Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Comparing word vectors in favor of lexical semantics

Bachelor Thesis

2023
Barbora Vicianová

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Comparing word vectors in favor of lexical semantics

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Computer Science
Supervisor:	Mgr. Endre Hamerlik

Bratislava, 2023
Barbora Vicianová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Barbora Vicianová
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Comparing word vectors in favor of lexical semantics
Porovnanie slovných vektorov z hľadiska lexikálnej sémantiky

Anotácia: Predmetná práca má pomôcť vyhodnotiť psychologické experimenty, kde autori skúmajú sémantickú pamäť a vybavovanie pojmov. [1]
V týchto experimentoch participanti zvyčajne generujú slovné odpovede na slovné podnety podľa určitých pravidiel. Výstup z týchto kognitívnych úloh je množina slov), ktoré sú buď vo forme slovných párov resp. "edgelistov" (Podnet->Odpoveď).

Cieľ: Cieľom predmetnej bakalárskej práce je vytvoriť softvérové riešenie (dielo), ktoré bude schopné vyhodnocovať sémantickú podobnosť slovných párov na základe podobnosti embeddingov (vektorových reprezentácii):
statických Word2vec [2]
a kontextuálnych Slovak BERT [3]

Literatúra: [1] Marko, M., Michalko, D., Dragašek, J., Vančová, Z., Jarčušková, D., & Riečanský, I. (2022). Assessment of automatic and controlled retrieval using verbal fluency tasks. Assessment, 10731911221117512.
[2] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2017). Advances in pre-training distributed word representations. arXiv preprint arXiv:1712.09405.
[3] Pikuliak, M., Grivalský, Š., Konôpka, M., Blšták, M., Tamajka, M., Bachratý, V., ... & Uhlárik, F. (2021). SlovakBERT: Slovak Masked Language Model. arXiv preprint arXiv:2109.15254.

Kľúčové slová: vektorová sémantika, lexikálna sémantika, statické embeddingy, kontextuálne embeddingy, verbálna plynulosť

Vedúci: Mgr. Endre Hamerlik
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 03.10.2022

Dátum schválenia: 04.11.2022 doc. RNDr. Damas Gruska, PhD.
garant študijného programu

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

....

študent vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Barbora Vicianová
Study programme:	Applied Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Comparing word vectors in favor of lexical semantics

Annotation: Present work is intended to help evaluate psychological experiments where the authors investigate semantic memory and concept recall. [1]
In these experiments, participants usually generate verbal responses to verbal stimuli according to certain rules. The output of these cognitive tasks is a set of words, which are in the form of word pairs/edgelist (Stimulus->Response).

Aim: The goal of the subject bachelor's thesis is to create a software solution that will be able to evaluate the semantic similarity between pairs of words based on the similarity of vector representations of verbal responses using:
- static embeddings (Word2vec [2])
- and contextual embeddings as well (Slovak BERT [3])

Literature: [1] Marko, M., Michalko, D., Dragašek, J., Vančová, Z., Jarčušková, D., & Riečanský, I. (2022). Assessment of automatic and controlled retrieval using verbal fluency tasks. Assessment, 10731911221117512.
[2] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., & Joulin, A. (2017). Advances in pre-training distributed word representations. arXiv preprint arXiv:1712.09405.
[3] Pikuliak, M., Grivalský, Š., Konôpka, M., Blšták, M., Tamajka, M., Bachratý, V., ... & Uhlárik, F. (2021). SlovakBERT: Slovak Masked Language Model. arXiv preprint arXiv:2109.15254.

Keywords: vector semantics, lexical semantics, static embeddings, contextual embeddings, verbal fluency

Supervisor:	Mgr. Endre Hamerlik
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	prof. Ing. Igor Farkaš, Dr.
Assigned:	03.10.2022
Approved:	04.11.2022

doc. RNDr. Damas Gruska, PhD.
Guarantor of Study Programme

Student

Supervisor

Acknowledgments: I would like to express my sincere gratitude to my supervisor, Mgr. Endre Hamerlik, for his invaluable guidance, support, and useful advice throughout the writing of my bachelor's thesis. I am particularly grateful for the consultations and the valuable feedback that he provided. In addition, I extend my heartfelt thanks to my family for their support during my studies.

Abstrakt

Cieľom tejto bakalárskej práce je vyhodnotiť sémantickú podobnosť slovných párov pomocou vektorových reprezentácií slov a tak pomôcť pri vyhodnocovaní psychologických experimentov skúmajúcich sémantickú pamäť a vybavovanie pojmov. V tejto práci porovnávam statické embeddingy slov generovaných pomocou Word2vec a kontextuálne embeddingy generovaných pomocou Slovak BERT, aby sme zistili, ktoré sú vhodnejšie na vyhodnocovanie sémantických podobnosti. Použila som tri rôzne metódy na vyhodnotenie sémantickej podobnosti slovných párov. V prvej som vyhodnotila euklidovské vzdialenosti a kosínusovú podobnosť embeddingov slovných párov a porovnali sme priemerné vzdialeností sémanticky súvisiacich a nesúvisiacich slovných párov. Druhá metóda spočívala v implementácii viacvrstvového perceptrónu na klasifikáciu súvisiacich a nesúvisiacich slovných párov. V tretej metóde som použila ďalší viacvrstvový perceptron na klasifikáciu slov podľa ich sémantickej kategórie. Prvá kapitola poskytuje prehľad o relevantných konceptov, ako sú umelé neurónové siete, word2vec, Slovak BERT a iné. Ďalej popisujem použité metódy na vyhodnotenie sémantickej podobnosti, a predstavujem výsledky práce. V záverečnej časti práce porovnávam výsledky, pričom navrhujem aj potenciálne oblasti ďalšieho výskumu.

Kľúčové slová: vektorová sémantika, lexikálna sémantika, statické embeddingy, kontextuálne embeddingy, verbálna plynulosť

Abstract

This bachelor's thesis aims to evaluate the semantic similarity between pairs of words using vector representations in order to assist in evaluating psychological experiments that investigate semantic memory and concept recall. This work compares the effectiveness of static embeddings, generated by Word2vec, and contextual embeddings, generated by Slovak BERT, to identify appropriate word representations for evaluating semantic similarity. Three different methods were used to evaluate the semantic similarity of word pairs. The first method involved evaluating the Euclidean distances and cosine similarity of the embeddings and comparing the average distance of associated and dissociated word pairs. The second experiment involved implementing a Multilayer Perceptron (MLP) to classify associated and dissociated word pairs. The third method used another MLP to classify the words according to their category. The first chapter provides an overview of relevant topics, such as Artificial Neural Networks, word2vec, Slovak BERT, and others. Next, I describe the methods used to evaluate semantic similarity and present the results of the study. The final part of the work compares and discusses the results, suggesting potential areas for future research.

Keywords: vector semantics, lexical semantics, static embeddings, contextual embeddings, verbal fluency

Contents

List of Figures

List of Tables

Introduction

The study of semantic memory and concept recall is an essential aspect of cognitive psychology, as it provides valuable insights into the nature of human memory and language processing. In psychological experiments, participants are required to generate verbal responses to verbal stimuli based on certain rules, which results in a set of word pairs. These word pairs are in the form of a stimulus-response relationship and are used to evaluate the semantic similarity between pairs of words.

The aim of this work is to evaluate the semantic similarity between pairs of words based on the similarity of vector representations of verbal responses. I used two types of embeddings in my study: static and contextual embeddings, discussed in Section 1.2 and Subsection 1.6.2.

I conducted three experiments to evaluate the effectiveness of using vector semantics in evaluating the semantic similarity between pairs of words. Firstly, I calculate the Euclidean distances between the embeddings of words and compare the average distance of the associated and dissociated word pairs. Secondly, I implemented a Multilayer Perceptron, presented in MLP, presented in Subsection 1.1.2, to classify the similarity of word pairs. Thirdly, I implemented an MLP, which classified the words according to their category.

Overall, this thesis has important implications for the fields of human sciences dealing with semantic similarities of textual data, as it highlights the potential of using vector semantics in the evaluation of semantic memory and concept recall.

I have structured my work into distinct chapters, each focusing on specific aspects. In Chapter 1, I explore the topics that hold relevance to my research. The methodologies employed to evaluate semantic similarity are explained in detail in Chapter 2. The outcomes and findings of my research are presented in Chapter 3. In the concluding sections of my work, specifically in Sections 3.3 and 3.3, I engage in a comprehensive comparison and critical analysis of the obtained results. Additionally, I discuss potential areas for future research and exploration.

Chapter 1

Related work

1.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computing systems that are inspired by the structure and function of the human brain. ANNs have become increasingly popular in recent years due to their ability to learn complex patterns and make accurate predictions on a wide range of tasks, such as image classification, speech recognition, and natural language processing. In this chapter I write about the basic structure of ANN, training methods and optimization, based on [6] and [24].

The structure of ANNs is based on interconnected nodes called neurons, which are organized into layers. These artificial neurons are conceptually derived from biological neurons in the human brain, which receive and transmit electrical signals. In ANNs, each neuron receives inputs from neurons in the previous layer, processes them using weights, and sends output signals to neurons in the next layer. The weights determine the impact of each neuron on the output of the next layer, and they are updated during the training process to optimize the network's performance.

1.1.1 Perceptron

The perceptron is a fundamental building block of neural networks. It is a simple mathematical model of a biological neuron, inspired by the way neurons in the brain work. The perceptron takes a set of inputs, applies weights to them, and produces an output based on a defined activation function.

The perceptron model was first introduced by Frank Rosenblatt in 1957 [19]. It served as the basis for the development of more complex neural network architectures, including the multi-layer perceptron (MLP).

1.1.2 Multi-Layer Perceptron (MLP)

The multi-layer perceptron (MLP) is a fully connected feedforward artificial neural network [24]. It extends the basic concept of the perceptron by incorporating multiple layers of interconnected neurons. An MLP typically consists of an input layer, one or more hidden layers, and an output layer. Each neuron in a layer is connected to all the neurons in the subsequent layer, allowing information to flow forward through the network during the training and inference processes.

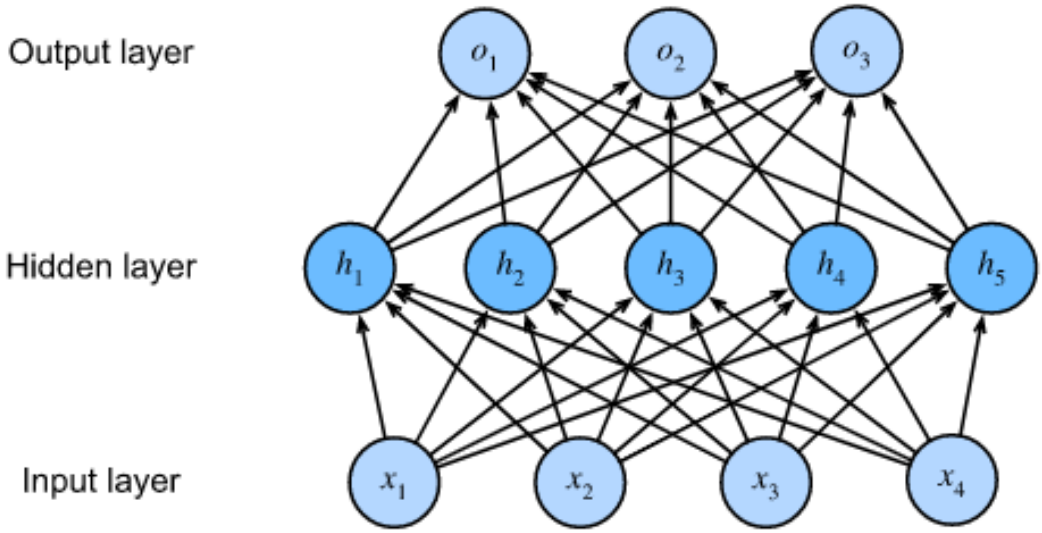


Figure 1.1: An MLP with a hidden layer of 5 hidden units. [24].

Batch refers to a subset or group of input examples that are processed together during training. Instead of processing the entire dataset at once, it is common practice to divide the dataset into smaller batches and process them sequentially.

Let see an example: the matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ denotes a batch of n input examples, each having d input features (d being 4 in the case of the example figure 1.1).

For the MLP with one hidden layer (as is our example 1.1) with h hidden units, the outputs of the hidden layer are denoted by $\mathbf{H} \in \mathbb{R}^{n \times h}$. By having both hidden and output layers fully connected, the weights $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ and biases $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$ for the hidden layer, as well as the weights $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ and biases $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$ for the output layer can be obtained. This enables us to compute the outputs $\mathbf{O} \in \mathbb{R}^{n \times q}$ of the one-hidden-layer MLP as follows:

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}, \mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \tag{1.1}$$

It can be proven that simply adding a hidden layer does not change the fact that the model can only approximate affine functions using linear operations (for a detailed explanation and proof, refer to [24]).

Therefore, it is expected to use a non-linear activation function, denoted by σ, which is applied to each hidden unit after the affine transformation. The outputs of activation functions are called activations. With activation functions in place, it is no longer possible to collapse our MLP into a linear model:

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}), \mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \tag{1.2}$$

Because each row in $\mathbf{X}$ corresponds to an example in the batch, The activation function σ is defined to be applied to the inputs of each row individually, i.e., one example at a time, as specified in [24].

