global Optimization SRL is closely bound with specific dependency annotations, and in the most datasets, it is designed in the way that every argument is the single subtree in the dependency tree of the sentence. As every label can be

assigned to only one node in the dependency tree, it should be beneficial to optimize the joined probability of all roles in the sentence. The global optimization step is often solved in the post-processing phase where we take the matrix of probabilities of assigning every label to every previously identified argument, assuming that individual role assignments are independent on each other for simplicity. The straightforward algorithm has exponential complexity. Some greedy solutions and heuristics are often used (for example, beam search).

3.3.2 Deep Learning

More recently, various neural network architectures have been used either to solve a separate subtask or whole SRL end-to-end.

Model	OntoNotes F1
He et al. (2018) + ELMo	85.5
He et al. (2017) + ELMo	84.6
Tan et al. (2018)	82.7
He et al. (2018)	82.1
He et al. (2017)	81.7

Table 3.2: SRL state-of-the-art results

He et al. (2017) proposed a standard LSTM model for SRL, based on BIO-tagging. They use a multi-layer bidirectional LSTM with gated highway (or residual) connections. In gated residual connection, the output is summed with the previous layer in the weighted way, where the weight of the linear combination is given by the residual gate. The gated residual connection operates on the inner state of LSTM, and it is computed in every timestep.

$$r_{l,t} = \sigma(W_r^l[h_{t-1}^l, x_t]) \tag{3.9}$$

$$h_{t,l} = r_{l,t} \cdot h'_{t,l} + (1 - r_{l,t}) \cdot W_h^l x_{t,l} \tag{3.10}$$

They use constraint A^* algorithm for the global optimization step (to incorporate dependencies between the output tags). They use three types of constraints:

1. **BIO constraints** – An output label sequence has to be a valid BIO sequence (that means the begin token of one role cannot be followed by the intermediate token of a different role)

2. **SRL constraints** – Constraints for Semantic Role Labeling proposed by Punyakanok et al. (2008).

   (a) *Unique Core Roles (U)* – Each core role (*ARG-0* - *ARG-5*, *ARG-A* should appear at most once for each predicate.

(b) *Continuation Roles (C)* – Continuation role C-X can exist only if its base role X is realized before it.

(c) *Reference Roles (R)* – Reference role R-X can appear only if its base role is realized.

3. **Syntactic constraints** – Penalizes arguments that are not constituents of a parse tree.

Tan et al. (2018) also use the BIO tagging, but they use a self-attention-based model, similar to the transformer encoder. The network architecture is shown in Figure 3.7. The Self-Attention block is the multi-head self-attention from the *Transformer* model. The authors did not use any constrained decoding. They state that the constraints only make the results worse.

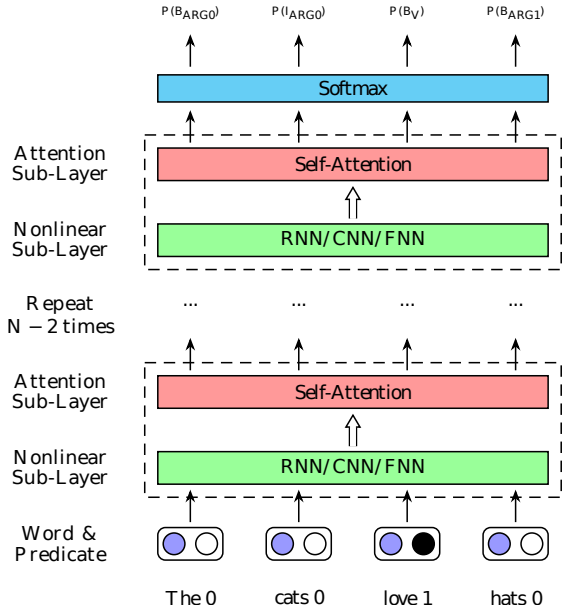


Figure 3.7: Depp SRL model architecture from Tan et al. (2018)

He et al. (2018) abandon the BIO tagging scheme, and they are rather predicting predicate-argument span tuples by searching through the possible combinations. They use a multi-layer bi-LSTM to produce contextualized representations of predicates and argument spans. Then they assign the role to the potential *predicate-argument* tuple.

Chapter 4

Cross-Lingual Semantics

Cross-lingual methods can be divided into three categories:

1. **Annotation Projection** methods try to transfer the annotation from one language into raw text corpus in a different language (typically using an alignment, some unsupervised techniques or the combination of both). For example, in cross-lingual semantic role labeling, Padó and Lapata (2009) use graph matching to project semantic roles to different language using the annotated parallel corpus.

