Contents

Related theory

Definition 1 (Eigenvector, eigenvalue). We can define the eigenvalue λ of matrix A as follows:

$$A\mathbf{x} = \lambda\mathbf{x}$$

where $\mathbf{x}$ is a non-zero vector. The corresponding eigenvectors $\mathbf{x}$ of matrix A are the solutions to this equation. ▪

Definition 2 (Covariance). The covariance between two random variables X and Y is defined as:

$$\text{cov}(X, Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

where $\mathbb{E}$ denotes the expected value. In other words, it measures the degree to which two variables are linearly related, with positive values indicating a positive relationship and negative values indicating a negative relationship. ▪

Definition 3 (Correlation). The correlation between two random variables X and Y is defined as:

$$\text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y}$$

where $\text{cov}(X, Y)$ is the covariance between X and Y, and σ_X and σ_Y are the standard deviations of X and Y, respectively. Correlation measures the strength and direction of the linear relationship between two variables, with values ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation), and 0 indicating no correlation.▪

Definition 4 (Covariance matrix). The covariance matrix of a set of n random variables $X_1, X_2, \ldots, X_n$ is an $n \times n$ matrix $\mathbf{\Sigma}$, where the (i, j)-th element is given by:

$$\mathbf{\Sigma}_{i,j} = \text{cov}(X_i, X_j)$$

In other words, the diagonal elements of the covariance matrix represent the variances of the individual variables, and the off- diagonal elements represent the covariances between pairs of variables. The covariance matrix is symmetric and positive semi-definite. ▪

Definition 5 (SVD). The Singular Value Decomposition (SVD) of a matrix $\mathbf{A}$ is a factorization of $\mathbf{A}$ into the product of three matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{\Sigma}$ is a diagonal matrix with non-negative entries known as the singular values of $\mathbf{A}$. ▪

Definition 6 (Cosine similarity).

$$similarity_{cos}(\mathbf{a}, \mathbf{b}) = \frac{\langle \mathbf{a} \cdot \mathbf{b} \rangle}{||\mathbf{a}||_2 ||\mathbf{b}||_2}, \text{where } \mathbf{a}, \mathbf{b} \in \mathbf{R}^n$$

Definition 7 (softmax). The softmax function is defined as follows:

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ is a vector of real numbers and N is the dimensionality of the vector. ▪

Introduction to word embeddings

2.1 Idea behind word embeddings

Understanding language is a fundamental human skill that we often take for granted. It may seem like a simple task, but there is a complex process that takes place within our brains to make it possible. When we hear a word, our brains do not just store it as a sequence of letters or sounds. Instead, it encodes the word's meaning based on its relationships with other words and concepts in our mental lexicon. This process involves a web of connections between different parts of the brain.

As human societies evolved and developed written language, new challenges emerged in processing and analyzing large volumes of text. This need led to the creation of a new field called Natural Language Processing (NLP), which focuses on developing algorithms and techniques to enable computers to understand and process human language. NLP involves the application of computational methods to text and speech data in order to derive meaning and enable intelligent interactions between humans and machines.

One of the key challenges in NLP is representing words in a way that enables computers to understand their meaning. To achieve this, we can use a technique known as word embedding. Word embedding involves representing words with vectors in a high-dimensional space, where each dimension represents a different feature or attribute of the word. By mapping words to vectors in this way, we can capture the semantic and syntactic relationships between words and enable computers to understand the context in which words are used.

The goal of word embedding is to create representations that reflect the relationships between words, meaning similar words should be represented with similar vectors, so we can measure similarity with the distances of the vectors. There are two main categories of word embeddings: static and dynamic.

2.2 Static word embeddings

Static embeddings are a type of word representation used in Natural Language Processing.
They are generated by training on large corpora of text and do not change once they are created. The goal is to represent words with similar meanings using similar vectors, so they can be used interchangeably in various language processing tasks. This process of word embedding involves representing words as vectors in a high-dimensional space, with each dimension representing a particular feature or relationship.
One of the earliest and most influential works in static embeddings is the LSI/LSA (Latent Semantic Indexing/Analysis) method, introduced by Deerwester et al. [1]. This method represents each word as a vector in R^N, where N is the unique number of words in a given dictionary. The values in this vector are based on the term-document frequencies. The following scatter plot from the article [2] shows the result of clustering articles with the help of LSA. It shows that it is capable of the task.

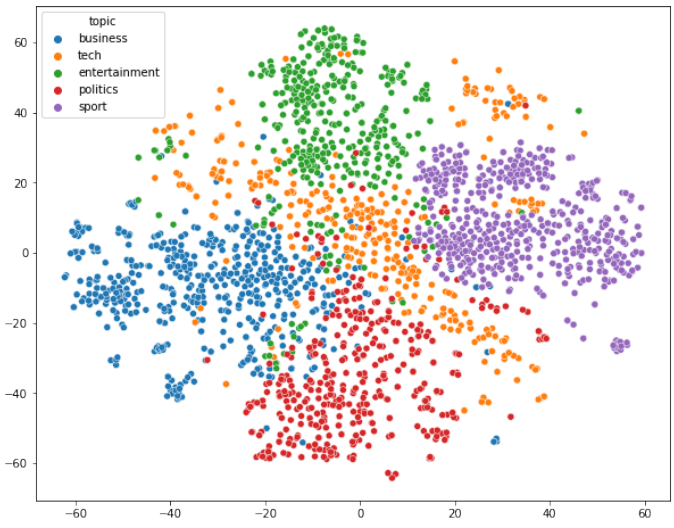

Figure 2.1: Clustering vectors obtained by LSA

However, this method outputs extremely high-dimensional vectors, making it computationally expensive and challenging to work with. To address this issue, they empirically discovered that these embeddings can be reduced to 300-dimensional vectors by doing a rank-300 SVD on the original NxN embeddings matrix. This approach made the computation of the LSA embeddings more feasible, and it is still used today in many applications.

2.2.1 Word2Vec

One of the most popular static embedding method is Word2Vec, presented in Mikolov et al. [3]. Word2Vec has two main models: the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. The CBOW model predicts target words from source context words, while the Skip-Gram model predicts source context words from the target words. Both models have their advantages and are used extensively in various applications. This subsection's goal is to give a little bit more insight on the architecture of *Word2Vec* and it is based on the paper of Goldberg and Levy [4].
To understand the *Word2Vec* architecture, it is essential to familiarize ourselves with some key concepts. The first of them is the Skip-Gram model. In this model, we are given a corpus of words w, along with their respective contexts, denoted as c. The context is the "neighborhood" of a word within a pre-defined window, such as five or ten words. To accomplish our objective, we will utilize conditional probabilities, denoted as $p(c|w)$, and work with a corpus *Text*. Our goal is to optimize the parameters, denoted as θ, of $p(c|w, \theta)$ in order to maximize the overall corpus probability.
Mathematically, we aim to find the argument θ, that maximizes the following corpus probability:

$$\arg\max_{\theta} \prod_{w \in \text{Text}} \left[\prod_{c \in C(w)} p(c|w;\theta) \right]$$

where $C(w)$ represents the set of contexts for the word w. To parameterize this model, we can employ softmax to model the conditional probability $p(c|w;\theta)$:

$$p(c|w;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}}$$

where v_c and v_w belong to the vector space $\mathbb{R}^d$. These vectors represent the embeddings for context c and word w respectively, and C represents the set of all available contexts. The parameters θ correspond to v_{c_i} and v_{w_i} for each word w in V, context c in C, and i in the range 1 to d. Therefore, we have a total of $|C| \times |V| \times d$ parameters.