1.1.3 Activations

Activation functions play a key role in determining whether a neuron should be activated or not, based on the weighted sum of inputs plus a bias term. They are differentiable operators that transform input signals to outputs, with many of them adding non-linearity. Given their importance to deep learning, it is useful to examine some commonly used activation functions.

Sigmoid Function

The sigmoid function [24] is a commonly used activation function in artificial neural networks. It maps the input values to outputs between 0 and 1, hence it is often referred to as a squashing function.

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{1.3}$$

Sigmoid activation functions are mostly applied on output units for binary classification problems, where outputs are interpreted as probabilities.

Tanh Function

Similar to the sigmoid function, the hyperbolic tangent (*tanh*) function [24] also squashes its input values, mapping them to the range between -1 and 1.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{1.4}$$

The function is similar in shape to the sigmoid function, but it has point symmetry about the origin of the coordinate system and is bipolar in nature. This means that it can both activate and inhibit certain neurons. While the sigmoid function is commonly used as an activation function for output layers in binary classification problems, the *tanh* function is more frequently utilized in hidden layers due to its ability to model complex non-linear relationships between input and output.

ReLU Function

One of the most widely used activation functions in ANN is the Rectified Linear Unit (ReLU) because it is both easy to implement and performs well on many predictive tasks. ReLU is a simple non-linear transformation where the output is defined as the maximum of the input element and 0. It is a popular choice due to its simplicity and good performance.

$$\text{Relu}(x) = max(0, x) \tag{1.5}$$

Softmax function

The softmax function is commonly used in the output layer of machine learning models [24]. It takes a vector $\mathbf{x}$ of n real numbers and transforms it into another vector where all components are in the interval (0, 1) and their sum equals one. This is done by normalizing the input vector divided by the sum of the exponential of the input variables. The resulting components can be thought about as a probability distribution, because their sum equals one. The function is defined with the following equation:

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{1.6}$$

where x_i are the components of vector $\mathbf{x}$ from x_1 to x_n and j going through all elements.

1.1.4 Forward Propagation

Forward propagation is the process of calculating the output of a neural network given an input[6] [24]. During forward propagation, the input values are multiplied by the weights of the first layer, then a non-linear activation function is applied, and this process is repeated for each layer until the final layer produces the output.

Let us look at this process step-by-step:

The input example is $\mathbf{x} \in \mathbb{R}^d$, then the intermediate variable is

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x},$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{h \times d}$ is the weight parameter from the hidden layer. The activation function ϕ is then applied, resulting in the hidden activation vector of length h,

$$\mathbf{h} = \phi(\mathbf{z}).$$

The output $\mathbf{h}$ of the hidden layer is also an intermediate variable.

Given the example shown in Figure 1.1, assuming that the parameters of the output layer correspond to a weight matrix $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times h}$, the output layer variable $\mathbf{o}$ with a vector of length q can be obtained as follows:

$$\mathbf{o} = \mathbf{W}^{(2)}\mathbf{h}.$$

The forward propagation step is performed for each input in the training dataset during the training. The output is compared to the expected output, and the difference between the two is used to adjust the weights and biases of the network in the backward propagation step.

1.1.5 Loss functions

The goal of a loss function [6] is to measure the discrepancy between the predicted output and the true output. The loss function provides feedback to the optimizer to update the model parameters in a way that reduces the error.

Two popular loss functions for binary classification tasks are Binary Cross Entropy and Cross Entropy Loss.

Binary Cross Entropy Loss [6] is used when there are only two classes, such as in a binary classification problem. The Binary Cross Entropy Loss function can be defined as:

$$L_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{1.7}$$

where $\hat{y}$ is the predicted probability of class 1 and y is the true label, which is either 0 or 1.

Cross Entropy Loss [6] is used when there are more than two classes. The Cross Entropy Loss function can be defined as:

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{1.8}$$

where $\hat{y}$ is the predicted probability distribution over C classes and y is the true label, which is a one-hot vector of length C.

1.1.6 Backpropagation

The main goal of backpropagation is to modify the parameters of a neural network in order to improve its predictions and make them better match the ground truth data. Backpropagation is a technique used to compute the gradient of neural network parameters, which enables this parameter modification.

In supervised learning, the term "ground truth" refers to the correct or true values of the target variable. It represents the actual or desired output that the neural network

aims to approximate or predict. During the training phase, the network is provided with input data along with corresponding ground truth values.

Backpropagation involves traversing the network in a reverse order, starting from the output layer and moving towards the input layer using the chain rule from calculus. This algorithm stores any intermediate variables (partial derivatives) needed while computing the gradient with respect to specific parameters. This gradient is then used to update the weights and biases in the opposite direction of the gradient, with the goal of reducing the loss function.[24]

1.1.7 Optimizers

Stochastic gradient descent (SGD) [6] is a popular variant of the gradient descent optimization algorithm used for training deep learning models. Gradient descent is a general optimization algorithm that aims to find the minimum of a given function by iteratively adjusting the parameters. However, in the context of deep learning, it is often infeasible to feed the entire training dataset into the model at once due to its size.

In stochastic gradient descent, the optimization process is performed on small subsets of the training data called mini-batches. Instead of computing the gradients of the loss function with respect to the model parameters on the entire dataset, SGD computes the gradients on a randomly selected mini-batch at each iteration, and then updating the parameters in the opposite direction of the gradient to minimize the loss. Despite its simplicity, SGD is still widely used due to its efficiency and effectiveness in many cases.

However, there are some limitations to SGD, such as its tendency to get stuck in saddle points or plateaus, and its sensitivity to the choice of learning rate. To address these issues, researchers have proposed various improvements to SGD, such as momentum, Adagrad [5], RMSprop [20], and Adam. In the next Section, I describe the Adam[10] optimization technique, that I used in my work.

1.1.8 Adam

Adam (adaptive moment estimation) is a first-order gradient-based optimization algorithm introduced by Kingma et al.[10]. This method combines the strengths of two commonly used optimization techniques: AdaGrad [5], which is effective for dealing with sparse gradients, and RMSProp [20], which performs well in non-stationary and online settings.

Adam [24] utilizes exponential weighted moving averages, also known as leaky averaging, to estimate both the momentum and the second moment of the gradient. In other words, it employs state variables to track and update these estimates.

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t, \tag{1.9}$$

$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2. \tag{1.10}$$

The weighting parameters β_1 and β_2 are non-negative and commonly set to values such as $\beta_1 = 0.9$ and $\beta_2 = 0.999$. It is worth noting that the variance estimate changes at a much slower rate compared to the momentum term. When initializing $\mathbf{v}_0$ and $\mathbf{s}0$ as zeros, there is an initial bias towards smaller values. To address this bias, we can utilize the fact that $\sum i = 0^t \beta^i = \frac{1-\beta^t}{1-\beta}$ for re-normalization purposes. Consequently, the normalized state variables are expressed as follows.

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \text{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}. \tag{1.11}$$

With the accurate estimates available, the update equations can be written. First, the gradient is rescaled in a manner similar to the approach used in RMSProp [20], resulting in the expression:

$$\mathbf{g}'_t = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon} \tag{1.12}$$

In contrast to RMSProp, the update operation relies on momentum instead of the gradient directly. Once all the components are ready, the updates can be easily calculated in a simple manner, following a straightforward update formula.

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}'_t. \tag{1.13}$$

1.1.9 Generalization

Generalization in machine learning refers to the ability of a model to make accurate predictions on new, unseen data, beyond the data it was trained on. It is the ultimate goal of machine learning, where the purpose is not only to fit the training data but also to discover underlying patterns that enable the model to make accurate predictions on new, unseen data. In deep learning, generalization is a crucial challenge due to the complexity of the models and the high dimensionality of the data. While the theory of deep learning is still evolving and far from a comprehensive account of both optimization and generalization, practitioners have developed a range of techniques and heuristics that can help to produce models that generalize well in practice. These methods are presented in Subsection 1.1.10.

1.1.10 Overfitting

Overfitting occurs when the model learns to fit the training data too closely and fails to generalize well to new, unseen data. This can happen when the model is too complex or when it is trained for too many epochs, resulting in a model that is too specialized to the training data and unable to capture the underlying patterns in the data. As a result, the model may perform very well on the training data but poorly on the test data, leading to a large generalization gap.

There are several methods that can be used to address overfitting. These methods include weight decay, dropout and early stopping.

Weight decay Weight decay adds a penalty term to the loss function that the model is trying to minimize. This penalty term is proportional to the square of the weights in the model, and it encourages the model to use smaller weights. The mathematical equation for weight decay is:

$$\mathcal{L}(w, b) = \mathcal{L} + \frac{\lambda}{2}||w||^2 \tag{1.14}$$

In this equation, $\mathcal{L}$ is the original loss function, λ is the weight decay coefficient and $||w||^2$ is the norm of the weight vector.

Dropout Dropout works by randomly dropping out (setting to zero) a proportion of the neural activations in a layer during training. The dropped out neurons are selected at random, with a given probability, typically around 0.5. With dropout probability p, each intermediate activation h is replaced by a random variable h' as follows:

$$h' = \begin{cases} 0 & \text{with probability } p \\ \frac{h}{1-p} & \text{otherwise} \end{cases} \tag{1.15}$$

Early stopping Early stopping constrains the number of epochs of training. During the training, the performance on validation data is monitoring, usually by checking it once after each nth epoch. The training stops, when the validation error has not decreased by more than some small amount ϵ for some number of epochs.

K-fold cross-validation If we have limited training data, it may not be possible to set aside enough data for a proper validation set. One commonly used technique to address this issue is K-fold cross-validation [24]. This method involves dividing the original training data into K subsets that do not overlap. The model is then trained and validated K times, with each training round using K-1 subsets for training and a different subset for validation.

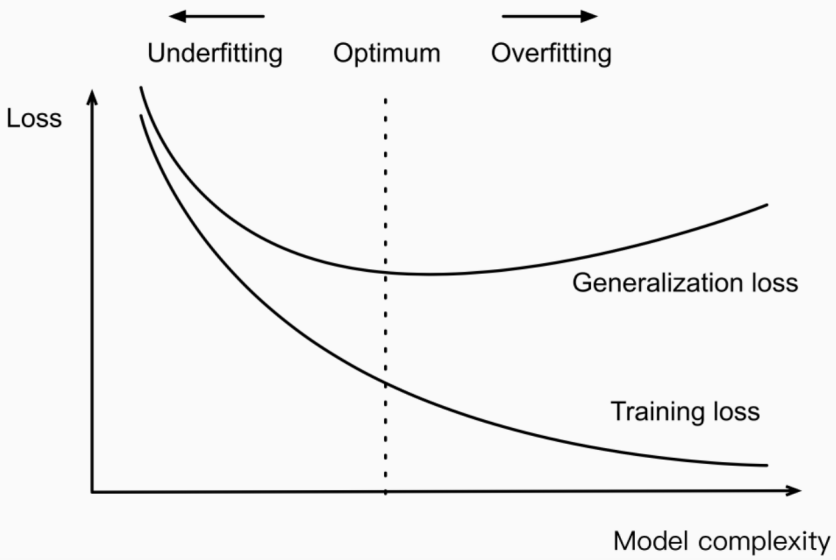


Figure 1.2: Influence of model complexity on underfitting and overfitting [24].

1.2 Word2vec

In this Section, I will discuss pre-trained word representations, such as Word2vec, according to studies [15] and [14]. Word2vec representations are commonly used in NLP pipelines to improve their performance. These pre-trained representations provide distributional information about words that can improve the generalization of models learned on limited amount of data. The typical method for obtaining word representations is through log-bilinear models trained using either the skip-gram [14] or continuous bag-of-words (CBOW) [14] architectures, which are commonly implemented in order to obtain word vectors. These models are trained on large unlabeled corpora of text data, and capture statistical information from vast sources of data.

To improve the quality of the resulting word vectors, several modifications have been made to the standard word2vec training pipeline. These include position-dependent features and phrase representations. These modifications have been evaluated on various benchmarks such as syntactic, semantic, and phrase-based analogies, rare words dataset, and as features in a question-answering pipeline [15].

The *CBOW* [15] model learns to predict a target word based on its surrounding context. The context is defined as a symmetric window that includes all the adjacent words. In other words, given a sequence of T words $w_1, w_2, ..., w_T$, the *CBOW* model aims to maximize the log-likelihood of the probability of the words, given their surrounding context, i.e.:

$$\sum_{t=1}^{T} \log p = (w_t|C_t) \tag{1.16}$$

where C_t is the context of the t-th word.

The Continuous Skip-gram Model [14] is similar to $CBOW$ model. However, instead of predicting the present word based on its surrounding context, the skip-gram model takes each current word as input and feeds it into a log-linear classifier with a continuous projection layer. The goal is to predict words that appear within a certain range both before and after the current word. Increasing the range improves the quality of the resulting word vectors, but it also increases the computational complexity.