2. **Model Transfer** methods are based on the assumption that input and output structures (and also meaning) are shared across the languages. If we train a model on data in one language with many annotated resources (typically English), we can use this model for prediction in different languages. Model transfer methods may also make it possible to train a model on multiple languages. In general, transfer learning is the machine learning technique where the model trained on one task is used to solve other tasks with some additional supervision.

3. **Unsupervised Methods** do not need any labeled data, thus many unsupervised methods are easy to make cross-lingual or at least easier than the supervised methods, where we need either annotated data in both languages or perfectly transferable input representations.

4.1 Bilingual and Multilingual Semantic Vectors

Bilingual and multilingual embeddings are currently used the most frequently for model transfer based cross-lingual model. Many different methods for creating such representations have been created recently. These methods can be divided into several categories:

1. **Linear transformation** methods have the two embedding spaces and dictionary on the input. They try to find the transformation matrix, so that word from the dictionary will be close to its translation after transformation.

2. **Joined optimization** methods train both languages jointly to share the embedding space from the beginning. The combination of monolingual and cross-lingual loss is used.

3. **Zero-shot learning** does not use any cross-lingual information and tries to transfer the knowledge from one language to another without any training. This approach is, of course, very limited.

4.1.1 Linear Transformations

The basic idea of linear transformations is that if we have the mapping of individual words (dictionary V) we can take monolingual semantic spaces and transform them linearly (translation, rotation, scaling) so that the word pairs in dictionary entries will be close to each other in the resulting space.

More formally, given semantic spaces of two languages A and B we are optimizing the transformation matrix T so that $\sum_i^{|V|} m(Ta_i, b_i)$ is maximal, where m is a similarity metric.

Least Squares Optimization

Mikolov et al. (2013b) proposed the most straightforward approach where the similarity metric is $\frac{1}{(a-b)^2}$, so the optimization objective is:

$$\underset{T}{\operatorname{argmin}} \sum_{i}^{|V|} (Ta_i - b_i)^2 \tag{4.1}$$

CCA

Faruqui and Dyer (2014) proposed embedding spaces transformation based on *canonical correlation analysis* (CCA). Basically, *CCA* seeks for the transformation matrices U and V so that the correlation $\rho(UA', VB')$ is maximized. A' and B' represent aligned word vectors from A and B according to vocabulary. It iteratively searches for basic vectors u and v according to criterion:

$$\underset{u_i, v_i}{\operatorname{argmax}} \ \rho(A \cdot u_i, B \cdot v_i) \tag{4.2}$$

so that $u_i^T u_j = 0, v_i^T v_j = 0 \forall j < i$ In other words, every canonical vector is orthogonal to all previous canonical vectors. The algorithm is a generalization of *PCA* for two multivariate random variables. Then we use projection matrices $A^* = UA'$ and $B^* = VB'$ where the spaces of A^* and B^* are shared. The schema of the *CCA* algorithm is shown in Figure 4.1.

Ammar et al. (2016) generalize the *CCA* into the true multilingual scenario where they project more than 50 languages into one common vector space.