By taking the logarithm of the equation, we obtain:

$$\arg\max_{\theta} \sum_{(w,c) \in D} \log p(c|w) = \sum_{(w,c) \in D} \left(\log e^{v_c \cdot v_w} - \log \sum_{c' \in C} e^{v_{c'} \cdot v_w} \right)$$

where D represents the training dataset. The assumption is that maximizing this objective function will yield a well-optimized embedding, resulting in similar words being represented by similar vectors. However, computing $p(c|w;\theta)$ can be computationally expensive due to the summation over all available contexts. To mitigate this computational challenge, we can employ hierarchical softmax, which can be used as a tool to efficiently compute the probabilities of a large number of mutually exclusive events arranged in a hierarchical structure. It decomposes the problem into a binary tree, where each node represents a decision between two child nodes, until reaching the leaf nodes corresponding to the target events.
Another crucial concept to consider is negative sampling, which was introduced by Mikolov as an alternative and more efficient method for deriving word embeddings. Negative sampling, although builds upon the Skip-Gram model, optimizes a different objective. Let (w, c) denote a word-context pair, and let $p(D = 1|w, c, \theta)$ represent the probability that (w, c) originates from the corpus data. The parameters θ control the underlying distribution. Our objective is to maximize the following function:

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(D = 1|w, c; \theta).$$

After undergoing several transformations, the final form of this objective function is as follows:

$$\arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log \sigma(-v_c \cdot v_w),$$

where σ denotes the sigmoid function. Remarkably, this function closely resembles the objective function derived in the skip-gram section. However, unlike the skip-gram model discussed above, this section does not model $p(c|w)$ explicitly. Instead, it models a quantity related to the joint distribution of word w and context c.

Figure 2.2 shows the architecture of Word2Vec, focusing more on the technical side.

The input and output vectors are one-hot encoded, which means that only one dimension has a value of one, and all the other dimensions have a value of zero. The hidden layer in the middle is the n-dimensional embedding itself. The goal is to create an embedding, from which we can recreate the one-hot encoded vector provided as input as close as possible.

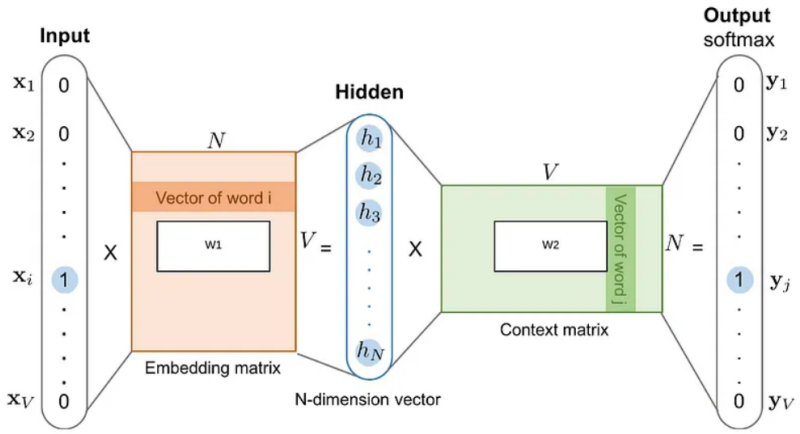

Figure 2.2: Architecture of Word2Vec

2.2.2 GloVe

Another widely used method for generating static embeddings is GloVe (Global Vectors for Word Representation), which was introduced by Pennington et al. [5].
GloVe, short for global vectors, derives its name from the fact that the model directly captures global corpus statistics. The fundamental component of GloVe is a matrix denoted by X, which contains the word-word co-occurrence counts. Specifically, X_{ij} represents the number of times word j occurs in the context of word i. Let's define some terminology: $X_i = \sum X_{ik}$, which represents the total number of times any word appears in the context of word i. Additionally, $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$ denotes the probability of word j appearing in the context of word i.
To illustrate this concept, let's consider an example with i = jég (ice) and j = gőz (steam). The relationship between these two words can be examined by analyzing the ratio of their co-occurrence probabilities with various probe words denoted by k (this could be 'szilárd' (solid) or 'gáz' (gas) in the example). If the probe word k is related to i but not to j, we expect the ratio $\frac{P_{ik}}{P_{jk}}$ to be large. On the other hand, if k is unrelated to both i and j, then the ratio should be close to one. This observation suggests that the appropriate starting point for word vector learning should involve the ratios of these co-occurrence probabilities, rather than the probabilities themselves.
The general model formulation takes the following form:

$$F(w_i, w_j, \hat{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Here, the vectors $w \in \mathbb{R}^d$ represent word vectors, and $\hat{w} \in \mathbb{R}^d$ represents separate context word vectors.
To encode the information present in the ratio $\frac{P_{ik}}{P_{jk}}$ into the word vector space, we modify

the function F. This results in the updated form of:

$$F(w_i - w_j, \hat{w}_k) = \frac{P_{ik}}{P_{jk}}$$

While the arguments of F are vectors, the right-hand side is a scalar. To maintain the linear structure, we can rewrite the arguments of F as $(w_i - w_j)^T \hat{w}_k$.
In word-word co-occurrence matrices, the distinction between a word and a context word is arbitrary, as the two roles are interchangeable. To ensure consistency, we not only exchange $w \leftrightarrow \hat{w}$ but also $X \leftrightarrow X^\intercal$. Consequently, the final form of the function becomes:

$$F((w_i - w_j)^T \hat{w}_k) = \frac{F(w_i^T \hat{w}_k)}{F(w_j^T \hat{w}_k)}$$

This equation can be solved using the exponential function, $F = \exp$. Thus, the final form of the equation is:

$$w_i^T \hat{w}_k + b_i + \hat{b}_k = \log(X_{ik})$$

Here, b_i represents the bias for the word vector w_i, and $\hat{b}_k$ represents the bias for the context word vector $\hat{w}_k$.
While static embeddings have enabled significant progress in many natural language processing applications, they do have some limitations. One of the primary limitations is their inability to capture contextual information, which is essential for tasks such as language translation and sentiment analysis. However, ongoing research and development of new embedding techniques continue to improve their accuracy and effectiveness.

2.3 Dynamic word embeddings

Static embeddings have been widely used in natural language processing tasks, where each word is represented as a fixed vector, irrespective of its context or any possible ambiguity. This approach can lead to suboptimal results in many scenarios. To address this limitation, contextualized embeddings were introduced by McCann et al. [6], which aim to capture the context-specific information of words by generating a unique vector representation for each word based on its context.