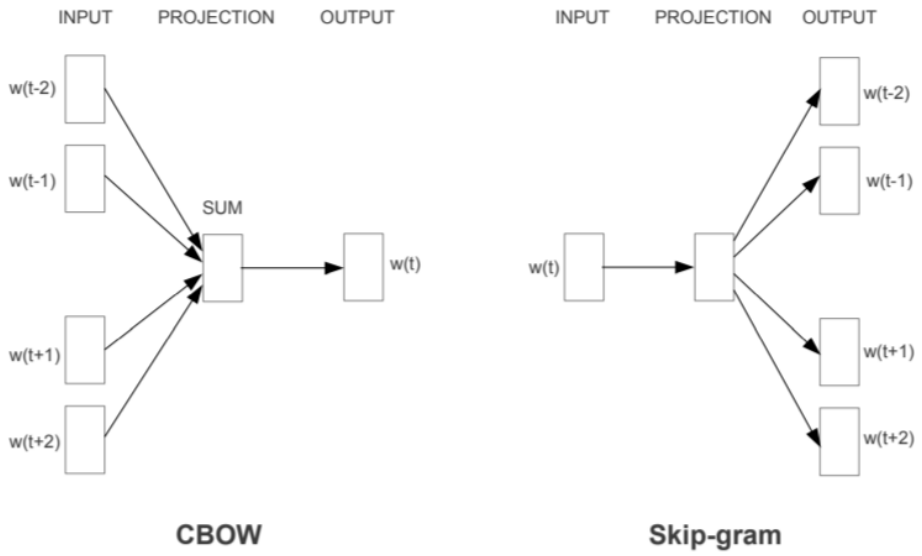


Figure 1.3: CBOW and Skip-gram models architecture [14].

1.3 Transformer

Recurrent neural networks, including long short-term memory [7] and gated recurrent neural networks [3], are commonly used for sequence modeling and transduction tasks such as language modeling and machine translation. Researchers have continued to improve these models and explore new architectures. However, these models are sequential, which limits parallelization within training examples and becomes critical at longer sequence lengths.

To address this issue, the Transformer [22] model was proposed. It relies entirely on a self-attention mechanism to draw global dependencies between input and output sequences, instead of recurrence. This allows for significantly more parallelization and achieves state-of-the-art results in translation quality with a shorter training time.

Architecture of the transformer

Nowadays, competitive neural models [22] for sequence translation have an encoder-decoder structure. The encoder maps an input sequence of symbols to continuous representations, while the decoder generates an output sequence one symbol at a time, using the previously generated symbols as additional input. The Transformer follows this structure, using self-attention and fully connected layers for both the encoder and decoder. This is shown in Figure 1.4, where the left half represents the encoder and the right half represents the decoder. The specific variant of the Transformer model I employed in my experiment is the encoder-only transformer [9].

The encoder in the Transformer model consists of six identical layers. Each layer has two parts: a multi-head self-attention mechanism and a simple, MLP defined in Section 1.1.2. Residual connections are employed around each sub-layer, followed by layer normalization. The output of each sub-layer is added to the input to facilitate residual connections. All sub-layers in the model and embedding layers produce outputs of a fixed dimension to enable these connections.

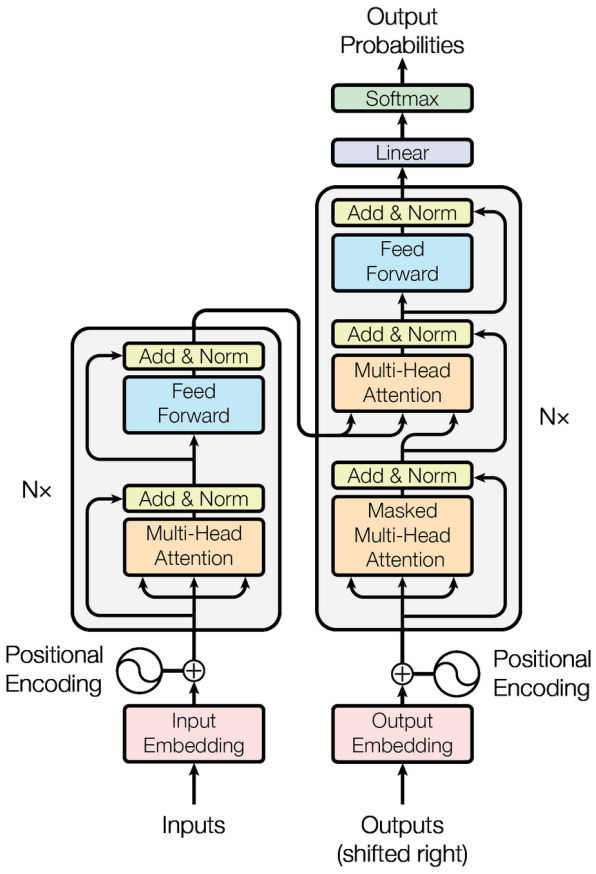


Figure 1.4: The architecture of the Transformer - model from [22]

1.4 BERT

The paper, titled "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [4], introduces a language representation model called BERT (Bidirectional Encoder Representations from Transformers). Unlike previous language models, BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. This approach enables BERT to be fine-tuned for a wide range of tasks, such as question answering and language inference, with just one additional output layer, without substantial task-specific architecture modifications. BERT is exclusively an encoder in the Transformer model because its main objective is to generate a language model.

The model is first pre-trained on unlabeled data over different pre-training tasks. One of the two tasks was masked language modeling, which involved randomly masking a certain percentage of the input tokens and then predicting the masked tokens. The second task was Next Sentence Prediction (NSP), which involve predicting whether a given sentence was the actual next sentence in a pair of sentences or a random sentence from the corpus, and helped the BERT model understand the relationship between two sentences.

After pretraining, the model can be fine-tuned using end-task labeled data. This process is simple and efficient due to the self-attention mechanism in the Transformer that enables the model to handle various downstream tasks. The input and output layers are plugged into BERT for each task, and all parameters are fine-tuned end-to-end. Fine-tuning is less expensive compared to pre-training.

The architecture of the model is a bidirectional Transformer encoder with multiple layers, which is based on the original implementation explained in the paper [22] BERT comes in two sizes: $BERT_{base}$ and $BERT_{large}$. $BERT_{base}$ has 12 Transformer blocks, 12 self-attention heads, and 110 million parameters, while $BERT_{large}$ has 24 Transformer blocks, 16 self-attention heads, and 340 million parameters.

Input/Output Representations

The input representation of BERT is able to handle both single sentences and pairs of sentences in one token sequence. Each sequence starts with a special classification token ([CLS]) and is separated by a special token ([SEP]). A learned embedding is added to every token indicating whether it belongs to sentence A or sentence B. The final embedding is constructed by summing the corresponding token, segment, and position embeddings as seen on Figure 1.5. The final embeddings go through deep bidirectional layers to produce the resulting output. In BERT, the output comprises hidden state vectors, representing each token in the input sequence with a fixed hidden size.

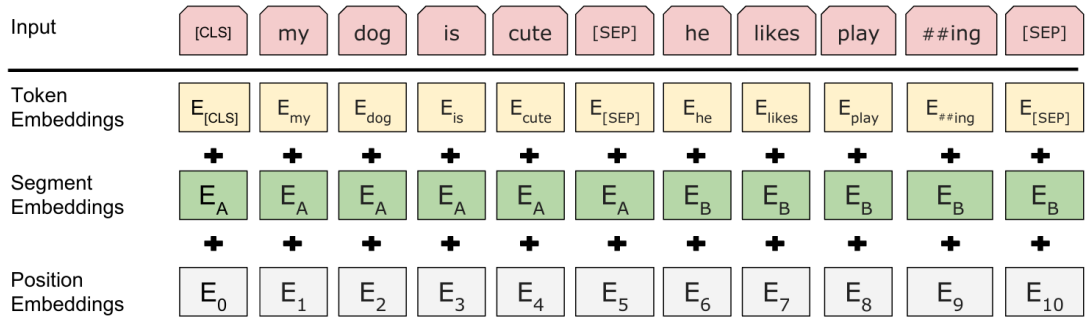


Figure 1.5: The representation of the BERT input. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.[4]

1.5 RoBERTa

RoBERTa (Robustly Optimized BERT Approach) is a transformer-based language model that was developed by Liu et al. [12]. This model is based on BERT released by Jacob Devlin et al.[4]. In their study, Liu et al. conducted a replication of the BERT pre-training approach with a focus on the effects of hyperparameter tuning and training set size. They found that, the original BERT model was undertrained, therefore they proposed an improved recipe for training BERT models that they called RoBERTa. Their modifications to the original approach includes training the model longer with larger batches and over more data, removing the next sentence prediction objective, training on longer sequences and dynamically changing the masking pattern applied to the training data.

1.6 Slovak Word Embeddings

In my research, I employed two distinct varieties of word embeddings specifically designed to represent Slovak words. These encompass the Slovak Word2Vec embeddings and the SlovakBERT embeddings.

1.6.1 Slovak word2vec

The word2vec representation I described in Section 1.2. The employed Word2vec model in this study had been trained on The Slovak National Corpus [25], an extensive electronic database comprising Slovak language texts from 1955 onward. This corpus covers a wide range of language styles, genres, areas, regions, and more. It contains approximately 110 million words, providing a substantial and representative dataset for training the Word2vec model.

1.6.2 SlovakBERT

Slovak masked language model called SlovakBERT [18] has RoBERTa architecture with 12 layers, 12 self-attention heads and 768 hidden size. It was trained using Web-crawled corpus. The available corpora they used were: Wikipedia (326MB of text), Open Subtitles (415MB) and OSCAR 2019 corpus (4.6GB). They crawled .sk top-level domain webpages, they extracted the title and the main content of each page as clean text without HTML tags (17.4GB).

Chapter 2

Methods

The aim of this work is to evaluate the semantic similarity between pairs of words and find the most suitable word representations capable of capturing semantic information. To achieve this, we opted not to utilize one-hot encodings because they solely indicate the word's position in the vocabulary. Instead, we employed distributed word representations.

Although one-hot encodings were previously widely utilized, they have now become outdated due to their limitations. These encodings represent words as unary vectors, where all elements except one are set to zero. The non-zero element corresponds to the word's position in the vocabulary. Nonetheless, one-hot encodings are incapable of capturing semantic relationships among words or providing contextual information.

In this chapter, I will discuss the data used for the experiments and the methods employed to evaluate semantic similarity. For this purpose, I utilized two types of distributed word representations - Word2Vec and Slovak BERT - which are well-known for their ability to capture semantic information about words. One can find more details about Word2Vec and Slovak BERT in Section 1.2 and in Subsection 1.6.2.

In Sections 2.1 and 2.2, I provided a detailed description of the methods used to obtain word embeddings for Slovak words using both Slovak Word2Vec, presented in Section 1.6.1, and Slovak BERT, discussed in Subsection 1.6.2 models.

In Section 2.3, I describe the method for evaluating semantic similarity through word embedding distances in three different experiments: the Chain Association Task, the Fluency Association Task, and the Discrete Task, presented in Sections 2.3.2, 2.3.3 and 2.3.1.

Section 2.4 provides a detailed description of the data preparation process for binary classification and the implementation of a Multilayer Perceptron (MLP), discussed in Subsection 1.1.2 model. The goal of this experiment is to predict the similarity of word pairs, classifying them as either similar or dissimilar.

The last Section 2.5 describes the process of data preparation and the implementa-

tion of an MLP, introduced in Subsection 1.1.2 model for classifying words into their respective categories.

2.1 Obtaining word2vec word embeddings

To acquire word embeddings for Slovak words, the pre-trained Word2vec model from the `SparkNLP` library was utilized [11]. `SparkNLP` is an advanced open-source Natural Language Processing (NLP) library that leverages Apache Spark, providing efficient and accurate NLP annotations for machine learning pipelines in distributed environments. It offers a comprehensive range of pre-trained pipelines and models supporting various NLP tasks such as tokenization, part-of-speech tagging, named entity recognition, text classification, sentiment analysis, machine translation, question answering, and more.

For tokenization and word embedding generation, `SparkNLP`'s classes were employed, including the `DocumentAssembler`, `Tokenizer`, and `WordEmbeddingsModel`. The `DocumentAssembler` class facilitated the preprocessing of the input text, while the `Tokenizer` class performed the tokenization process.

Afterwards, the `WordEmbeddingsModel` class was utilized to generate word embeddings using the pre-trained Word2vec model. Each token was mapped to a numerical representation capturing its semantic meaning.

The resulting word embeddings were stored in a dictionary format, where each key-value pair represented a word and its corresponding embedding vector.

2.2 Obtaining SlovakBERT word embeddings

SlovakBERT, as described in Section 1.6.2, is a context-sensitive model known for its ability to produce distinct word embeddings based on contextual usage. This feature allows for effective capturing of diverse word meanings and usages. The retrieval of embeddings was accomplished by utilizing the `transformers` library [23] in Python [21], specifically employing the `RobertaTokenizer` and `RobertaModel` introduced in Section 1.5.

The process was initiated by loading the pre-trained tokenizer and model specifically designed for the Slovak language using the `from_pretrained` method. The tokenizer played a pivotal role in transforming each word into a sequence of tokens, while the model was responsible for generating hidden states for each token.

For the purpose of retrieving the embeddings, a function called `get_word_vector` was implemented. Within this function, a single word was accepted as input, and a 768-dimensional embedding vector representing the word was returned. The initial step

involved encoding the input word using the tokenizer. Following that, the resulting tensor was passed through the pre-trained model to obtain the hidden states of the model for each token.