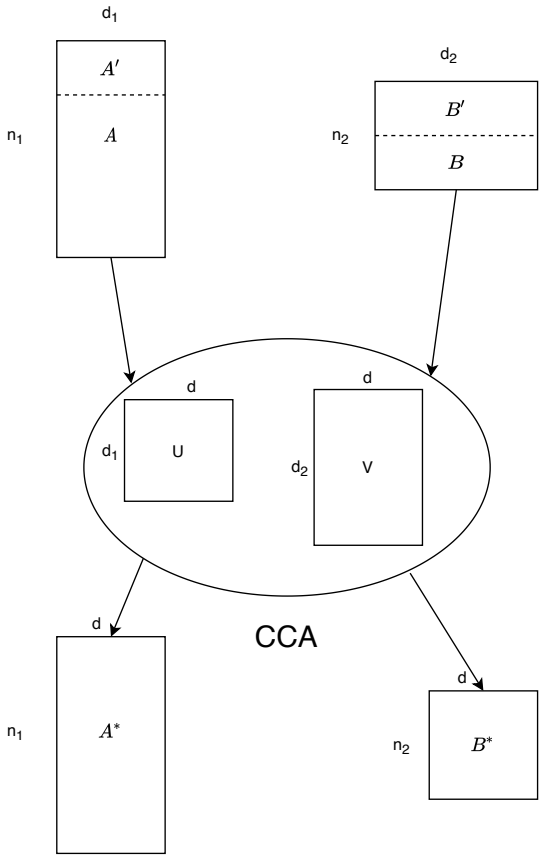


Figure 4.1: Illustration of CCA method

Orthogonal Transformation

Xing et al. (2015) propose orthogonal transformation based on inconsistencies observed between a monolingual embedding model objective, transformation objective and standard evaluation metric. The common objective of embedding models (like **Skip-Gram**) is to minimize negative log probability computed via softmax of the dot product of co-occurring vectors (see 3.2.3 for Equations) which is equivalent to maximizing the dot product of co-occurring vectors. The metric used for evaluating is often cosine similarity:

$$cos(w_1, w_2) = \frac{w_1 \cdot w_2}{||w_1|| \cdot ||w_2||} \tag{4.3}$$

which is simply normalized the dot product. The objective of a simple linear transformation is to minimize *MSE*, which is a very different objective. To avoid this, Xing et al. (2015) propose to normalize monolingual embeddings during training. After that, the dot product and cosine similarity are equivalent but yet Ta_i from the transformation objective (eq. 4.1.1) is not guaranteed to have unit length, so the other constraint is for transformation matrix T to be orthogonal. Transformation with an orthogonal matrix does not rescale the space (it just rotates the space), so it preserves the dot product of the projected vectors. In

Xing et al. (2015), they make the transformation matrix orthogonal by rescaling via SVD: $T' = UV^T, T = U\Sigma V^T$ They also propose to change the minimization of squares to the maximization of the dot products, but as Artetxe et al. (2016) show, it does not make any difference.

Later Artetxe et al. (2016) propose an improvement of this method and explain how it is related to other methods. They propose an analytical method to find the global optimum of orthogonal transformation objective:

$$T = VU^T, B^T A = U\Sigma V^T \tag{4.4}$$

They also propose dimension-wise mean centering of vector spaces to make expected cosine of two random words equal to 0.

Max-Margin Optimization

Lazaridou et al. (2015) pointed out the problem of *hubness* in cross-lingual embeddings mapping. It is one of the problems called *the curse of dimensionality*, where the higher the dimensionality of the space is, the more similar the pair-wise distances of its elements are. In an extreme scenario of a very high-dimensional space, all the pair-wise distances are practically the same. In semantic spaces, some words (hubs) occur in the nearest neighbors set of many completely different words. Lazaridou et al. (2015) find out that with the least-squares mapping, the *hubness* of projected space is much worse than the *hubness* of the original spaces. They propose the max-margin loss for linear transformation to mitigate this problem:

$$J = \sum_{j \neq i}^{k} max(0, \gamma + dist(\hat{y}_i, y_i) - dist(\hat{y}_i, y_j)) \tag{4.5}$$

where hyper-parameter γ is the margin, and k is the number of negative examples (can be sampled randomly from a uniform distribution). Intuitively the loss is forcing the transformation matrix to make the translation more similar than any other word, and the margin controls how much more.

4.1.2 Joined Optimization

Joined optimization methods train the embeddings for both languages simultaneously. They often need a parallel corpus to find cross-lingual relations. The idea here is that knowledge from one language can help to learn something interesting for another language (with much less data, for example), and the embeddings might be better than the monolingual ones. This is just the special case of multi-task learning, where instead of different tasks, we have the same task in different languages.

Bilingual Skip-Gram

Luong et al. (2015b) proposed the modification of the *Skip-gram* model for training on sentence-aligned bilingual data. The objective is to predict the context in both languages, given the central word in one of them. This objective is optimized for both languages.

4.1.3 Unsupervised Transfer

Unsupervised Transformations

There are several attempts to map two semantic spaces to each other without any explicit dictionary nor aligned corpora. In theory, we can do it by minimizing the difference between pair-wise distances through all the words, although it is not very effective in practice.

Artetxe et al. (2017) use the dictionary adaptation method for finding the transformation matrix with a very small seed dictionary. The algorithm repeats two steps until it converges:

1. Compute the optimal orthogonal mapping maximizing the similarities for the current dictionary D:

$$\underset{W_x, W_z}{\operatorname{argmax}} \sum_i \sum_j D_{ij}((X_{i*}W_x) \cdot (Y_{j*}W_z)) \tag{4.6}$$

   An optimal solution is given by $W_x = U$ and $W_z = V$ where $USV^T = X^T DZ$ is singular value decomposition.