2.3.1 Tokenization

Tokenization is a crucial step in Natural Language Processing, where the text is divided into smaller units known as *tokens*. These tokens can be based on words, subwords

or characters. The simplest form of tokenization involves segmenting text based on whitespaces and punctuation marks. In languages like Hungarian, this approach can be effective since words are separated by whitespaces. However, in languages like Chinese, this approach may not be suitable due to the lack of clear word boundaries.
Modern tokenization techniques employ sophisticated algorithms and statistical models, combined with linguistic knowledge, to achieve more accurate results. Rule-based tokenizers, such as the Natural Language Toolkit (NLTK), utilize regular expressions and predefined language-specific rules for segmentation. Statistical tokenizers, like the Moses Tokenizer, employ statistical models trained on large text datasets to determine token boundaries based on the likelihood of word sequences. Another category of tokenizers is neural network-based, such as Byte Pair Encoding (BPE) [7], which is a data compression technique.
BPE operates by replacing the most common pair of consecutive bytes in the data with a byte that does not occur in the data. For example, consider the string *aaabcaaacab* that we wish to encode. In this case, *aa* is the most common pair, so we can replace it with *D*, a byte that does not appear in the data. Thus, we obtain *DabcDacab*, where *D* represents *aa*. The next most common byte pair is *ab*, which can be replaced with *E*. The encoded data now becomes *DEcDacE*. Now there are only unique byte pairs, so they are not encoded further. This technique can also be adapted for word segmentation by merging frequent character sequences instead of byte pairs.
For instance, let's suppose we have a corpus with the following words and their corresponding frequencies: {'szebb</w>':5,'jobb</w>':2,'kutya</w>':6,'bab</w>':1}. Then the frequencies of the characters happen to be:

Token/Character	Frequency
</w>	14
s	5
z	5
e	5
b	16
j	2
o	2
k	6
u	6
t	6
y	6
a	7

Table 2.1: Character frequencies in the corpus.

Next, we find the most common character pairing, which in this case is b</w>. We create a token out of this pairing, and the frequencies are updated accordingly. The revised

table is shown in Table 2.2 with one more iteration done, where bb</w> also became a token.

Token/Character	Frequency
</w>	14 - 8 = 6
s	5
z	5
e	5
b	16 - 8 = 8
j	2
o	2
k	6
u	6
t	6
y	6
a	7
b</w>	8 - 7 = 1
bb</w>	7

Table 2.2: Token frequencies after two iterations

We can continue this algorithm iteratively until we obtain the desired number of tokens. After multiple iterations, common words will act as single tokens, while rare words will be split into subwords. This characteristic is particularly useful in tasks like machine translation, as it ensures that rare words are also appropriately handled by splitting them into more frequent words/tokens. In agglutinative languages like Hungarian, we expect the root of a word to be one token and the suffixes to be separate tokens.
It is important to note that the quality of tokenization significantly impacts downstream NLP tasks. Therefore, it is essential to pay close attention to this step to ensure accurate and meaningful results.

2.3.2 Attention

The concept of attention was introduced to address the challenge of capturing long-distance dependencies in an input sequence. More specifically, when dealing with text this could mean that a name, such as 'János' appears in the first sentence, and later on referenced as 'he', several sentences later. In such cases, a window of 5-10 words is insufficient to provide context for understanding who 'he' refers to. Attention solves this problem by applying different weights to different parts of the input, allowing the model to focus on relevant information. The *Transformer* architecture, introduced in the paper "Attention Is All You Need" (Vaswani et al. [8]), was designed specifically to capture long-distance dependencies that traditional recurrent and convolutional architectures struggled with.

The *Transformer* model follows an encoder-decoder structure. The encoder maps an input sequence $(x_1, \ldots, x_n)$ to a continuous representation $z = (z_1, \ldots, z_n)$. Given z, the decoder generates an output sequence $(y_1, \ldots, y_m)$ one symbol at a time. The model is auto-regressive, meaning it uses previously generated symbols as additional input when generating the next symbol. The *Transformer* employs stacked self-attention and point-wise fully connected layers for both the encoder and decoder. The complete model is visualized in the figure below, and each component will be explained in detail.

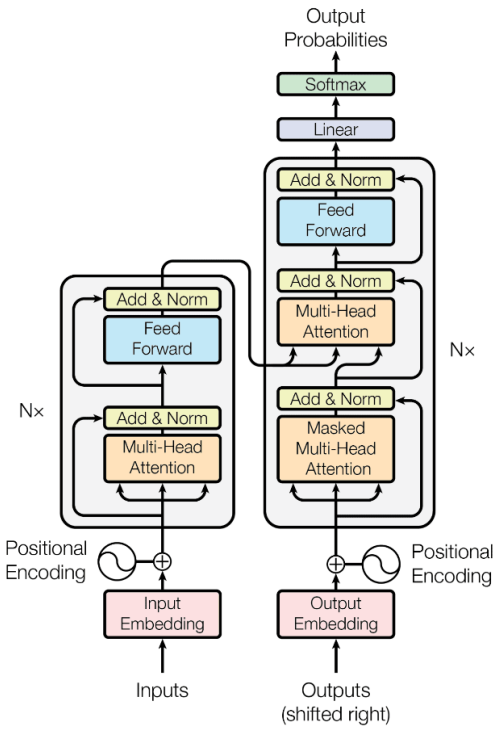

Figure 2.3: Transformer architecture

The encoder consists of $N = 6$ identical layers, each comprising two sub-layers. The first sub-layer is a multi-head self-attention mechanism, enabling the model to focus on different parts of the input sequence simultaneously. The second sub-layer is a position-wise fully connected feed-forward network, which applies a linear transformation independently to each position. Both sub-layers incorporate a residual connection, where the output of the sub-layer is added to the input, followed by layer normalization to ensure stable training. Mathematically, if we represent the function implemented by the sub-layer as $Sublayer(x)$, the output is obtained as $LayerNorm\ (x + Sublayer(x))$. Notably, all sub-layers in the model produce outputs with a dimensionality of $d = 512$, ensuring a consistent representation across the layers.

Similar to the encoder, the decoder also consists of $N = 6$ layers. It includes three sub-layers, two of which are the same as those in the encoder. The third sub-layer performs multi-head attention over the output of the encoder stack. The decoder

section also incorporates residual connections and layer normalization around the sub-layers. However, the self-attention sub-layer is modified to prevent positions from attending to subsequent positions. This masking ensures that predictions for position i can depend only on the known outputs at positions less than i, considering that the output embeddings will be offset by one position.
An attention function maps a query and a set of key-value pairs to an output, where all these components are vectors. The output is obtained by taking a weighted sum of the values, and the weights are computed using a compatibility function between the query and the corresponding key. The *Transformer* architecture utilizes two types of attention: Scaled Dot-Product Attention and Multi-Head Attention.