In the subsequent stage, we extracted the hidden states from the second-to-last layer of the model. Leveraging the methodology described in [1], we specifically utilized the advantage of the last token. The resulting value was then returned as the final embedding vector, encapsulating the contextualized embedding of the input word using SlovakBERT.

2.3 Data sources and task descriptions

The Chain Association Task, Fluency Association Task, and Discrete Task involve participants being presented with stimuli and then generating responses. To examine the semantic relationship between the stimuli and responses, the distances between the word pairs' embeddings are calculated using word2vec and SlovakBERT embeddings. This analysis aims to explore how the semantic relationship can be determined through these distance calculations.

2.3.1 Discrete task

The Discrete Task [13] involves participants generating single-word responses to a given stimulus, which can be categorized as either associative or dissociative. The response data is stored in CSV files.

To process the data, we utilize the `DiscreteTask.py` script, utilizing the `DataFrame` class from the `pandas` library [16]. This allows us to extract word pairs and their corresponding labels, indicating their association or dissociation.

Next, we obtain word2vec and SlovakBERT embeddings for each word using the `get_word_vectors` function from the `utilities.py` script, as described in Section 2.1 and 2.2. Afterward, we calculate the Euclidean distances and cosine similarities between the embeddings of associated and dissociated word pairs.

To organize and present the results, separate Excel files are created for word pairs, including their distances calculated using word2vec and SlovakBERT embeddings. These files contain the word pairs themselves, along with their Euclidean distances, cosine similarities, and the corresponding condition indicating their association or dissociation. These Excel files can be found in Appendix.

The average distances between associated and dissociated word pairs are calculated, providing a quantitative measure of the semantic similarity or dissimilarity in the responses. Moreover, to enhance the comprehensibility of the findings, visual representations in the form of bar charts 3.1 3.2 with error bars are generated using the

`matplotlib` library [8]. These bar charts illustrate the difference in means between associated and dissociated word pairs, and the error bars provide a representation of the standard deviation of these distances.

2.3.2 Chain association task

The Chain Association Task [13] involves the generation of a chain of words where each word is semantically related to the previous word. The task data is stored in Excel files, which are processed using the `pandas` library [16] to extract the word pairs. Subsequently, embeddings are obtained for each word, and the distances between these word pairs are calculated using both word2vec and SlovakBERT models. The methodology for obtaining Word2vec embeddings is described in Section 2.1, while the process for acquiring SlovakBERT embeddings is outlined in Section 2.2.

The analysis results are presented in separate files for word2vec and SlovakBERT. The output of our analysis is presented in a separate Excel file for each model.These Excel files can be found in Appendix. These files contain the word pairs along with their Euclidean distances and cosine similarities. These distances provide insights into the semantic relationships and associations between the words. Furthermore, the average distances between the associated word pairs are calculated for each model.

To present the results in a visually informative manner, distinct bar charts are generated for the Euclidean distances and cosine similarities utilizing the `matplotlib` library [8], as shown in Figure 3.3.

2.3.3 Fluency association task

In the Fluency Association Task, participants generate exemplars belonging to specific categories, such as animals, liquids, tools, and vegetables. In this study, word pairs were created using these generated words to assess their semantic similarity. Word pairs in which both words belonged to the same category were labeled as "associated," while word pairs consisting of words from different categories were labeled as "dissociated."

To begin, the words and their corresponding categories were extracted from a CSV file, using the `pandas` library [16]. Word2Vec and SlovakBERT embeddings were obtained for each word, using the `get_word_vectors` function from the `utilities.py` script, and the aforementioned word pairs were formed. The Excel files, which were created to store this information separately for word pairs using Word2Vec and SlovakBERT embeddings, can be found in Appendix. These files included the word pair, its label (associated or dissociated), as well as the calculated Euclidean and cosine similarity scores derived from the embeddings.

To compare the distances between word pairs from the same category and those from different categories, bar charts 3.4 were generated using the `matplotlib` library. These

charts visually represent the difference in means between associated and dissociated word pairs, with error bars indicating the standard deviation of these distances.

2.4 Binary classification experiments

Initially, we attempted to assess word similarity by calculating the distances between word embeddings. However, this method did not yield definitive results in determining whether a word pair is similar or dissimilar. Following that, we implemented a Multilayer Perceptron (MLP), introduced in Subsection 1.1.2, to predict the similarity of word pairs. To accomplish this, we implemented a binary classification process. To begin, I will discuss the necessary steps for data preparation in our experiment, followed by the implementation of the MLP.

In the first experiment, I used word2vec embeddings, showcased in Section 1.2, which had 300 dimensions. The model received embeddings of the word pair and a target value as input, resulting in an input size of 600 and an output size of one.

In the second experiment, I used Slovak BERT embeddings, introduced in Subsection 1.6.2, which had 768 dimensions, for training. The model also received embeddings of the word pairs and a target value as input, resulting in an input size of 1536. The output size remained one, and I tested the performance of the model with different hidden layer sizes.

The dataset, as shown in Table 2.1, consists of 967 unique words along with their corresponding categories, which fall into four categories: Animals, Vegetables, Tools, and Liquids. Word embeddings were obtained according to the methods described in Sections 2.1 and 2.2. The difference in word counts between SlovakBERT and Word2Vec is attributed to the presence of out-of-vocabulary words in Word2Vec. A CSV file was generated containing the words, embeddings, and categories. The code responsible for this process can be found in the file `GenerateTrainingData.py` located in the directories `word2vec-embeddings` and `slovakBERT-embeddings`.

In order to train the MLP, a dataset of word pairs was required. Consequently, a custom dataset class was created using the `PyTorch` [17] library, which generates distinct pairs of words and a corresponding target label from the aforementioned dataset. The data samples were concatenated into a single tensor, and the target label was assigned as 1 if the two samples belonged to the same category, and 0 otherwise.

To perform K-fold cross-validation as defined in Paragraph 1.1.10, a method was implemented within our customized dataset class. The relevant code can be found in the file `data_handler_BCT.py` located in the `TrainMLP` directory. This method accepts an argument specifying the number of distinct subsets to generate. By utilizing `PyTorch`'s subset class [17], the dataset was divided into separate subsets.

Table 2.1: This table presents the word counts of categories before and after dataset balancing. The "Original Word Count" column represents the word counts of categories before balancing, while the "Word Count (SlovakBERT)" and "Word Count (Word2Vec)" columns display the word counts after balancing using SlovakBERT and Word2Vec embeddings, respectively.

Category	Original Word Count	(SlovakBERT) Word Count	(Word2Vec) Word Count
Animals	333	113	70
Liquids	264	133	92
Tools	258	128	73
Vegetables	112	112	75

In this particular case, the dataset was split into five distinct subsets, with four subsets designated for training purposes and one subset reserved for validation. It is worth noting that the percentage of associated word pairs in both the training and validation datasets was approximately 27%. To address this, the dataset was balanced using the undersampling method, resulting in a 43% proportion of associated word pairs.

An MLP model, outlined in Subsection 1.1.2 was implemented for these experiments, utilizing `PyTorch`'s `nn.Module` [17]. The model's parameters include the following:

- number of input neurons
- number of neurons in the hidden layer,
- number of output neurons

In this implementation, a single hidden layer was employed, with different numbers of neurons. Specifically, the hidden layer sizes considered were 50, 150, and 300.

To train the model, I used the Binary Cross Entropy from the `nn` module in the `PyTorch` library under the name `BCEloss` function and the `Adam` optimizer, as described in Section 1.1.8. During training, both the training and validation accuracy and loss values were monitored, and early stopping based on a patience parameter was implemented. If the validation loss did not show any improvement over a specific number of epochs, specifically in this scenario, two conditions were used to decide when to stop the training: after 3 epochs or 5 epochs. At that point, the training was ended prematurely, and the model that demonstrated the highest performance was selected and returned.

The training and validation loss and accuracy values were recorded for each epoch during the evaluation of the model's training process. These values were plotted using the `Matplotlib` library [8], enabling the visualization of the model's learning curve.

In order to evaluate the model's performance, another set of experiments was conducted using data from a different distribution. This dataset, described in Subsection 2.3.1, comprises free associations and consists of a list of word pairs. In this dataset, the first word acts as a stimulus, while the second word represents the response, which can be either associated or dissociated.

2.5 Multi-class classification experiments

In the previous experiment, the model encountered limitations and stochastic information, indicating a potential lack of capacity. To address these issues, an alternative approach was employed, focusing on predicting word categories instead of word pair similarities.

For this purpose, a Multilayer Perceptron (MLP), discussed in Subsection 1.1.2 was implemented to classify words based on their categories. The dataset used for the binary classification experiments, presented in Section 2.4, consisting of 967 words, was utilized, eliminating the need for word pair creation.

Word2vec embeddings, introduced in Section 1.2, and Slovak BERT embeddings, presented in Subsection 1.6.2, were used in these experiments. The Word2vec embeddings have 300 dimensions, while the Slovak BERT embeddings have 768 dimensions. The model received word embeddings and a one-hot encoded target value as input. The output size of the model was set to four, corresponding to the number of categories.

A custom dataset class was defined, inheriting from PyTorch's [17] Dataset object, to load and process the data. The code can be found in the file `data_handler_MCT.py` located in the `TrainMLP` directory. This class converts the embeddings to tensors and the categorical labels to one-hot encoded vectors.

Similar to the previous experiment, presented in Section 2.4, the k-fold cross-validation technique, introduced in Paragraph 1.1.10, was used to partition the data into subsets for training. The dataset was divided into ten subsets, with each subset representing a distinct partition of the data. Within each iteration of the cross-validation process, eight subsets were used for training, one subset for validation, and one subset for testing.

Initially, the dataset, as presented in Table 2.1, was imbalanced, leading to difficulties in accurately categorizing the minority class, specifically the "vegetables" category. To address this issue, the dataset was balanced using undersampling techniques. Before balancing, the number of words in the "animals" category was three times larger

than in the "vegetables" category.

The MLP model, introduced in Subsection 1.1.2, utilized the `nn.Module` from PyTorch [17] package. The model's free parameters include the number of input neurons, the number of neurons in the hidden layer, and the number of output neurons. A single hidden layer with varying numbers of neurons, specifically 50, 100, and 300 neurons were used.

Similar to the previous experiments, the Adam optimizer, discussed in Subsection 1.1.8, was chosen for its strong performance in comparison to the SGD optimizer, discussed in Subsection 1.1.7. The Cross Entropy loss function from the `nn` module in the `PyTorch` library, named
`CrossEntropyLoss`, was chosen as the loss function. During training, both the training and validation accuracy and loss values were monitored, and early stopping based on a patience parameter was implemented. If the validation loss did not improve for five epochs, training was terminated early, and the model with the best performance was returned.

To evaluate the model's performance, the training and test loss and accuracy values were recorded for each epoch and plotted using the `Matplotlib` library [8]. This visualization allowed for the observation of the model's learning curve. Additionally, the `sklearn` library [2] was used to generate a confusion matrix, which provides insights into the model's performance by displaying the number of true positive, true negative, false positive, and false negative predictions made by the model.

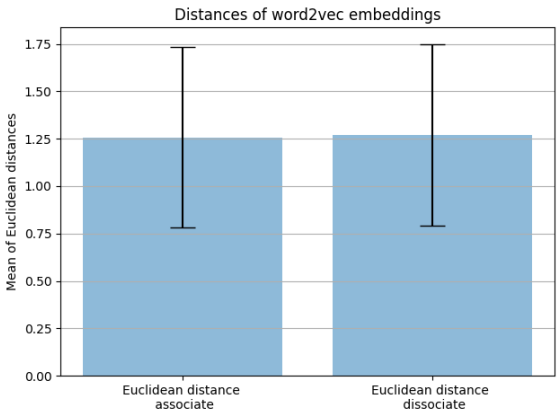
Chapter 3

Results

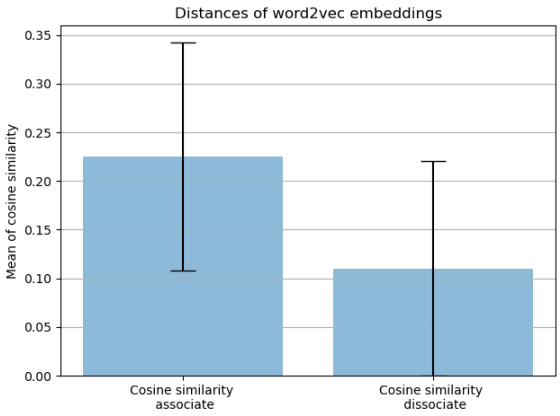
3.1 Comparing distances between word embeddings

To evaluate the semantic similarity between pairs of words, I used both Euclidean distance and cosine similarity metrics between their corresponding word embeddings.