2. Compute the optimal dictionary over the similarity matrix of the mapped embeddings $XW_X W_Z^T Z^T$. The nearest neighbors are taken as the dictionary words (argmax over rows and columns).

Later Artetxe et al. (2018) extend this approach to completely unsupervised by building the seed dictionary automatically. The seed dictionary is created by first computing the similarity matrices of X and Z: $M_X = XX^T$, then they sort the elements of each row of M_X and M_Z. After sorting, the best translations should correspond to the most similar rows of the matrices.

Zero-Shot Learning with BERT

(Devlin et al., 2018) BERT is not using any cross-lingual information during training. Their multilingual model is trained on many monolingual corpora. The authors show that supervised tasks can be solved in a zero-shot scenario very well with this multilingual (but not generally cross-lingual) model.

Wu and Dredze (2019) performed a large number of experiments on various tasks for many languages to evaluate zero-shot cross-lingual transferability of the BERT model. Their experiments show that BERT achieves competitive results in the zero-shot transfer.

Pires et al. (2019) empirically investigate cross-lingual properties of multilingual BERT. For example, they show that when we use English BERT for other languages in a zero-shot setting, it achieves poor results for languages with small vocabulary overlap. However, in the case of multilingual BERT, there is a very weak correlation between vocabulary overlap and zero-shot performance.

Artetxe et al. (2019) further investigate the cross-linguality of BERT. They proved that its cross-lingual properties do not come primarily from shared words (lexically the same). They design the model where in the first step they train English BERT and then they train new embeddings for another language keeping the rest of the model weights frozen. They found that this approach achieves only slightly worse results in the zero-shot cross-lingual transfer. Their approach cannot learn from lexically the same words, because it does not share the vocabulary between languages. They also show that word-level cross-lingual mappings perform really poorly in comparison with a deep model like BERT.

4.2 Parallel Corpora and Machine Translation

Many cross-lingual methods need aligned parallel corpora to be trained on. Mostly for the joined optimization techniques, we need sentence aligned parallel corpus, but for example, even for word-level mapping, the supervised dictionaries are often created from machine translation alignments on parallel corpora.

- Probably the most well-known parallel corpus is *Eutoparl* (Koehn, 2005).
- For Czech-English cross-lingual methods, we have *CzEng* (Bojar et al., 2016).
- Another interesting cross-lingual resource for Czech-English is *Prague Czech-English Dependency Treebank* (Hajič et al., 2012), which is the supervised parallel corpus hand-annotated with many syntactic and semantic features.

4.3 Universal Dependencies and Other Cross-Lingual Resources

The *Universal Dependencies (UD)* (Nivre et al., 2016) is a framework for consistent annotation of grammar (parts of speech, morphological features, and syntactic dependencies) across different human languages. UD is an open community effort with over 300 contributors producing more than 150 treebanks in 90 languages. The UD annotation scheme evolved as a compilation of three universal annotation principles: universal dependency relations from Stanford (de Marneffe and Manning, 2008), part-of-speech tags from Google (Petrov et al., 2012), and morphological features (Zeman, 2008) from UFAL, Charles University.

The design of UD annotations is, in principle, similar to SD[1] annotations, but it differs in some crucial aspects. From the SRL point of view, the most

[1]Standard language-specific annotations, for example Penn Treebank annotations.

critical difference is making content words the heads. In SD annotation for many languages, the auxiliary verbs, prepositions, and conjunctions are often the heads.

There are several frameworks that enable end-to-end parsing into UDs: UDpipe (Straka et al., 2016), Stanford CoreNLP (Manning et al., 2014), Malt parser (Nivre and Hall, 2005), and others.

4.4 Cross-Lingual SRL

In semantic role labeling, all three types of cross-lingual methods have been developed.

4.4.1 Annotation Projection

Annotation projection methods make use of a parallel corpus. The corpus is annotated in one language (by annotators or by a language-dependent automatic system), and this annotation is projected into other languages through alignments for machine translation.