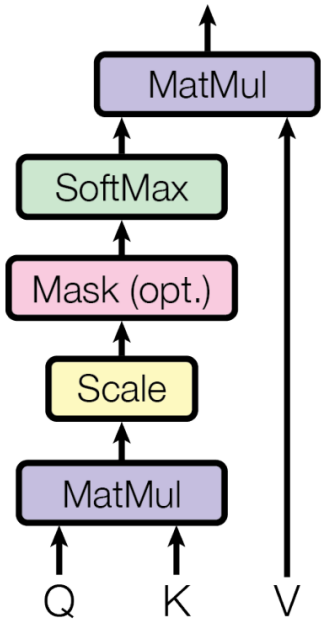

Figure 2.4: Scaled Dot-Product Attention

In Scaled Dot-Product Attention, the input consists of queries and keys of dimension d_k, and values of dimension d_v. The dot products of the queries with all keys are computed and divided by $\sqrt{d_k}$. Then, the softmax function is applied to obtain the weights for the values. The attention function can be computed simultaneously on a set of queries, which are packed together into a matrix Q. Similarly, the keys and values are packed into matrices K and V. The output is computed using the following formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The most common types of attention functions are additive and multiplicative (dot-product) attention. Dot-product attention, as described above, is identical to the Scaled Dot-Product Attention, except for the scaling factor of $1/\sqrt{d_k}$. On the other

hand, additive attention computes the compatibility function using a feed-forward network with a single hidden layer. Although they have similar theoretical complexity, multiplicative attention is much faster and space-efficient in practice due to highly optimized matrix multiplication. The scaling factor of $1/\sqrt{d_k}$ is crucial because with large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where the gradients are extremely small.

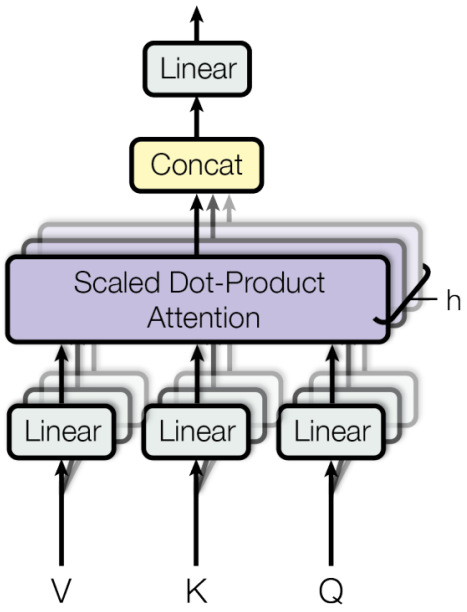

Figure 2.5: Multi-Head Attention

The other attention function used in the Transformer architecture is Multi-Head Attention. Instead of a single attention function, the queries, keys, and values are linearly projected h times using different learned linear projections to dimensions d_k, d_k, and d_v, respectively. The attention function is evaluated on each projected version, resulting in outputs of dimension d_v. These outputs are then projected again to obtain the final values. Multi-Head Attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this capability. Mathematically, the Multi-Head Attention can be represented as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

The projections are parameter matrices: $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{h \times d_v \times d_{\text{model}}}$. In the paper, $h = 8$ heads are used, and $d_k = d_v =$

$d_{\text{model}}/h = 64$. In summary, the computational cost of Multi-Head Attention is similar to that of single-head attention with full dimensionality.

2.3.3 Dynamic models

One of the most prominent models for contextualized embeddings is BERT (Bidirectional Encoder Representations from Transformers), which was introduced in Devlin et al. [9]. BERT is a pre-trained deep learning model that uses the *Transformer* architecture and self-attention, explained in *Chapter* 2.3.2.
BERT is a bidirectional model, which means that it can take into account the context of words from both directions in a sentence, making it more powerful than models that rely only on left-to-right or right-to-left context. For this thesis, the Hungarian huBERT model will be used introduced in Nemeskey [10], which was trained on the Hungarian Webcorpus 2.0, a 9-billion-token corpus of Web text collected from the Common Crawl. This model outperforms multilingual BERT in masked language modeling by a significant margin and achieves state-of-the-art performance in named entity recognition and NP chunking. The following graphic shows the working of BERT. With the tokenized input we get 768 dimensional vectors for each token. This happens on each layer, so we can extract vectors from all layers.

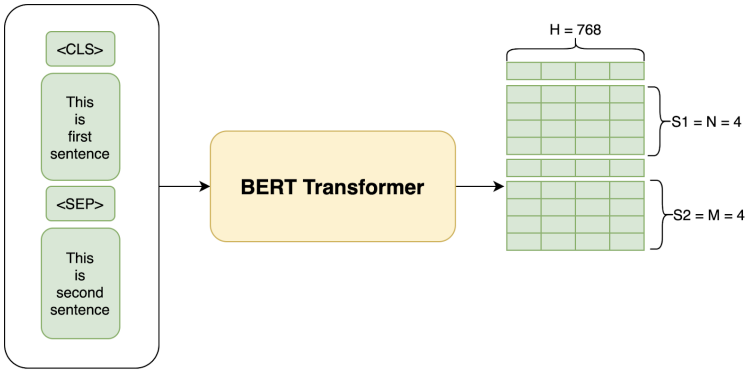

Figure 2.6: Architecture of BERT

Overall, the use of contextualized embeddings like BERT and huBERT can significantly improve the performance of natural language processing tasks, particularly in scenarios where context plays a critical role in determining the meaning of words.

2.4 Extracting vectors from huBERT

I used huBERT-base for my experiments, which is a pre-trained deep learning model, consisting 12 transformer layers, each of which is composed of a self-attention mechanism

and a feed-forward neural network. The output of each layer is then passed on as the input to the next layer.
To create embeddings for words using huBERT-base, the first step is to tokenize the text *Tokenization* is explained in *Chapter* 2.3.1. In this case, each word is converted into a sequence of tokens that can be fed into the huBERT-base model. For my experiments, I used the tokenizer of hubert-base-cc.
After a word is tokenized and fed through the transformers, we can access the output of each layer. This output will consist of one vector for each token that the word was split into. To obtain a single vector for the whole word, I used pooling, which is a technique used to reduce the dimensionality of input data by summarizing it. However, there are multiple ways to assign a single vector to a word using pooling, so I tested several methods to determine which performed best for my experiments.

Analogy tasks

3.1 Introduction to analogy tasks

Analogy tasks are a popular way to assess the ability of language models to capture the complex relationships between words. These tasks are language modeling tasks that involve identifying relationships between words in the form of analogy questions. While the roots of this concept can go back to the work of linguist Roman Jakobson in the 1950s, who was known for his structuralist approach and focused on the analysis of language as a system of signs and the relationships between those signs, the recent advances in deep learning and natural language processing have brought renewed interest to the field.

Analogies are typically presented in the form of word pairs, such as "férfi:nő" ("man:woman") and "király:királynő" ("king:queen"), from which the first three is given and the model is required to identify the fourth word that is related to the first three words in the same way. These tasks have been used to evaluate the performance of language models in capturing the complex relationships between words, such as semantic and syntactic similarity.

In the field of Natural Language Processing, there has been an increasing focus on developing language models that can handle a wide range of tasks, from basic language understanding to more complex tasks such as question answering, sentiment analysis, and language translation. Analogy tasks are a challenge that language models need to tackle to demonstrate their ability to understand and model the underlying structure of language. While they are a simple concept on the surface, they require the model to possess a deep understanding of the nuances of language and the relationships between words.