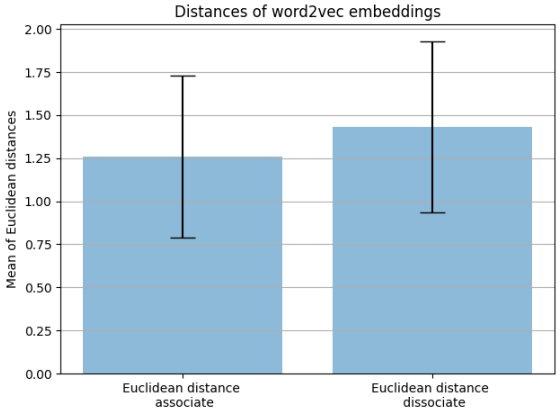
In the first experiment, the distances between words with associative and dissociative meanings were compared, as described in Subsection 2.3.1. The average Euclidean distance was calculated separately for each category, and then the average distance of word vectors with associative and dissociative meanings was compared. Figures 3.1 and 3.2 display the Euclidean distances and cosine similarity between word pairs that are associated and dissociated. In Figures 3.1b and 3.1d, it can be observed that the mean cosine similarities between dissociated word pairs are significantly closer to zero, indicating a larger angular separation between these pairs. However, it is worth noting that there is some degree of overlap in the similarity values between associated and dissociated word pairs.



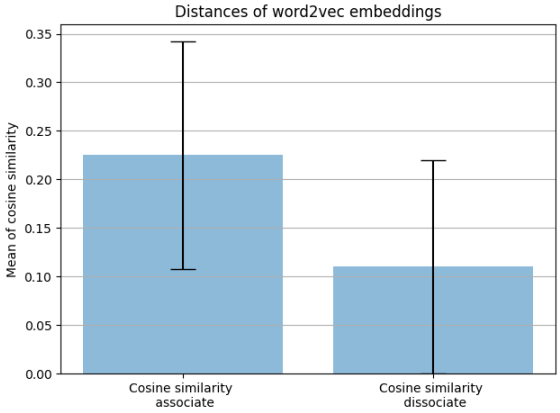
(a) The mean Euclidean distances of word2vec embeddings



(b) The mean cosine similarities of word2vec embeddings

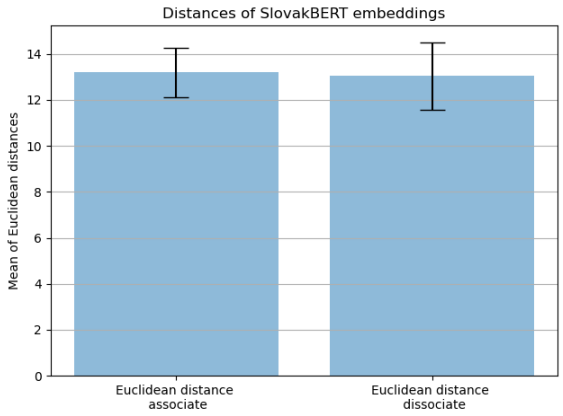


(c) The mean Euclidean distances of word2vec embeddings

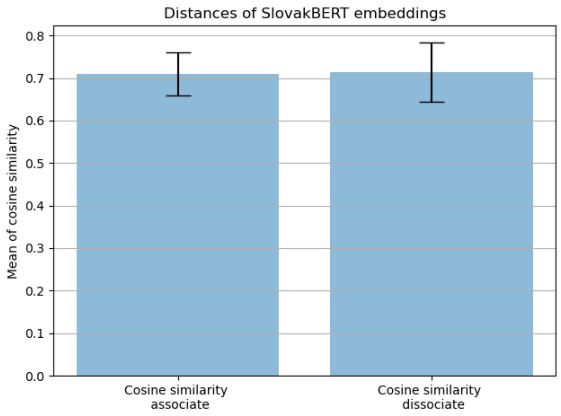


(d) The mean cosine similarities of word2vec embeddings

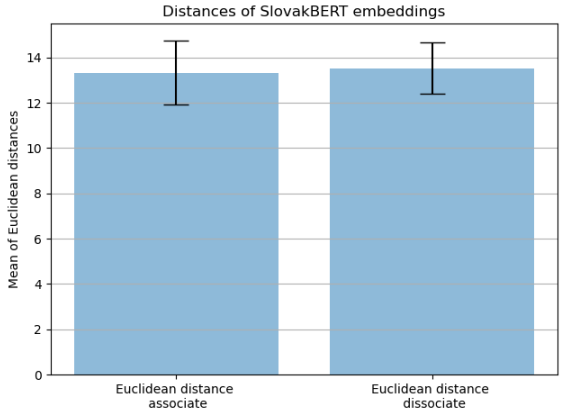
Figure 3.1: This figure depicts the mean Euclidean distances and cosine similarities between associated and dissociated word pairs obtained from the discrete association task. The height of each bar represents the average value, and the black error bars indicate the standard deviations. The methodology for this analysis is discussed in Subsection 2.3.1



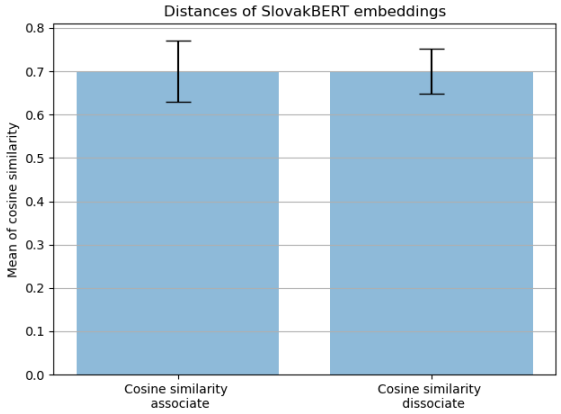
(a) The mean Euclidean distances of Slovak BERT embeddings.



(b) The mean cosine similarities of Slovak BERT embeddings.



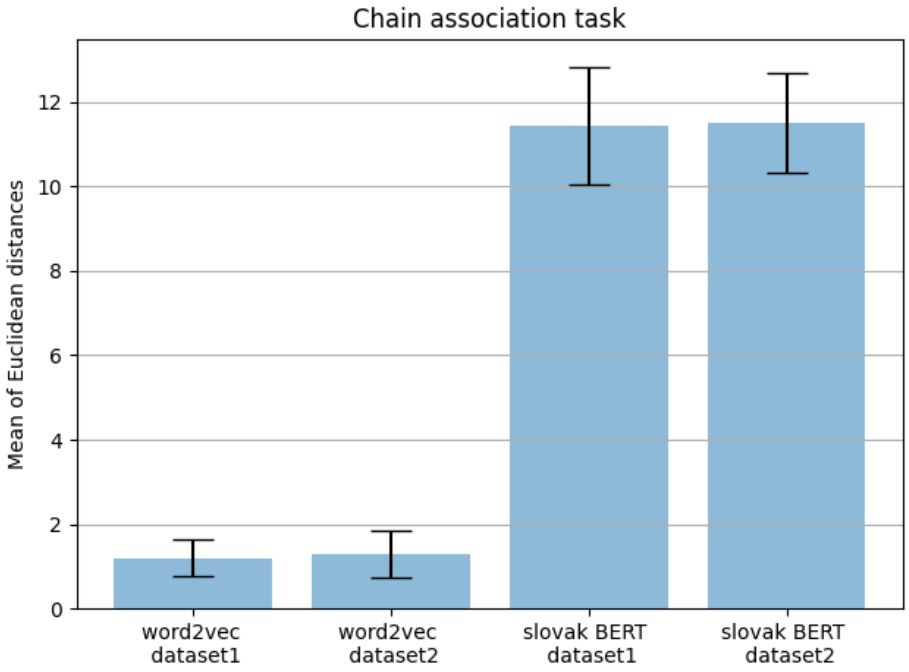
(c) The mean Euclidean distances of Slovak BERT embeddings.



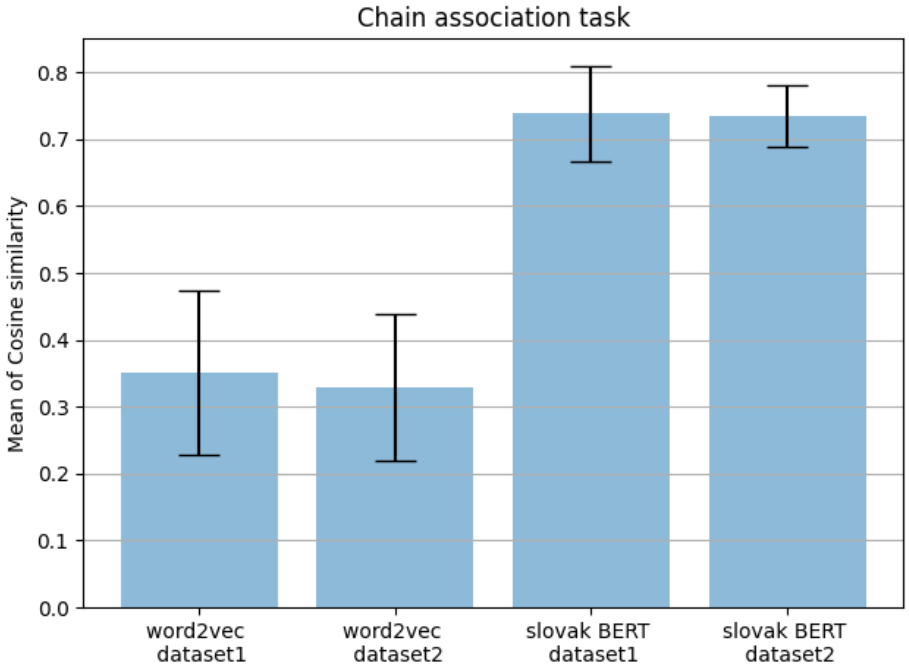
(d) The mean cosine similarities of Slovak BERT embeddings.

Figure 3.2: This figure illustrates average Euclidean distances and cosine similarities between associated and disassociated word pairs obtained from the discrete task, the averages are the height of each bar in my chart, the standard deviations are represented by the black error bars. This method is discussed in Subsection 2.3.1.

For the chain association task data, presented in Subsection 2.3.2, I compared every pair of words in an associative word chain and calculated the average distance between them, as well as the differences between the minimum and average distance, and the maximum and average distance. Figure 3.3 displays the average Euclidean distances and cosine similarities between associated word pairs.



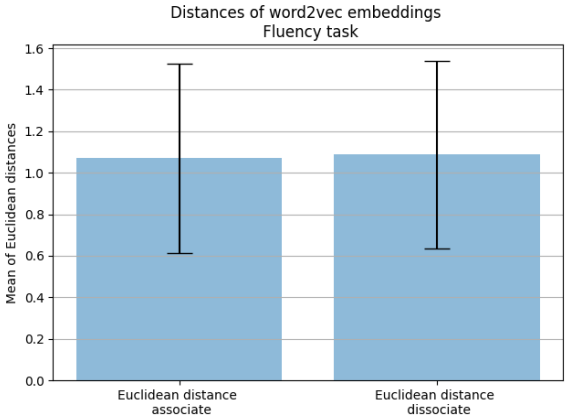
(a) The mean Euclidean distance between word2vec and Slovak BERT embeddings is computed for two different datasets, which differ in terms of the participants involved.



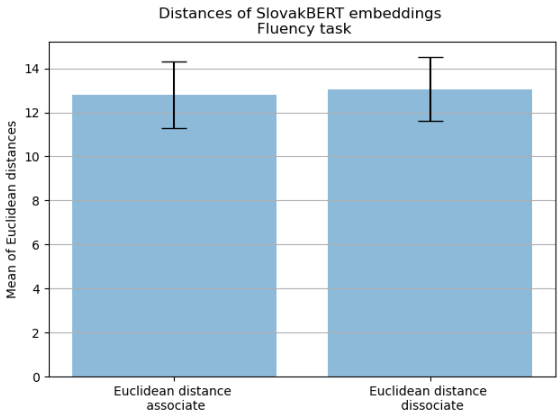
(b) The mean cosine similarity between word2vec and Slovak BERT embeddings is computed for two different datasets, which differ in terms of the participants involved.

Figure 3.3: The Figure displays the mean Euclidean distances and cosine similarities between associated word pairs from a word chain. The height of each bar in the chart corresponds to the mean value, while the black error bars indicate the standard deviation. The method of this experiment is presented in Subsection 2.3.2.

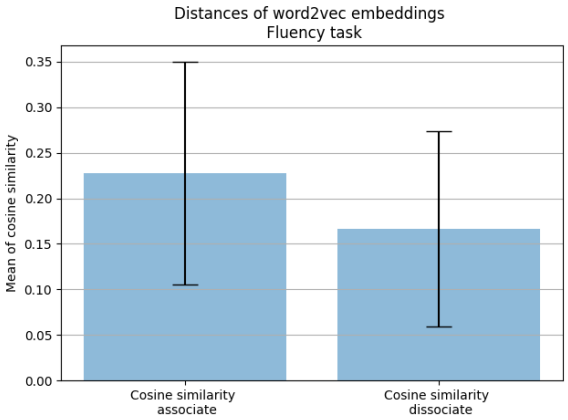
Finally, as described in Subsection 2.3.3, Euclidean distances and cosine similarities were calculated between word pairs using word2vec embeddings and Slovak BERT embeddings. The dataset consisted of words belonging to four categories. Figure 3.4 shows the mean Euclidean distances and cosine similarities between the word pairs.