Padó and Lapata (2009) propose the annotation projection from English to German. The authors use a parallel corpus with either the gold alignments or with the automatic ones produced by *GIZA++* (Och and Ney, 2003). The general objective is given by:

$$\hat{A} = \underset{A \in \mathcal{A}}{\operatorname{argmin}} \sum_{(u_s, u_t) \in A} weight(u_s, u_t) \tag{4.7}$$

where $\hat{A}$ is the best alignment, $\mathcal{A}$ is the set of all possible alignments, and u_s and u_t are the source and the target node, where

$$weight(u_s, u_t) = -\log\left(sim(u_s, u_t)\right) \tag{4.8}$$

The baseline algorithm transfers a role between two words if the words are aligned. More complex algorithms work with constituents instead of single words. The authors use matching in a bipartite graph. In post-processing, they apply several filters:

1. filling the gaps – adding the words as arguments so that the argument would be a continuous sequence of words;
2. word filter – removing some words with simple rules.

Annesi and Basili (2010) use a similar approach and extend it with an HMM model to increase the transfer accuracy.

4.4.2 Unsupervised Approaches

Grenager and Manning (2006) deploy unsupervised learning using the EM algorithm based upon a structured probabilistic model of the domain. Lang and Lapata (2011) discover arguments of verb predicates with high accuracy using a small set of rules. A split-merge clustering is consequently applied to assign (nameless) roles to the discovered arguments. Titov and Klementiev (2012a) propose a superior argument clustering by using the *Chinese restaurant process* and Titov and Klementiev (2012b) use this model in the cross-lingual scenario, where they add cross-lingual agreement on the parallel data to the cost function.

Woodsend and Lapata (2015) use *CBOW* based FFNN to learn embeddings for predicates and arguments. First, they identify arguments according to linguistic rules from Lang and Lapata (2011). Then they train a FFNN to predict an argument given the predicate and its other arguments within the context window of the size given by the hyper-parameter. In the input, a predicate is represented by the one-hot vector, and arguments are represented with syntactic features (dependency relation, POS...) and with the semantic embedding of the argument root word. Context vectors are weighted with a single matrix. After that, they are concatenated resulting in the final context representation v_c. The central word is weighted with a different matrix resulting in v_0. The network architecture is shown in Figure 4.2. Probability of the central word is given by:

$$P(a_c|a_{context}, b) = v_c v_0 \tag{4.9}$$

as a cost function, they use standard negative log probability:

$$E = -\frac{1}{T}\sum_{i=0}^{T} P(a_i|a_{context}, b_i) \tag{4.10}$$

The final vector representations are then clustered with linear programming formulation of hierarchical clustering, where they can model task-specific knowledge.

4.4.3 Model Transfer

Kozhevnikov and Titov (2013) use cross-lingual word mappings and cross-lingual semantic clusters obtained from parallel corpora, and cross-lingual features extracted from unlabelled syntactic dependencies to create a cross-lingual SRL system. In (Kozhevnikov and Titov, 2014), they try to find a mapping between language-specific models using parallel data automatically.

Our approach (Pražák and Konopík, 2017) belongs among the model transfer approaches. We introduce a first attempt to use Universal Dependencies as cross-language features in SRL. Most of the state-of-the-art approaches to SRL rely on lexical features (e.g. word lemmas). In the cross-language scenario, such features require bilingual models (e.g. word mapping via machine translation or bilingual clusters).