In addition to evaluating the performance of language models, analogy tasks have also been used to study the cognitive processes involved in language processing. They have

a long history in cognitive psychology and linguistics, and have been used to explore how the human brain processes and represents language so it can be used in a similar fashion in computer linguistics. Overall, analogy tasks are a valuable tool for both evaluating the performance of language models and gaining insights into the cognitive processes involved in language processing. The next plot from [11] shows a visualization of analogy tasks . Country names are paired with their capitals, which is a common analogical question. The visualization is in 2 dimensions for easy understanding. We can observe similar directions with the difference vectors but not in all cases.

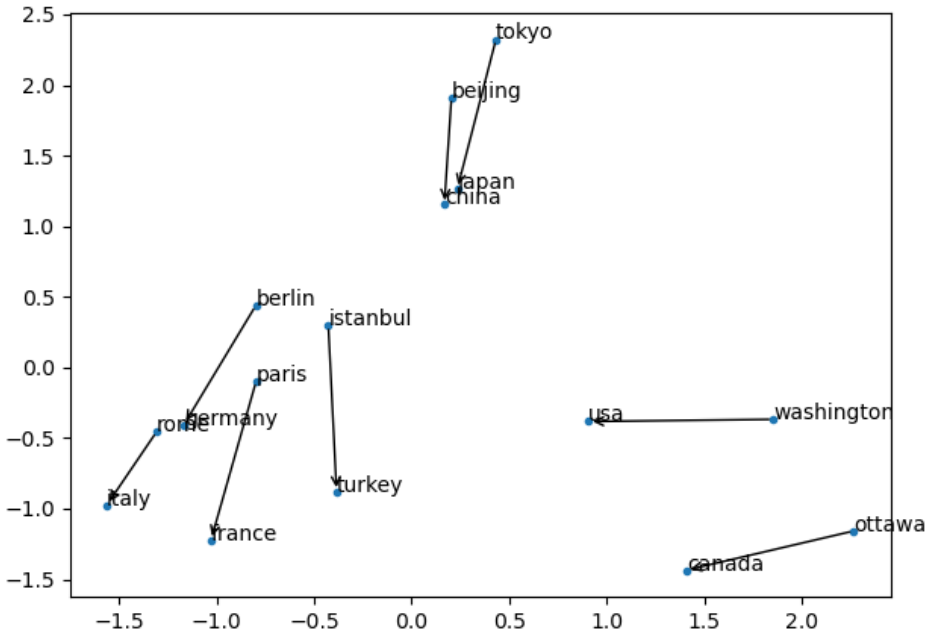

Figure 3.1: Visualizing analogy tasks

3.2 Mathematics of analogy tasks

To solve an analogy task, a vector arithmetic operation is performed on the word embeddings. Specifically, the difference vector between the embeddings of the two source words is computed, and this difference vector is then added to the embedding of the third source word. The resulting vector is then compared to the embeddings of all words in the vocabulary to find the word whose embedding is closest to the resulting vector. As default, the closeness is measured in cosine similarity.
Let **w1**,**w2**,**w3** be the embeddings of the three source words in the analogy task and d be the difference vector between the embeddings of **w1** and **w2**. Then, the resulting vector

r can be obtained by adding d to the embedding of **w3**:

$$\mathbf{r} = \mathbf{w3} + \mathbf{d}$$

The resulting vector r is then compared to the embeddings of all words in the vocabulary to find the word whose embedding is closest to r in terms of cosine similarity.
For my experiments, I used the analogical question set published by Makrai [12] which were created in a similar fashion to the analogical questions made publicly available by Mikolov. These questions are widely used in NLP research and provide a standard set of questions for evaluating the performance of language models on analogy tasks. The use of these questions allowed me to compare the performance of my models to other state-of-the-art models on a standardized set of tasks.

3.3 Specifics of the analogy questions

The analogical questions can be divided into two categories: morphological or grammatical, and semantic. The morphological category includes pairs of words that share a common morpheme, such as the root word and its derived form (e.g., "boldog" and "boldogan"), or pairs of words that differ in tense or gender (e.g., "fut" and "futott"). The semantic category includes pairs of words that share a similar meaning or belong to the same semantic class (e.g. "macska" and "kutya" are both animals).
There are a total of 21396 questions in the analogical pairs dataset, with 14681 semantic and 6715 morphological questions. Within the morphological category, there are nine subcategories, such as *gram1-adjective-to-adverb* or *gram3-comparative*, each with its own set of questions. In the semantic category, there are five subcategories, such as *currency* or the largest subcategory, *capital-world*.
Both the morphological and semantic questions were created by matching every pair with every other pair, resulting in a large number of possible question pairs. For example, the *family* category generates $\binom{20}{2}$ or 190 possible question pairs, each of which involves finding a fourth word that is related to the first three words in the same way.

3.4 Preparation ahead of the tests

In my research, I compared a 600 dimensional static word embedding (which is available on the site of the efnilex project [13]) to the embeddings extracted from huBERT-base. The static word embedding I used for comparison has a vocabulary of almost 2 million words. However, this large vocabulary includes a lot of noise that can negatively affect

the performance in the analogy tasks. To filter out this noise, I downloaded word frequencies from the two sources of the static embedding, the Hungarian Webcorpus [14] and the Hungarian National Corpus [15]. I then merged these frequencies, creating a new dataset where the first column contains a word and the second column contains the sum of the two word counts found in the two sets.
To filter the vocabulary, I sorted the merged dataset and examined the size of the intersect with the vocabulary of the analogical questions. Using a threshold of 100, I found that 90% of the vocabulary in the analogical questions was still present in the filtered set. This resulted in a vocabulary of around 480,000 words, which was used in my tests.
This filtering process helped to remove unnecessary noise from the static word embedding, resulting in a more focused vocabulary for the analogical questions. By reducing the vocabulary size, the embedding was able to capture the semantic and syntactic relationships between words better, ultimately leading to more accurate results in the analogy tasks.

3.5 Analogy task results

I used the 600-dimensional static word embedding, which was filtered in the way written before as a reference point to compare with the performance of the huBERT language model.
To evaluate the performance of huBERT, I first ran the analogy task on the static embedding, which served as a benchmark. I evaluated the results separately for both morphological and semantic questions, as well as combined. The static embedding performed at 54.95% in the morphological questions and at 24.42% in the semantic ones, resulting in an overall performance of 42.36%
Next, I attempted to match or exceed the performance of the static embedding using huBERT. I began by extracting a static vector from a single layer of huBERT, using mean pooling with an attention mask. Mean pooling is a technique used to condense the information in a matrix or tensor by taking the mean value of the elements. Mean and max pooling are visualized in *Figure* 3.2.