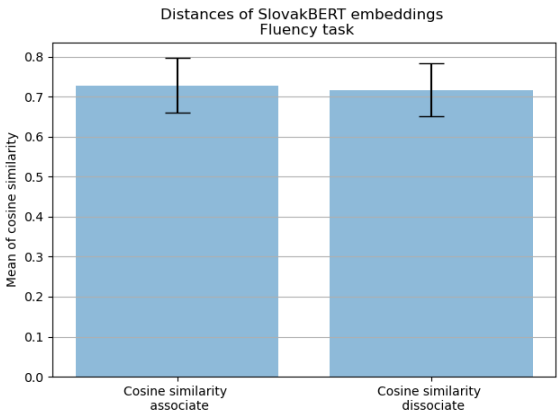
(a) Averages of Euclidean distances between pairs of word2vec embeddings.



(b) Averages of Euclidean distances between pairs of Slovak BERT embeddings.



(c) Averages of cosine similarities between pairs of word2vec embeddings.



(d) Averages of cosine similarities between pairs of Slovak BERT embeddings.

Figure 3.4: The Figure illustrates the average distances and similarities among the fluency task data, specifically word pairs categorized as either associated or dissociated. These word pairs were derived from a dataset consisting of words belonging to four distinct categories. Associated word pairs are formed from words within the same category, while dissociated word pairs are created using words from different categories. The y-axis represents the mean distance, and the black error bars represent the standard deviation. The method is showcased in Subsection 2.3.3.

Table 3.1: Evaluating model performance in the binary classification using Word2Vec embeddings and varying hyperparameters. The dataset for training and validation consisted of 967 unique words. Word pairs were generated from these words, classified as either association or dissociation, regardless of their category membership. 5-fold cross-validation was employed to assess the model's performance. The figure displays the accuracy and loss values obtained during the evaluation.

Number of epochs	Dim_{hid1}	Learning rate	Validation accuracy	Validation loss
7	50	0.01	99.3%	0.043
8	150	0.01	99.4%	0.034
7	300	0.01	99.5%	0.021
20	50	0.001	99.8%	0.018
21	150	0.001	99.8%	0.010
17	300	0.001	99.9%	0.005

3.2 Binary classification experiments

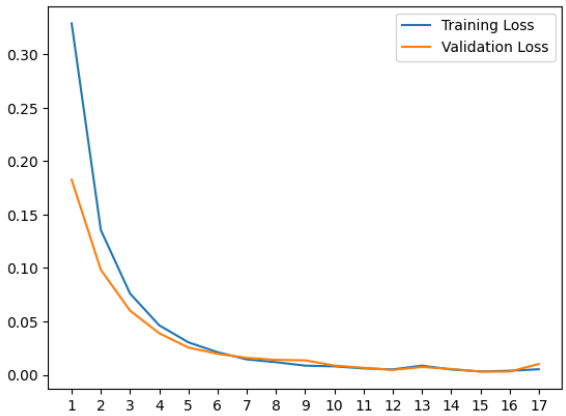
In this Section, I will present the results of the experiment, where a Multilayer Perceptron (MLP), discussed in Subsection 1.1.2, was implemented to classify the similarity of word embedding pairs. The preparation of the dataset and the implementation of MLP for training was described in Section 2.4. To train the MLP, word2vec embeddings were initially utilized, followed by Slovak BERT embeddings.

In the first experiment, I used word2vec embeddings for training. The model received two word embeddings and a target value as input. Since word2vec embeddings have 300 dimensions, the MLP received a concatenation of two embeddings, resulting in an input size of 600. The output size remained one, and different hidden layer sizes were tested.

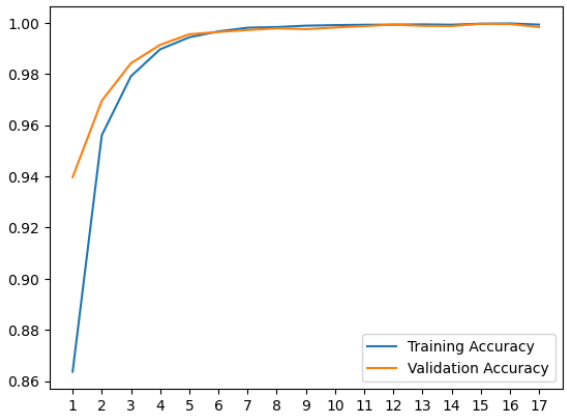
Various hyperparameters were experimented with to assess their impact on accuracy. As observed in Table 3.1 the results were quite consistent across different hyperparameters. When the learning rate was set to 0.01, early stopping occurred earlier. On the other hand, a learning rate of 0.001 led to higher accuracy and lower loss as the training progressed over more epochs. Increasing the number of neurons in the hidden layer resulted in a slight improvement in the model's accuracy, albeit the difference was marginal.

Figure 3.5 illustrates the improvement of the training accuracy and validation accuracy during the epochs. During this training, I used a hidden layer size of 300, a learning rate was set to 0.001, and the momentum was set to 0.9. As shown in the

figure, the most significant improvements in accuracy and loss occurred during the first five epochs. Subsequently, the training and validation loss and accuracy remained relatively constant. Early stopping was triggered on the 17th epoch since the validation loss did not improve for three consecutive epochs.



(a) Training loss and validation loss.



(b) Training and validation accuracy.

Figure 3.5: Training and validation in the binary classification using Word2Vec embeddings. The dataset consists of word pairs categorized as associative or dissociative, depending on whether they belong to the same category or not. 5-fold cross-validation was utilized to evaluate the model's performance. The experiment employed the following hyperparameters: a hidden layer size of 300, a learning rate of 0.001, and a momentum of 0.9.

Since the model demonstrated exceptional performance on the validation dataset as well, we proceeded to assess its performance using a dataset from a distinct task. This dataset consisted of free associations, described in Subsection 2.3.1, rather than category-based comparisons like those used in the training and validation datasets. The best accuracy the model has achieved on this test dataset is 51% and the loss is 9.022. However, upon examining the word pairs, it becomes evident that certain word pairs that are expected to be associated based on their category show dissociation in free associations, while some word pairs that are expected to be dissociated based on their category exhibit association in free associations.

In my second experiment, I used Slovak BERT embeddings for training. The MLP was fed with embeddings of a word pair and a target value. Each embedding was 768 dimensional, since the MLP gets a concatenation of two embeddings, the input size is 1536. The output size is one. The MLP had one hidden layer, and the experiment has been run with different hidden layer sizes. I conducted several experiments to evaluate the impact of different hyperparameters on the accuracy of my model. The results, as shown in the Table 3.2, were quite similar to the previous experiment that utilized word2vec embeddings.

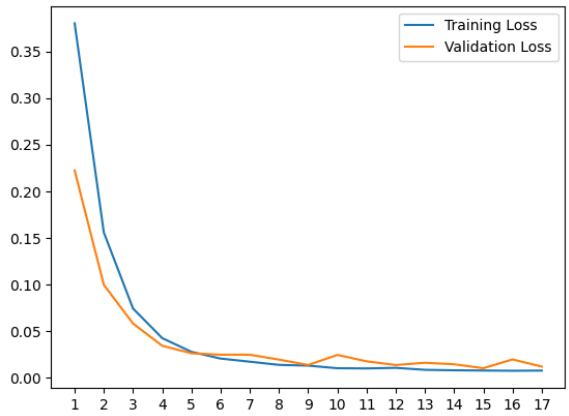
During these experiments, I tested various hyperparameters, including different learning rates. A learning rate of 0.01 led to early stopping occurring sooner, while a learning rate of 0.001 resulted in higher accuracy and lower loss as the training progressed over more epochs.

Furthermore, I explored the effect of increasing the number of neurons in the hidden layer. Interestingly, I observed that this change improved the accuracy of the model, albeit with only marginal differences.

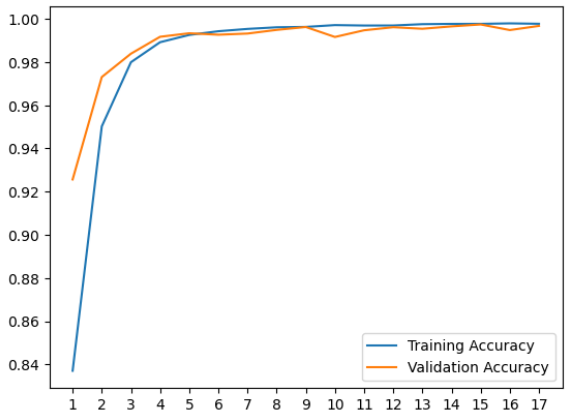
Table 3.2: Evaluation of model performance in the binary classification using varied hyperparameters and SlovakBERT embeddings. The training and validation dataset consisted of 967 distinct words. Word pairs were created from these words, categorized as association or dissociation, whether they belonged to the same category or not. 5-fold cross-validation was applied during the evaluation of the model's performance. The figure displays the accuracies and loss values obtained during the evaluation of the model.

Number of epochs	Dim_{hid1}	Learning rate	Validation accuracy	Validation loss
14	50	0.01	98.5%	0.058
11	150	0.01	96.9%	0.096
12	300	0.01	97.2%	0.063
16	50	0.001	99.1%	0.025
18	150	0.001	99.5%	0.021
17	300	0.001	99.2%	0.023

Figure 3.6 illustrates that both the training accuracy and validation accuracy improve similarly during the epochs.



(a) Training and validation loss.



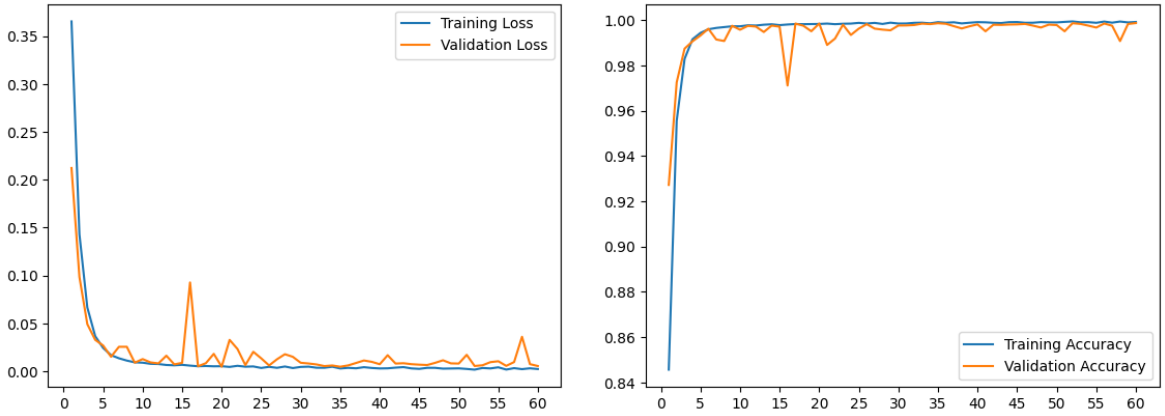
(b) Training and validation accuracy.

Figure 3.6: Training and validation results for a binary classification using Slovak BERT embeddings. The dataset for training and validation comprised 967 distinct words. Word pairs were generated from these words and categorized as association or dissociation, regardless of their category membership. The hyperparameters used in the experiment included a hidden layer size of 150, a learning rate of 0.001, and a momentum of 0.9. 5-fold cross-validation was applied. The results demonstrate that the model achieved high accuracy and low loss on both the training and validation sets.

During this training, as depicted in Figure 3.6, I used a hidden layer size of 150, a learning rate I set to 0.001, and a momentum set to 0.9. As shown in figure 3.6, the most significant improvements in accuracy and loss reduction occurred during the first five epochs. The training and validation loss and accuracy values remained relatively constant for the rest of the epochs. The early stopping occurred at the 17th epoch because the validation loss did not improve for 5 epochs. The accuracy on the validation set was 99.5% and the loss was 0.021.

Following the experiments using word2vec embeddings, I also tested the model on a dataset that consists of free associations, presented in Subsection 2.3.1, in contrast to the category-based comparisons present in the training and validation datasets. The best accuracy that the model achieved on this test dataset was 54% and the loss is 6.928. However, upon examining the word pairs, it becomes evident that certain word pairs that are expected to be associated based on their category show dissociation in free associations, while some word pairs that are expected to be dissociated based on their category exhibit association in free associations. For example, in the training data, the words "pivo" (beer) and "krém" (cream) are both categorized as liquids and therefore they are associated with each other. However, in the test data, this word pair is disassociated. In the Methods Section, I provided a description of the datasets, which can be found in Section 2.4.

In this experiment, early stopping was not employed to observe potential overfitting. I used a model with one hidden layer with 300 neurons, the learning rate I set to 0.001 and the momentum to 0.9. The training was running for 60 epoch, but I did not observe overfitting, the training loss, validation loss, and accuracy values remained relatively constant for the epochs, as one can see in the Figure 3.7.



(a) Training and validation loss.

(b) Training and validation accuracy.