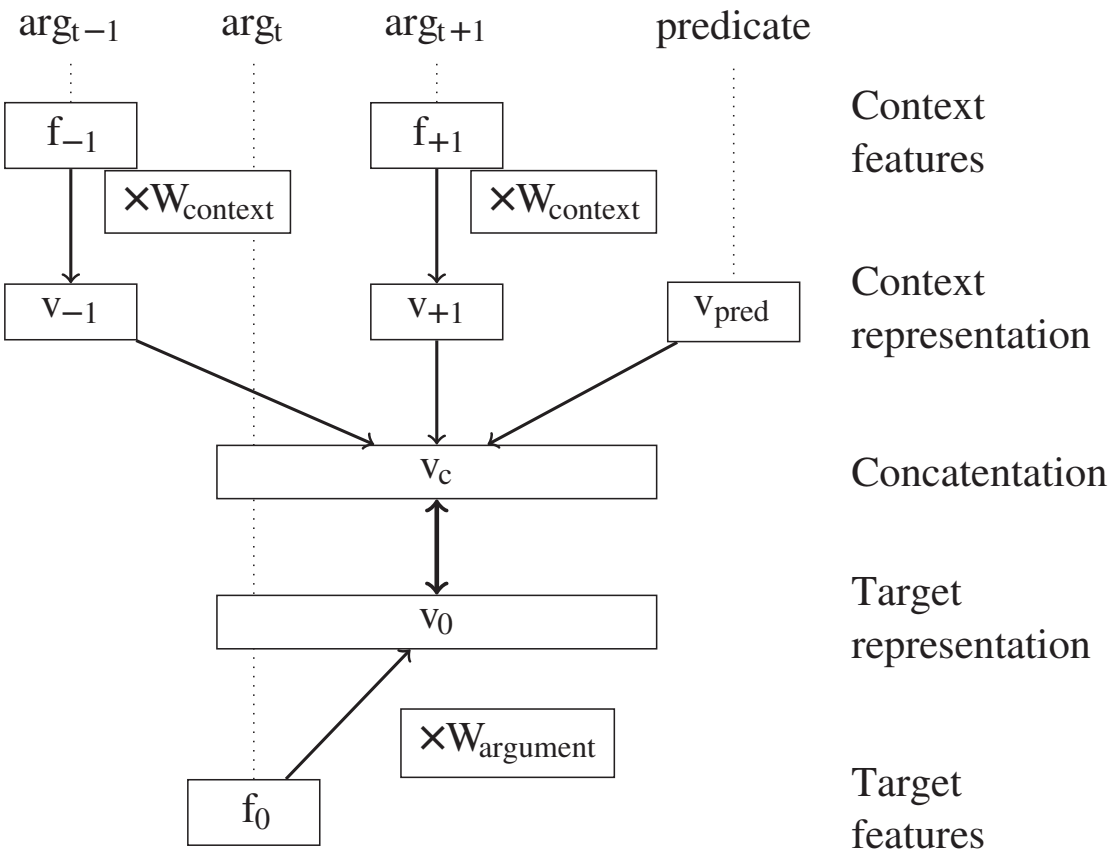


Figure 4.2: Network Architecture (from Woodsend and Lapata (2015))

Chapter 5

Preliminary Experiments and Future Work

As the first experiment, we studied and evaluated word-level semantic methods (Konopík and Pražák, 2015), where we compared formal and distributional methods. Later we participated in *SemEval 2016 Task 2: Interpretable Semantic Textual Similarity* (Konopík et al., 2016), where the task was to analyze deeper semantic relationships between two sentences. We have won one of two sub-tasks. More recently, we have participated in another challenging semantic task, *SemEval 2020 Task 1: Unsupervised Lexical Semantic Change Detection* (Pražák et al., 2020) where we analyzed semantic change of words through time. The task was to:

1. Classify if there is a change of the meaning of words between two corpora from different time periods (binary classification).

2. Rank the word according to their degree of semantic change between the two time periods.

We used linear transformation of embedding spaces of both time span corpora, and then we compared representations of the target words in resulting spaces for the ranking task.

We ranked 1^{st} in the sub-task 1 and 4^{th} in the sub-task 2.

Nowadays, there are many methods for creating deep semantic representations. From the cross-lingual perspective, the most interesting is multilingual BERT. It has been empirically shown that it has an unexpectedly high cross-lingual performance. However, multilingual BERT has its limits in the case of low resource languages. One problem of multilingual BERT is- the joined vocabulary pruned by token frequency. In this way, the low-resource languages lose almost all tokens in favor of English. We believe there is much more to improve in current state-of-the-art semantic representation to perform better in minor languages.

Another unresolved issue is how to represent longer text and capture long-distant dependencies (like whole documents) with deep models. In Konopík and

Pražák (2018), we successfully injected global information obtained from *Latent Dirichlet Allocation* into a deep learning model, which significantly improved the performance of the Named Entity Recognition system. This experiment shows that the ability of deep neural networks to capture long-distant dependencies is at least limited. Another problem is that the current models are not able to process long texts at all. In recurrent neural network due to its sequential processing, it is able to process sequences of around 100-200 tokens. Transformer-based models have no such restriction in theory, but still, they are trained on either single sentence or on a sentence pair, and they would not be very effective in processing long inputs. Most current state-of-the-art methods use the language model as an unsupervised objective, which does not have many long-distance dependencies. There are at least two ways how to deal with long inputs:

1. Analyze long-distant dependencies in an unsupervised way and then use the result in a deep model (like we did in Konopík and Pražák (2018))
2. Modify the deep model to be able to deal with long inputs itself (the details are discussed in Section 3.2.5).