In this case, it was applied to the embeddings generated by huBERT for each word in the analogy task. An attention mask was used to prevent irrelevant tokens from contributing to the final vector. I applied this technique to the first and last layer of huBERT and found that the first layer performed better on the morphological questions, while the last layer performed better on the semantic ones, which is a common observation that the first layers contain more morphological information while the last layers more semantic information.
After experimenting with extracting static vectors from a single layer, I tried using

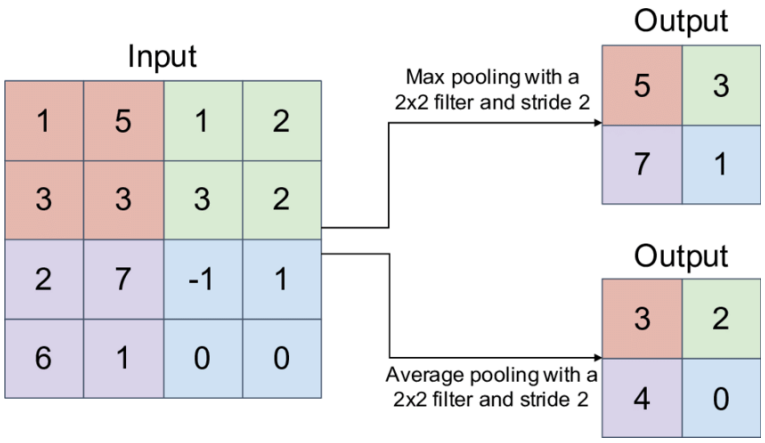

Figure 3.2: Visualizing pooling

several layers and performing separate pooling on them. I then combined the pooled results and performed another pooling to get a single static vector. This approach proved to be the most efficient, allowing me to match the performance of the static embedding. Specifically, it achieved 64.73% in morphological questions, 13.41% in semantic questions, and 43.21% overall, which was comparable to the performance of the static embedding. The next table shows the detailed comparison between the tested methods. *double_at* stands for the technique, where I did a pooling on the result for all the layers too.

method	morph	semantic	total
double_at	64.7%	13.4%	43.2%
hunembed	55.0%	24.4%	42.4%
mean_pooling	51.6%	2.7%	31.1%
last_token	51.6%	2.1%	30.8%
last_layer	34.0%	11.9%	24.7%
max_pooling	26.5%	0.5%	15.6%

Table 3.1: Comparison of methods/embeddings

Principal Component Analysis

4.1 Principal Component Analysis in NLP

In the field of Natural Language Processing (NLP), dealing with large datasets and high dimensionality is a common practical challenge. Hence, there is a need to reduce dimensionality while minimizing information loss. Principal Component Analysis (PCA) is a technique capable to achieve this goal by reducing the number of dimensions while preserving essential information. The reduction in dimensionality not only facilitates easier interpretation of the data but also enables the elimination of noise and irrelevant information, thus allowing a focus on the most significant features that capture word relationships.

In the context of word embeddings, PCA can be used to reduce the number of dimensions in a word embedding model while still retaining the most important features of the data. This can lead to improved performance in various NLP tasks, including analogy tasks. It can also be used to improve the performance of NLP models in other tasks. For example, in sentiment analysis, PCA can be used to reduce the dimensionality of the input data and improve the accuracy of the model.

PCA serves as a valuable tool for identifying the essential features within the data and removing redundant or irrelevant information, which in turn can enhance overall performance. This technique aims to identify uncorrelated variables that maximize variance through the utilization of eigenvalues and eigenvectors. Notably, the adaptability of PCA lies in the fact that the new variables are defined based on the specific dataset at hand, rather than being predetermined. Moreover, various methods have been developed for different tasks, further enhancing its adaptability and applicability.

The concept of principal components was initially introduced by F.R.S. [16] and further formalized by Hotelling [17], who established the mathematical framework for PCA.

However, it is important to note that the practical application of PCA on larger datasets only became feasible with the advent of computers.

4.2 Mathematics of PCA

PCA can be performed using Singular Value Decomposition (SVD), which is a factorization of a matrix into three matrices, described in *'Related theory'*.
The first principal component is defined as the direction that maximizes the variance of the projected data. It can be obtained by computing the eigenvector corresponding to the largest eigenvalue of the covariance matrix of X. Let $\mathbf{C}$ be the covariance matrix of X, and $\mathbf{v}_1$ be the eigenvector corresponding to the largest eigenvalue. The first principal component $\mathbf{PC}_1$ is given by:

$$\mathbf{PC}_1 = X \cdot \mathbf{v}_1$$

where $\cdot$ represents matrix multiplication.

The second principal component is orthogonal to the first principal component and captures the second largest amount of variation. Similarly, the k-th principal component can be obtained by computing the eigenvector corresponding to the k-th largest eigenvalue of the covariance matrix $\mathbf{C}$ of X. Let $\mathbf{v}_k$ be the eigenvector corresponding to the k-th largest eigenvalue. The k-th principal component $\mathbf{PC}_k$ is given by:

$$\mathbf{PC}_k = X \cdot \mathbf{v}_k$$

By computing the first k principal components, we can reduce the dimensionality of the data to k. This way, we only need to calculate the eigenvectors once and always select the first k, enabling us to reduce the dimensionality with steps as we choose.

4.3 Analogy tasks with PCA

In this chapter, I examined the effectiveness of Principal Component Analysis (PCA) on word embeddings. The aim of this investigation is to determine the impact of PCA on performance in analogy questions with respect to the dimensionality reduction.
To perform the analysis, PCA was performed on all the word embeddings evaluated before, and the dimensions were reduced in increments of 50. Subsequently, the performance was evaluated on the same analogy questions as before. The findings

indicate a consistent pattern across all embeddings, where performance initially increased slightly and then decreased gradually before a more rapid drop occurred. The results are visually represented in Figure 4.1. The abbreviations used are f for first layer extractions, l for last layer extractions, and m for mixed layer extractions.