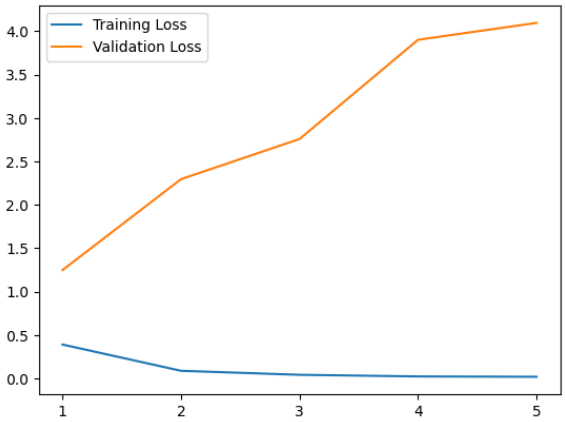
Figure 3.7: Training and validation results for a binary classification using Slovak BERT embeddings without early stopping. The hyperparameters used in the experiment included a hidden layer size of 300, a learning rate of 0.001, and a momentum of 0.9. The results demonstrate that the model achieved high accuracy and low loss on both the training and validation sets.

In the aforementioned experiments, I used a dataset with distinct word pairs, so the same word pair did not occur in the training and validation sets simultaneously. However, individual words that made up the word pairs could have occurred in multiple pairs and thus might have appeared in both the training and validation sets, but never with the same pair.

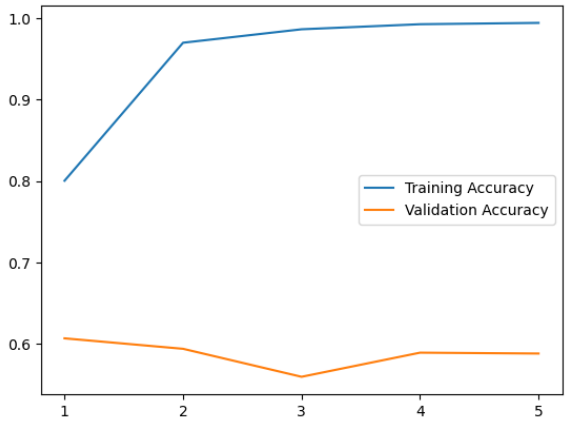
To evaluate how the model would perform on word pairs composed of completely new words, I modified the data preparation process as follows: I divided the dataset of 967 words into two sets, with 650 words for training and 315 words for validation. Additionally, I sampled unique pairs of words for both the training and validation sets. The size of the training set was 93921, while the validation set was 22556. The percentage of associated word pairs in both the training and validation datasets was approximately 27%. In order to have a more balanced dataset, I removed some dissociated word pairs to achieve a similar number of associated and dissociated word pairs, so the percentage of associated word pairs in the training dataset became 53%, and in the validation dataset, it was 54%.

I trained an MLP with two hidden layers, and I experimented with various hyperparameters. the results showed a consistent pattern where the validation accuracy and

loss deteriorated after the first epoch of training. Early stopping was applied when the validation loss did not improve for more than five epochs. Figures 3.8a and 3.8b illustrate the progress of training and validation. For this experiment, the MLP had 300 neurons in each hidden layer, the learning rate was set to 0.001, and the momentum was set to 0.9. During the first epoch, the accuracy was 60.6%, and the loss was 1.248. However, for the remaining epochs, there was no improvement.I found that the model struggled to perform effectively on word pairs where both words had not been seen in any other pair as well.



(a) Training and validation loss improvements during epochs with modified dataset using SlovakBERT embeddings.



(b) Training and validation loss improvements during epochs with modified dataset using SlovakBERT embeddings.

Figure 3.8: Training and validation results for a binary classification using Slovak BERT embeddings with a modified dataset. The hyperparameters used in the experiment included two hidden layers size of 300, a learning rate of 0.001, and a momentum of 0.9. 5-fold-cross validation was applied.

3.3 Multi-class classification experiments

In this experiment, I employed the identical dataset as my previous experiments. The dataset comprises 967 unique words, accompanied by their embeddings and categories. These words are categorized into four distinct groups: Animals, Vegetables, Tools, and Liquids. Unlike my previous experiments that involved creating word pairs for binary classification, I utilized individual words and their corresponding categories to perform multi-class classification in this study.

To address the initial imbalance in the individual categories, where the "vegetables" category was underrepresented, the dataset was balanced using the undersampling technique. Table 2.1 presents the word counts of categories before and after dataset balancing.

In this experiment, I utilized the Adam optimizer, which is described in detail in Subsection 1.1.8. I also attempted to use the SGD optimizer, discussed in Subsection 1.1.7, but the training required significantly more epochs on average and resulted in notably lower accuracies. For my loss function, I opted for the Cross-Entropy loss, which calculates the cross-entropy loss between the output and target.

Multi-class classification using word2vec For this experiment, word2vec embeddings of words were used for training. Table 3.3 displays the impact of hyperparameters on the test outcomes. The "Epoch" column indicates the epoch number at which the early stopping criterion was met. It can be observed that a learning rate of 0.001 resulted in longer training but achieved higher accuracy. The model with a learning rate of 0.001 and a hidden layer size of 300 achieved the highest accuracy.

Table 3.3: Accuracies in multi-class classification using varied hyperparameters and Word2Vec embeddings as input. The dataset comprised 967 distinct words from four categories. 10-fold cross-validation was applied to evaluate the model's performance. The table presents the best accuracies and loss values achieved during the test of the model, specifically obtained through 10-fold cross-validation.

Epoch	Dim_{hid}	Learning rate	Test accuracy	Test loss
10	50	0.01	74.2%	0.688
9	100	0.01	74.2%	0.623
7	300	0.01	67.7%	0.817
44	50	0.001	90.3%	0.409
35	100	0.001	93.5%	0.241
21	300	0.001	96.8%	0.210

Figure 3.9 displays the confusion matrix, where the rows indicate the actual classes and the columns represent the predicted classes. The diagonal cells of the table show the percentage of instances that the model correctly classified. For example, the cell in the first row and first column indicates that 100% of the actual animal instances were accurately classified as animals by the model.

Figures 3.10a and 3.10b show the training and validation loss, as well as the training and validation accuracy, respectively. The MLP model used in this experiment had 300 neurons in each hidden layer, a learning rate of 0.001, and employed early stopping and weight decay with a value of 0.001. The model achieved an accuracy of 96.8% and a loss of 0.210 on the overall test dataset.

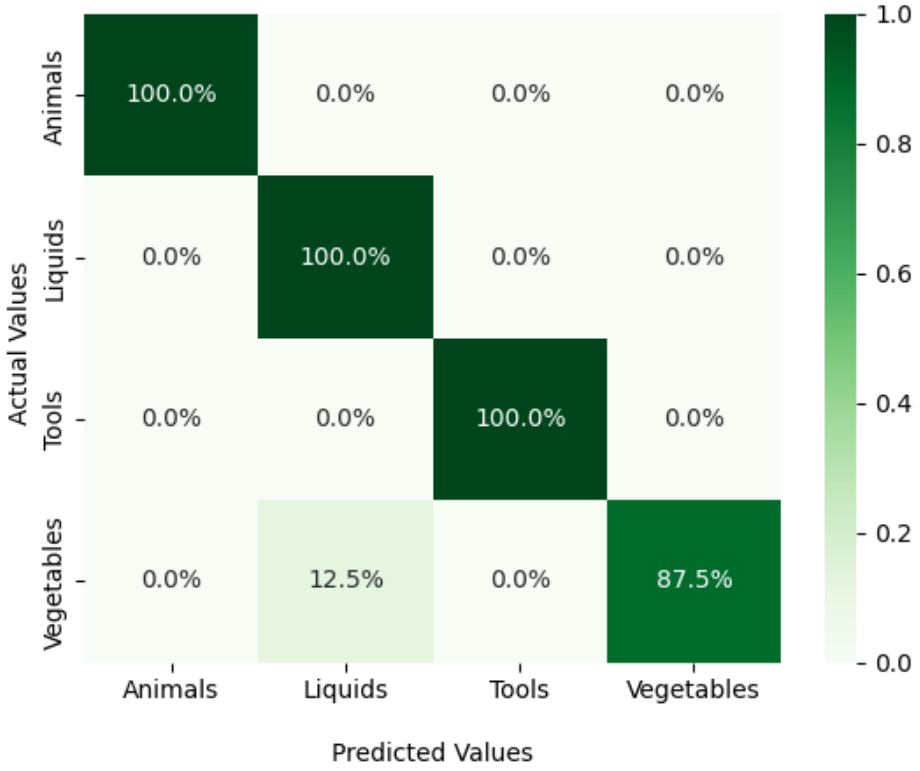
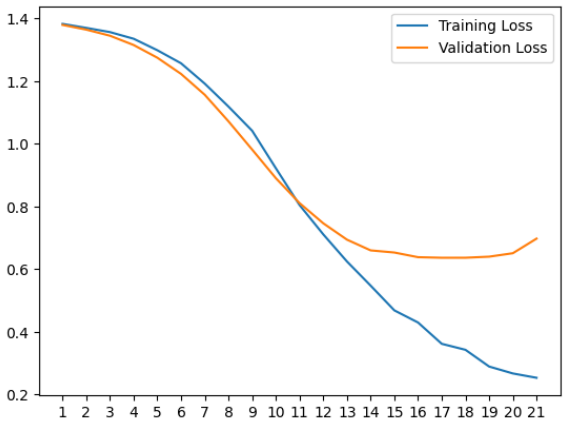
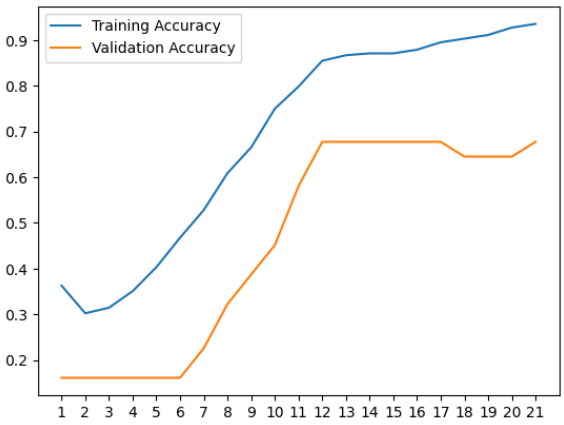


Figure 3.9: Confusion matrix of the multi-class classification using word2vec embeddings



(a) Training and validation loss of multi-class classification using Word2vec embeddings



(b) Training and validation accuracy of multi-class classification using Word2vec embeddings as input

Figure 3.10: Learning curves illustrate the changes in accuracy and loss throughout the process of multi-class classification, where Word2Vec embeddings are utilized as the input. The experiment utilized hyperparameters including a hidden layer size of 300, a learning rate of 0.001, and a momentum of 0.9. 10-fold cross-validation was employed to evaluate the model's performance.

Multi-class classification using Slovak BERT Table 3.4 presents the influence of hyperparameters on the test outcomes when using Slovak BERT embeddings. The model with a learning rate of 0.0001 and a hidden layer size of 300 achieved optimal results.

Table 3.4: Evaluation of multi-class classification accuracies using diverse hyperparameters and SlovakBERT embeddings as input. The dataset comprised 967 distinct words categorized into four categories. 10-fold cross-validation was applied to assess the model's performance. The table showcases the best accuracies and loss values obtained during the model's testing phase, obtained through 10-fold cross-validation.

Epoch	Dim_{hid}	Learning rate	Test accuracy	Test loss
12	50	0.01	68.8%	0.957
9	150	0.01	66.7%	0.942
11	300	0.01	72.9%	0.805
23	50	0.001	72.2%	0.821
16	150	0.001	74.1%	0.845
11	300	0.001	70.4%	0.859
103	50	0.0001	72.9%	0.845
50	150	0.0001	70.8%	0.870
38	300	0.0001	75%	0.844

Figure 3.11 shows the confusion matrix, while Figures 3.12a and 3.12b display the training and validation loss, as well as the training and validation accuracy, respectively. The MLP model used in this experiment had 300 neurons in each hidden layer, a learning rate of 0.0001, and a momentum of 0.9. Early stopping and weight decay with a value of 0.001 were employed to prevent overfitting. The model achieved an accuracy of 75% and a loss of 0.844 on the test dataset.