Most of the cross-lingual representations have been created and evaluated in a bilingual scenario. There are not many attempts to create joint cross-lingual representation for many languages. In Pražák and Konopík (2017), we created the joint cross-lingual model for Semantic Role Labeling evaluated on four languages. Later in Pražák and Konopík (2019), we extended this approach, and we created the system capable of producing SRL annotations for 51 languages. We plan to extend this approach in the future.

5.1 Aims of the PhD thesis

The goal of the doctoral thesis is to propose novel methods for cross-lingual semantic representations. The work will be focused on the following research tasks:

- Propose a new method to deal with semantics of long texts and its long-term dependencies.
- Adapt current cross-lingual methods and try to improve performance in Czech or other minor languages[1].
- Propose a method capable of creating a joint semantic representation for many languages.

[1]Czech and languages with similar amount of textual data available. For example in the number of articles in Wikipedia, Czech is approximately 13x smaller than English (463,175 x 6,166,566 according to Wikipedia statistics, 30.10.2020.)

List of Tables

List of Figures

List of Equations

Bibliography

Ammar, W., Mulcaire, G., Tsvetkov, Y., Lample, G., Dyer, C., and Smith, N. A. (2016). Massively multilingual word embeddings. *arXiv preprint arXiv:1602.01925*.

Annesi, P. and Basili, R. (2010). Cross-lingual alignment of FrameNet annotations through hidden markov models. In *Proceedings of the 11th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'10, pages 12–25, Berlin, Heidelberg. Springer-Verlag.

Artetxe, M., Labaka, G., and Agirre, E. (2016). Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2289–2294.

Artetxe, M., Labaka, G., and Agirre, E. (2017). Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462, Vancouver, Canada. Association for Computational Linguistics.

Artetxe, M., Labaka, G., and Agirre, E. (2018). A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 789–798.

Artetxe, M., Ruder, S., and Yogatama, D. (2019). On the cross-lingual transferability of monolingual representations. *arXiv preprint arXiv:1910.11856*.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3:1137–1155.

Blei, D. M., Ng, A. Y., Jordan, M. I., and Lafferty, J. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:2003.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

Bojar, O., Dušek, O., Kocmi, T., Libovický, J., Novák, M., Popel, M., Sudarikov, R., and Variš, D. (2016). CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In Sojka, P., Horák, A., Kopeček, I., and Pala, K., editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 231–238, Cham / Heidelberg / New York / Dordrecht / London. Masaryk University, Springer International Publishing.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Caruana, R. (1997). Multitask learning. *Machine learning*, 28(1):41–75.

Chang, W.-C., Yu, F. X., Chang, Y.-W., Yang, Y., and Kumar, S. (2020). Pre-training tasks for embedding-based large-scale retrieval. *arXiv preprint arXiv:2002.03932*.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734.

de Marneffe, M.-C. and Manning, C. D. (2008). The Stanford Typed Dependencies representation. In *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, CrossParser '08, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

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

Faruqui, M. and Dyer, C. (2014). Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 462–471.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational linguistics*, 28(3):245–288.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356.

Graves, A., rahman Mohamed, A., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks.

Grenager, T. and Manning, C. D. (2006). Unsupervised discovery of a statistical verb lexicon. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hajič, J., Hajičová, E., Panevová, J., Sgall, P., Cinková, S., Fučíková, E., Mikulová, M., Pajas, P., Popelka, J., Semecký, J., Šindlerová, J., Štěpánek, J., Toman, J., Urešová, Z., and Žabokrtský, Z. (2012). Prague czech-english dependency treebank 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

He, L., Lee, K., Levy, O., and Zettlemoyer, L. (2018). Jointly predicting predicates and arguments in neural semantic role labeling. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 364–369.

He, L., Lee, K., Lewis, M., and Zettlemoyer, L. (2017). Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 473–483.

Henderson, M., Al-Rfou, R., Strope, B., Sung, Y.-H., Lukács, L., Guo, R., Kumar, S., Miklos, B., and Kurzweil, R. (2017). Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

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

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. (2016). Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*.

Kaiser, Ł. and Sutskever, I. (2015). Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Konopík, M. and Pražák, O. (2015). Information sources of word semantics methods. In *International Conference on Speech and Computer*, pages 243–250. Springer.

Konopík, M. and Pražák, O. (2018). Lda in character-lstm-crf named entity recognition. In *International Conference on Text, Speech, and Dialogue*, pages 58–66. Springer.

Konopík, M., Pražák, O., Steinberger, D., and Brychcín, T. (2016). Uwb at semeval-2016 task 2: Interpretable semantic textual similarity with distributional semantics for chunks. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 803–808.