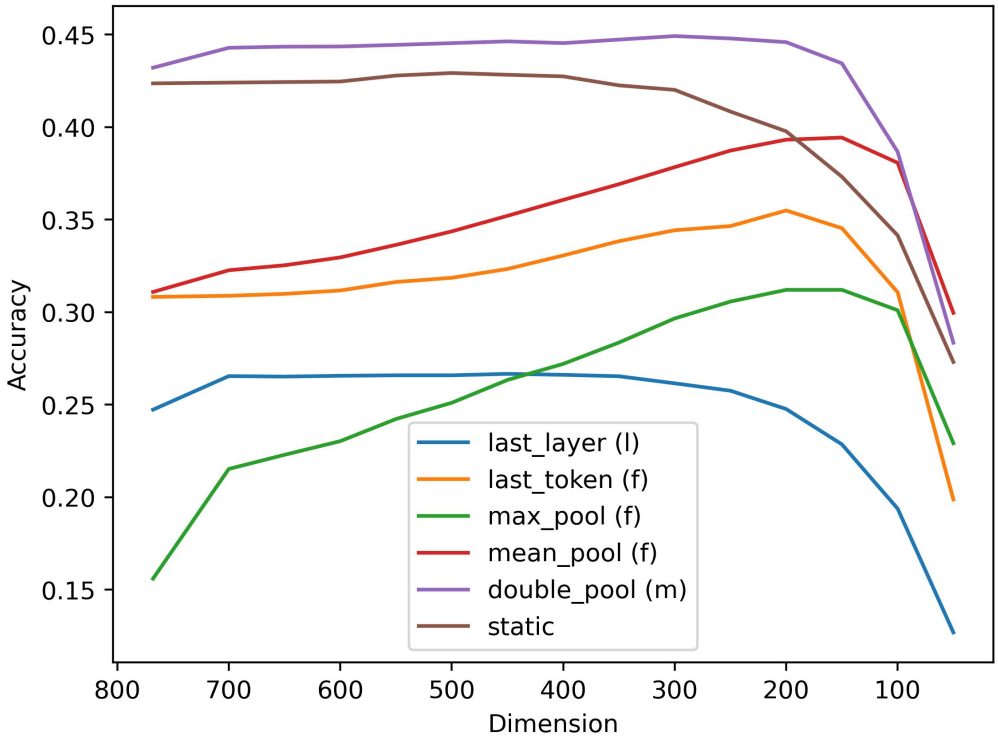

Figure 4.1: The performance of each model

An interesting observation emerged from the analysis, indicating that the embedding with the worst performance, max pooling on the first layer yielded a remarkable improvement to 30% from the original 15% upon performing PCA. Furthermore, we can note that mean pooling performed on the last layer did not improve as much as the first layer or combined layer models, suggesting that embeddings extracted from the last layer are not practical.

The findings suggest that the best method for such tasks is mean pooling using an attention mask. Additionally, in terms of layers, first layer extractions demonstrated superior performance compared to last layer ones, but the most effective approach is to combine the results from multiple layers using another pooling technique.

The outcomes of this investigation suggest that while PCA can be effective in reducing the dimensionality of word embeddings, there may be a limit to how much dimensionality can be reduced before losing significant information. Furthermore, the results indicate that static embeddings can compete with dynamic embeddings,

although the embeddings extracted from huBERT performed slightly better.
In conclusion, these tests provide insight into the effectiveness of PCA on word embeddings and highlights the importance of choosing the appropriate pooling technique when extracting embeddings for downstream tasks.

Ambigutity

5.1 Ambigous words

Ambiguous words possess multiple meanings within the same language. They present a significant challenge in Natural Language Processing and play a crucial role in language understanding. Ambiguity can manifest in two main forms: polysemy and homonymy. Polysemy refers to the phenomenon where a single word has multiple meanings that are related or similar to each other. For example, the Hungarian word "szív" can refer to both the organ "heart" and the action "sucking." These different senses of the word are connected by a shared underlying semantic or conceptual link, as the heart sucks (and pumps) blood.

The concept of polysemy is mentioned by Apresjan [18], who states that it represents the capacity of a word to have several distinct but related meanings, with relatedness being characterized by the presence of common components in their definitions. Polysemy is pervasive across languages, and as Zgusta [19] points out, they are so common that polysemous words can be found on almost every page of a language's dictionary. Various linguistic factors, such as metaphor, metonymy, and cultural influences, can give rise to polysemy, making it a common occurrence in natural languages.

In contrast, homonymy occurs when two or more words share the same spelling but have unrelated or non-similar meanings. For instance, the Hungarian word "ég" can refer to both the "sky" and the act of "burning". Unlike polysemous words, homonymous words do not share any underlying semantic or conceptual connections. Homonymy can arise due to historical changes in language, borrowing from different languages, or coincidental convergence of sounds or spellings. Homonymy is a more straightforward concept compared to polysemy, as the distinct meanings of homonymous words are unrelated and lack semantic overlap. This distinction arises from the fact that polysemy is more prevalent than homonymy.

Differentiating between polysemy and homonymy poses challenges and requires specialized tools. According to Lyons [20], the criteria for distinguishing between these two phenomena on a pre-theoretical level remain uncertain. In some cases, the different meanings of ambiguous words correspond to different word classes. For example, the Hungarian word "nyúl" can be both a verb (to reach for something) and a noun (rabbit). As demonstrated by the previous examples, the various senses of ambiguous words often have different translations in other languages. This linguistic contrast can assist in capturing the frequency of different meanings and not merely their combined occurrence. This topic has been extensively studied in the field, as supported by relevant research [21].
In conclusion, understanding and disambiguating ambiguous words, whether they fall into the category of polysemy or homonymy, is a fundamental task in NLP. The ability to accurately determine the intended meaning of a word in a given context is crucial for building robust language understanding models and applications.

5.2 Word ambiguity in NLP

In NLP, the challenge of dealing with ambiguous words, whether they are polysemous or homonymous, arises due to the need for accurate representation of these words/senses in order to capture their intended meanings in different contexts. Addressing this challenge requires robust techniques to determine the appropriate sense of an ambiguous word in a given context, as well as effective methods to represent these senses.
One common approach to representing polysemous words using word embeddings is to assign different vectors for each sense. By associating distinct vector representations with different senses, the resulting embeddings can capture the inherent variations in meaning. This technique is widely employed in various NLP applications to disambiguate ambiguous words.
Word Sense Disambiguation (WSD) techniques play a crucial role in determining the sense of a word in a given context. These techniques rely on different linguistic features, such as syntactic patterns, semantic cues, or contextual information, to assign specific senses or meanings to ambiguous words. By leveraging the rich information provided by these features, WSD techniques aim to accurately disambiguate ambiguous words and enable the selection of the most appropriate sense for a given context. Once the sense is disambiguated, the corresponding vector representation for that sense can be used.
In addition to the use of static word embeddings that assign different vectors for different senses, the other approach is the use of contextualized word embeddings. These embeddings, exemplified by models such as GPT (Generative Pre-trained Transformer)

and BERT, are specifically designed to capture the meaning of a word within the context of the surrounding words in a sentence. Unlike traditional word embeddings, which assign a fixed vector representation to each word, contextualized word embeddings have the ability to dynamically adjust the word representation based on the context. This dynamic adjustment allows these embeddings to effectively capture the various meanings of ambiguous words in different contexts, accommodating both polysemous and homonymous words.
By incorporating contextual information, contextualized word embeddings enable the modeling of complex linguistic phenomena, such as Word Sense Disambiguation, with greater accuracy. The contextualized nature of these embeddings empowers them to capture the subtle nuances and contextual cues that influence the meaning of a word in different settings.
In summary, addressing the challenge of ambiguous words in NLP requires accurate representation of these words/senses in order to capture their intended meanings across diverse contexts. This can be achieved through the use of word embeddings, where assigning different vectors to different senses is a common practice. Additionally, contextualized word embeddings, such as GPT and BERT, offer a powerful approach by incorporating contextual information to dynamically adjust word representations. By leveraging word sense disambiguation techniques and contextualized embeddings, we can enhance the accuracy of NLP systems in capturing the multiple meanings of ambiguous words, both polysemous and homonymous, in a wide range of applications.