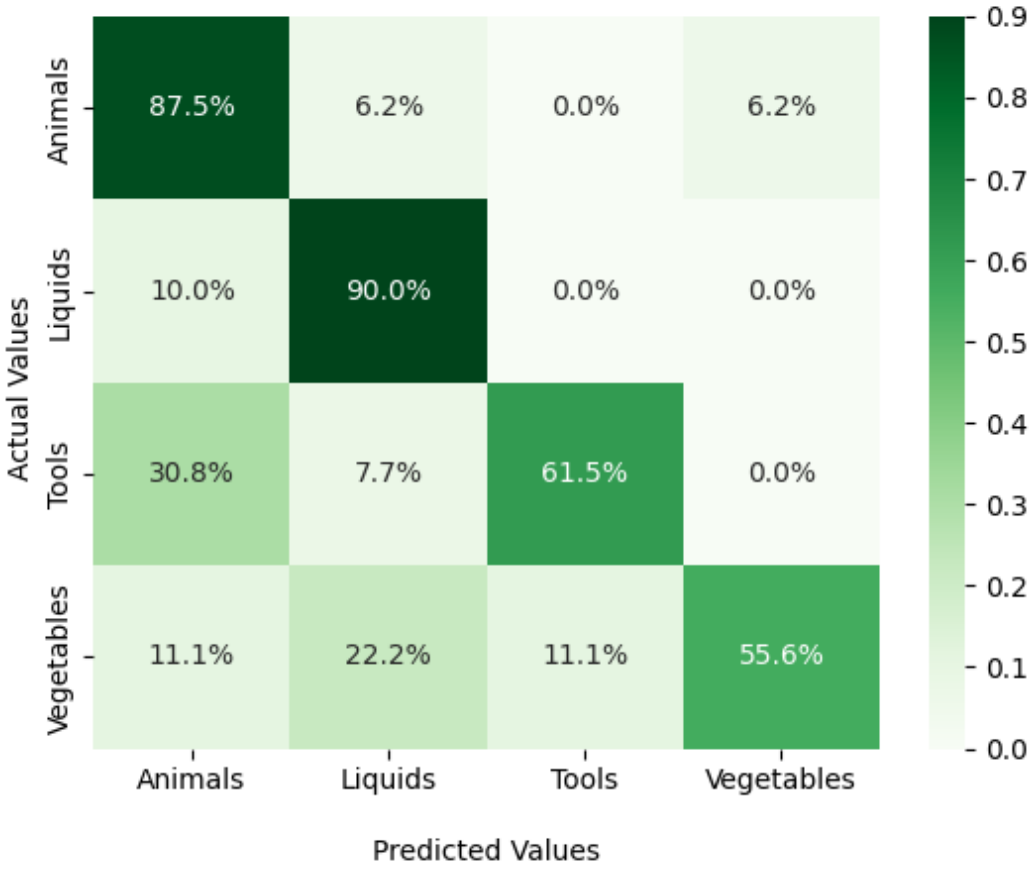
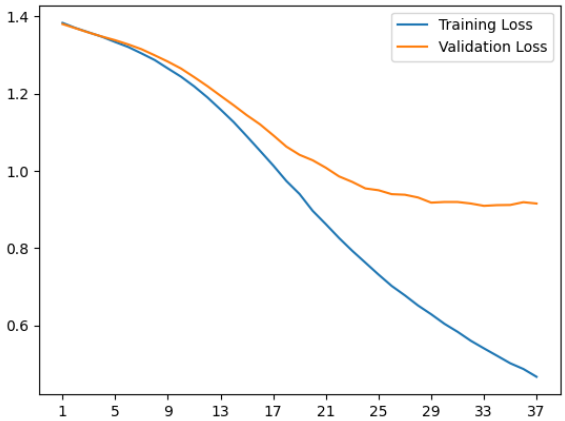
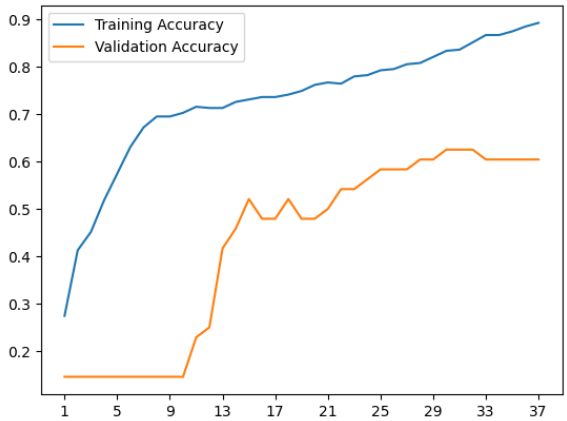


Figure 3.11: Confusion matrix of the multi-class classification using Slovak BERT embeddings



(a) Training and validation loss of multi-class classification using Slovak BERT



(b) Training and validation accuracy of multi-class classification using Slovak BERT

Figure 3.12: Learning curves of multi-class classification using SlovakBERT embeddings as input. The experiment employed hyperparameters with a hidden layer size of 300, a learning rate of 0.0001, and a momentum of 0.9. 10-fold cross-validation was conducted to assess the model's performance.

Discussion

In this work, I employed various methods to evaluate the semantic similarity of word pairs. I utilized two types of word embeddings, namely the Slovak BERT and Word2vec, to determine which one provides more suitable word representations for these experiments.

Using the first method, discussed in Section 2.3, to evaluate the semantic similarity between word pairs, I expected the average Euclidean distance of associated word pairs to be greater than that of dissociated word pairs. However, as discussed in Section 2.3, there were instances where the average distance of dissociated word pairs was greater or equal to that of associated word pairs. Similarly, when considering the cosine similarity of word pairs, I observed similar results, with some cases showing higher cosine similarity for associated word pairs. Additionally, I noticed an overlap in these distances, making it difficult to definitively determine whether a word pair is associated or dissociated based solely on these distances. Consequently, my initial expectation was not confirmed.

In the binary classification, presented in Section 2.4, high scores were achieved for both training accuracy and validation accuracy. However, the model's performance on the test data from different task, presented in 2.3.1, was poor. One possible reason for this discrepancy is that, in the training set, words are compared based on the category they belong to.

For example, in the training data, the words "pivo" (beer) and "krém" (cream) are both categorized as liquids and therefore they are associated with each other. However, in the test data, this word pair is disassociated. In another case, the word "palivo" (fuel) is a liquid and "auto" (car) is a tool, so based on their categories, they are disassociated with each other. However, in the test data, this word pair is marked as associated. Similarly, the word "rak" (crayfish) is an animal and "rieka" (river) is a liquid. Therefore, in the training data, they are disassociated with each other. However, in the test data, they are also associated.

In the multi-class classification, discussed in Section 2.5, the model achieved a higher accuracy when using Word2Vec embeddings, compared to using Slovak BERT. This outcome was unexpected because there was an anticipation for a higher accuracy with Slovak BERT. One possible reason for this difference could be that Slovak BERT

is a contextual language model, but during the process of obtaining the embeddings, described in Section 2.2, the model only receives isolated words without any contextual information.

One of the conclusions drawn from this work is that evaluating semantic similarity based on word pairs is challenging without considering the specific experimental context in which the associated or dissociated word pairs occur. The findings highlight that relying solely on word pair analysis, such as measuring distances or similarities, may not provide a definitive indication of the semantic relationship between words. The observed overlap in the distances and similarities of associated and dissociated word pairs further emphasizes the need to consider the context and experimental setup to accurately determine the association or dissociation of word pairs. Therefore, to effectively assess semantic similarity, it is crucial to take into account the context in which the word pairs are presented during the evaluation process. Due to significant variability among subjects, it is nearly impossible to evaluate the semantic similarity of words based on euclidean/cosine metrics.

Another finding from this study is that using Word2vec embeddings exhibited faster learning compared to the Slovak BERT embeddings. Furthermore, when training the MLP on Word2vec embeddings, higher accuracy was achieved compared to using Slovak BERT embeddings. This discrepancy in performance can be attributed to the contextual nature of Slovak BERT. During the process of obtaining Slovak BERT embeddings, only isolated words were considered, lacking the contextual information that the model is designed to capture.

Conclusion

In conclusion, this study aimed to evaluate the semantic similarity of word pairs using two types of word embeddings: Slovak BERT and Word2vec. The results showed that the expected relationship between associated and dissociated word pairs based on distances and similarities was not always consistent. This indicates that evaluating semantic similarity based solely on word pair analysis is challenging without considering the context. The study highlighted the importance of considering the context and experimental setup to accurately determine the association or dissociation of word pairs.

Furthermore, the study found that Word2vec embeddings exhibited faster learning and achieved higher accuracy in the multi-class classification compared to Slovak BERT embeddings. In the binary classification, the difference in performance can be attributed to the contextual nature of Slovak BERT, as the embeddings were obtained from isolated words without contextual information. However, it is worth noting that the performance of learning using Word2vec and Slovak BERT embeddings was comparable in the binary classification task.

There are several potential avenues for future research in the field of assessing the similarity of word pairs in terms of their meaning. One potential area for improvement involves utilizing higher-quality data during the training and evaluation processes. In the current study, existing datasets were relied upon, but there is potential for enhancement by incorporating larger and more diverse datasets.

Additionally, a valuable approach for future work could involve utilizing datasets that provide context for obtaining Slovak BERT embeddings. As mentioned previously, one limitation of the current study was that Slovak BERT embeddings were derived from isolated words, which may not fully capture the contextual information that the model is designed to comprehend.

By addressing these aspects, it should be possible to achieve improved results in this field.

Appendix

This section provides a condensed overview of the folders encompassing Python scripts, datasets, tables showcasing the results, and some of their visualizations.

- **word2vec-embeddings**: This folder contains scripts for obtaining word2vec embeddings of words from various Excel and CSV files. Additionally, it includes scripts for comparing the distances between word pairs. The following files are included:
  - `ChainAssociationTask.py`, `DiscreteTask.py`, `FluencyTask.py`: These scripts process files with words, calculate the distances between word pairs using word2vec embeddings, and write the results to an Excel file.
  - The directories named **Chain**, **Discrete**, and **Fluency** contain Excel and CSV files that are utilized in the Chain association task, Discrete task, and Fluency task, respectively. These files are specifically linked to the tasks described in Subsections 2.3.2, 2.3.3, and 2.3.1.
  - The **results** folder contains the Excel files of the results obtained by evaluating the semantic similarity using the distances of word2vec word embedding pairs. The files included are: `Chain1.xlsx`, `Chain2.xlsx`, `Discrete1.xlsx`, `Discrete2.xlsx`, `SVF_aso_dis_w2v.xlsx`
  - The **figures** folder contains the visualizations of the results.
  - The `GenerateTrainingData.py` and `GenerateTestData.py` files contain Python code responsible for processing files that contain words for both training and testing. These scripts generate files that include the corresponding word embeddings. The folders **train_data** and **test_data** contain the input and output data of the above mentioned scrips.
  - The `utilities.py` file contains auxiliary functions for computing distances and visualizing the results.
- **slovakBERT-embeddings**: This folder contains scripts for obtaining slovakBERT embeddings of words from various Excel and CSV files, along with scripts for comparing distances between word pairs. The following files are included:

  - `ChainAssociationTask.py`, `DiscreteTask.py`, `FluencyTask.py`: These scripts process files with words, calculate the distances between word pairs using slovakBERT embeddings, and write the results to an Excel file.
  - The directories **Chain**, **Discrete**, and **Fluency** contain Excel and CSV files containing inputs for the previously mentioned python programs.
  - The folder named **results** stores Excel files that contain the results obtained from the evaluation of semantic similarity using the distances between SlovakBERT word embedding pairs. The files included are: `Chain1.xlsx`, `Chain2.xlsx`, `Discrete1.xlsx`, `Discrete2.xlsx`, `SVF_aso_dis_w2v.xlsx`
  - The **figures** folder contains the figures presenting the results.
  - The `GenerateTrainingData.py` and `GenerateTestData.py` files perform the processing of files containing words for training and testing purposes, creating files with the corresponding SlovakBERT word embeddings. The folders **train_data** and **test_data** contain the relevant files linked to these scripts.
  - The `utilities.py` file contains helper functions for computing distances and visualizing the results.

- **TrainMLP** This folder contains scripts for handling data and training the MLP.
  - The `data_handlerBCT.py` file is responsible for data preparation for the binary classification, and the `BinaryClassificationTask.py` file contains the script for training.
  - Similarly, the `data_handlerMCT.py` file handles data preparation for the multi-class classification, and the `MultiClassClassificationTask.py` file contains the script for training.
  - The file named `SVF_word_vectors_w2v.csv` comprises words along with their corresponding word2vec embeddings, intended for training purposes. Similarly, the file called `SVF_word_vectors_bert.csv` includes the same words accompanied by their corresponding SlovakBERT embeddings.
  - The folders named **test_data_slovakBERT** and **test_data_word2vec** contain word pairs along with their corresponding word2vec and SlovakBERT embeddings. These folders are utilized for testing purposes.

Bibliography

[1] Judit Ács, Ákos Kádár, and Andras Kornai. Subword pooling makes a difference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2284–2295, Online, April 2021. Association for Computational Linguistics.

[2] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[3] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[5] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. In *Journal of machine learning research*, 2011.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[9] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. Ammus : A survey of transformer-based pretrained models in natural language processing, 2021.

[10] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[11] Veysel Kocaman and David Talby. Spark nlp: Natural language understanding at scale. *Software Impacts*, page 100058, 2021.

[12] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[13] Martin Marko, Drahomír Michalko, Jozef Dragašek, Zuzana Vančová, Dominika Jarčušková, and Igor Riečanský. Assessment of automatic and controlled retrieval using verbal fluency tasks. *Assessment*, 0(0):10731911221117512, 0. PMID: 35979927.

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[15] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations, 2017.

[16] The pandas development team. pandas-dev/pandas: Pandas, January 2023. If you use this software, please cite it as below.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] Matúš Pikuliak, Štefan Grivalský, Martin Konôpka, Miroslav Blšták, Martin Tamajka, Viktor Bachratý, Marián Šimko, Pavol Balážik, Michal Trnka, and Filip Uhlárik. Slovakbert: Slovak masked language model, 2022.

[19] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[20] T. Tieleman and G. Hinton. Lecture 6.5 - rmsprop, coursera: Neural networks for machine learning. technical report. 2012.

[21] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[23] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online, October 2020. Association for Computational Linguistics.

[24] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning, 2023.

[25] Slovak Academy of Sciences Ľ.Štúr Institute of Linguistics. Slovak national corpus. `https://korpus.juls.savba.sk/`. Accessed: 2023-05-30.