Kozhevnikov, M. and Titov, I. (2013). Cross-lingual transfer of semantic role labeling models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, ACL 2013, 4-9 August 2013, Sofia, Bulgaria, Volume 1: Long Papers*, pages 1190–1200.

Kozhevnikov, M. and Titov, I. (2014). Cross-lingual model transfer using feature representation projection. In *ACL (2)*, pages 579–585.

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. In *International Conference on Learning Representations*.

Lang, J. and Lapata, M. (2011). Unsupervised semantic role induction via split-merge clustering. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 1117–1126. Association for Computational Linguistics.

Lazaridou, A., Dinu, G., and Baroni, M. (2015). Hubness and pollution: Delving into cross-space mapping for zero-shot learning. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 270–280.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, J., Luong, M.-T., Jurafsky, D., and Hovy, E. (2015). When are tree structures necessary for deep learning of representations? *arXiv preprint arXiv:1503.00185*.

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. (2015a). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Luong, T., Pham, H., and Manning, C. D. (2015b). Bilingual word representations with monolingual quality in mind. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 151–159.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Miller, G. A. (1998). *WordNet: An electronic lexical database*. MIT press.

Moschitti, A., Pighin, D., and Basili, R. (2008). Tree kernels for semantic role labeling. *Computational Linguistics*, 34(2):193–224.

Mou, L., Peng, H., Li, G., Xu, Y., Zhang, L., and Jin, Z. (2015). Discriminative neural sentence modeling by tree-based convolution. *arXiv preprint arXiv:1504.01106*.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R. T., Petrov, S., Pyysalo, S., Silveira, N., et al. (2016). Universal dependencies v1: A multilingual treebank collection. In *LREC*.

Nivre, J. and Hall, J. (2005). Maltparser: A language-independent system for data-driven dependency parsing. In *In Proc. of the Fourth Workshop on Treebanks and Linguistic Theories*, pages 13–95.

Och, F. J. and Ney, H. (2003). A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51.

Padó, S. and Lapata, M. (2009). Cross-lingual annotation projection for semantic roles. *Journal of Artificial Intelligence Research*, 36:307–340.

Parikh, A. P., Täckström, O., Das, D., and Uszkoreit, J. (2016). A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. (2013). How to construct deep recurrent neural networks.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Petrov, S., Das, D., and McDonald, R. (2012). A universal part-of-speech tagset. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA).

Pires, T., Schlinger, E., and Garrette, D. (2019). How multilingual is multilingual bert? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001.

Pražák, O. and Konopík, M. (2017). Cross-lingual srl based upon universal dependencies. In *RANLP*, pages 592–600.

Pražák, O. and Konopík, M. (2019). Ulsana: Universal language semantic analyzer. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 967–972.

Pražák, O., Přibáň, P., Tailor, S., and Sido, J. (2020). Uwb at semeval-2020 task 1: Lexical semantic change detection. In *Proceedings of the 14th International Workshop on Semantic Evaluation (SemEval-2020)*.

Punyakanok, V., Roth, D., and Yih, W.-t. (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2):257–287.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.

Ruder, S. (2017). An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Socher, R., Bauer, J., Manning, C. D., et al. (2013). Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 455–465.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Straka, M., Hajič, J., and Straková, J. (2016). UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, pos tagging and parsing. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, Paris, France. European Language Resources Association (ELRA).

Tai, K. S., Socher, R., and Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Tan, Z., Wang, M., Xie, J., Chen, Y., and Shi, X. (2018). Deep semantic role labeling with self-attention. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Titov, I. and Klementiev, A. (2012a). A bayesian approach to unsupervised semantic role induction. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '12, pages 12–22, Stroudsburg, PA, USA. Association for Computational Linguistics.

Titov, I. and Klementiev, A. (2012b). Crosslingual induction of semantic roles. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, Jeju Island, South Korea. Association for Computational Linguistics.

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.

Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688.

Woodsend, K. and Lapata, M. (2015). Distributed Representations for Unsupervised Semantic Role Labeling. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, {EMNLP} 2015, Lisbon, Portugal, September 17-21, 2015*, pages 2482–2491.

Wu, S. and Dredze, M. (2019). Beto, bentz, becas: The surprising cross-lingual effectiveness of bert. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 833–844.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xing, C., Wang, D., Liu, C., and Lin, Y. (2015). Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1006–1011.

Zeman, D. (2008). Reusable tagset conversion using tagset drivers. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.