5.3 Extracting vectors for word senses

To conduct experiments using distinct vectors for different senses, I constructed a basic dataset comprising example sentences for each meaning of multiple ambiguous words. For each sense, I gathered a minimum of 15 sentences from Hungarian Wikipedia. From these 15 sentences, I used 10 for creating the embeddings and reserved the remaining 5 for testing the quality of these vector representations.
To obtain contextualized embeddings, I processed the example sentences using huBERT, just like before, while specifically marking the target word. This allowed me to extract the vectors solely for the target word rather than the entire sentence. For pooling the vectors, I adopted the pooling method that demonstrated superior performance in analogy tasks. I performed mean pooling on each layer individually and then conducted pooling on these resulting vectors. As a result, I acquired two distinct vectors representing the two different senses of the word.
To evaluate these embeddings, I used the remaining sentences from the dataset. Similarly, I processed each sentence through huBERT individually, producing different vectors for

each sentence. I then employed cosine similarity to compare these vectors with the two previously obtained vectors representing the word's different senses. This comparison allowed me to measure the accuracy of the embeddings. Specifically, I computed the number of cases in which the cosine similarity with the vector corresponding to the actual meaning of the word was higher than that with the other vector. Dividing this number by the total number of test sentences provided me with the accuracy of the model. Remarkably, the model performed exceptionally well, as the vector corresponding to the correct meaning exhibited higher similarity in all test sentence cases compared to the other vector.

Upon completing these steps, I successfully obtained distinct vectors for the two different meanings of 10 ambiguous words. These words (*Hun*) and the english translation of their two senses (*Eng1* and *Eng2*) can be found in *Table 5.1*. This achievement lays a solid foundation for further investigations and analyses in the subsequent stages of the research.

Hun	Eng1	Eng2
ár	price	flood
dob	drum	throw
ég	burn	sky
ír	write	Irish
nő	woman	grow
nyelv	tounge	language
nyúl	reach	rabbit
rák	cancer	crab
szerv	organ	organization
toll	pen	feather

Table 5.1: Word Translation Table

5.4 Linearity assertion

When creating embeddings, word2vec and most neural network-based models use an objective that involves a nonconvex, nonlinear function of the word co-occurrences. This nonlinearity makes it hard to understand how ambiguous words are represented with static vectors. In Arora et al. [22], a linearity assertion is introduced. Let's consider a polysemous word, say 'szív', which can refer to the heart or the verb suck. Let's suppose that 'szív' is one single token. Then, the word embeddings should satisfy:

$$v_{\text{szív}} \approx \alpha_1 v_{\text{szív1}} + \alpha_2 v_{\text{szív2}}$$

where coefficients α_i are nonnegative and $v_{\text{szív1}}$ and $v_{\text{szív2}}$ are the hypothetical embeddings of the different senses. This finding suggests the presence of a linear structure underneath the highly nonlinear embedding technique. This result can also be applied to Word Sense Induction (WSI), which is introduced in the cited paper.
The justification for the linear assertion comes from the log-linear topic model proposed by Mnih and Hinton [23]. It posits that in the corpus, at every point, there is a micro-topic called discourse, which represents what is being discussed. Based on this concept, a Gaussian random walk model can be created, which is a modification of the random walk model presented in Arora et al. [24].
For each word, there is a vector $v_w \in \mathbb{R}^d$. The model assumes that the corpus is generated as follows: A discourse vector c is drawn from a Gaussian distribution with mean 0 and covariance Σ. Then, a window of n words $(w_1, w_2, ..., w_n)$ is generated from c with the following probability:

$$\Pr[w_1, w_2, ..., w_n|c] = \prod_{i=1}^{n} \Pr[w_i|c]$$

$$\Pr[w_i|c] = \frac{e^{c \cdot v_{w_i}}}{Z_c}$$

where $Z_c = \sum_w e^{v_w \cdot c}$ is the partition function. Another assumption is that $Z_c \approx Ze^{\|c\|^2}$ for some constant Z, based on [22] [Lemma 2.1].

Theorem 1. Assume the above generative model, and let s denote the random variable of a window of n words. Then, there is a linear transformation A such that:

$$v_w \approx A\mathbb{E}\left[\frac{1}{n}\sum_{w_i \in s} v_{w_i} | w \in s\right]$$

The significance of this theorem is that it shows the existence of a linear relationship between the vector of a word and the vectors of the words in its context. This means that if we take a word w and for each window containing that word, compute the average of the vectors of the words in s and denote it as v_s, then take the average of v_s over all the windows containing w and denote it as u, based on the theorem, u can be mapped to v_w by a linear transformation that does not depend on w.
From Theorem 1, we can derive the Linearity Assertion as follows. Suppose a word w has two distinct senses s_1 and s_2. We can compute a word embedding v_w for w. Then, tag every occurrence of the word w with s_1 and s_2 depending on the sense used. Next, we can train an embedding that maps two different vectors to represent the two different senses.

Theorem 2. Assuming the model of Section 2.1, embeddings in the thought experiment above will satisfy $\|v_w - \hat{v}_w\|_2 \to 0$ as the corpus length tends to infinity, where $v_w \approx$

$\alpha v_{s_1} + \beta v_{s_2}$, and $\alpha = \frac{f_1}{f_1+f_2}$, $\beta = \frac{f_2}{f_1+f_2}$, where f_1 and f_2 are the numbers of occurrences of s_1 and s_2 in the corpus, respectively. The cited paper goes beyond mathematical explanation and supports it with empirical tests using word2vec and GloVe embeddings.

5.5 Testing the assertion

In the previous section, *Theorem 2* was introduced. In this section, the aim is to recreate the results using Hungarian ambiguous words and word embeddings extracted from huBERT. It is important to note that the original research's tests were conducted on English words. However, the theorem should hold regardless of the language. Another important difference is the use of huBERT instead of Word2Vec. BERT-based models, such as huBERT, have a complex structure without a clear objective function. Therefore, it is even more interesting to see whether the linear assumption holds in this context.
As written before, I obtained different vectors for 10 ambiguous Hungarian words. We can now test the linear assertion on these vectors. Let's consider the generic static vector w_m, and let the vectors for the different senses be annotated with s_1 and s_2.
In the experiment phase, the goal was to choose weights α and β, so that:

$$w_m(\hat{h}) = \alpha \cdot s_1 + \beta \cdot s_2$$

is the closest possible to w_m in Euclidean distance or most similar in cosine similarity (α and β are non-negative). Next, we set α and β as the word frequencies for the different senses, as stated in Theorem 2. To determine the frequency of each sense, I used English word frequency data from [25], since all ambiguous words used have distinct English translations, the frequencies of these translations were set as the frequencies for the different senses.
To decide if the Linearity Assertion holds, we can take the ratio of the coefficients obtained in the two tests. According to the assumption, these two should be nearly equal. Firstly, I ran the tests on the full, 768-dimensional embeddings. The next table shows the obtained coefficients (*coeff1* and *coeff2*) and their ratio (*ratio1*) for three sample words as well as the ratio of the frequencies of the English translations (*ratio2*).

Word	coeff1	coeff2	ratio1	ratio2
nyelv	0.10	0.05	2	5.38
dob	0.09	0.06	1.5	0.14
nyúl	0.11	0.04	2.75	0.11

Table 5.2: 768-dimensional embeddings with Euclidian distance

The next table shows the same information for the same three sample words as before, optimized for cosine similarity.

Word	coeff1	coeff2	ratio1	ratio2
nyelv	3.90	2.04	1.91	5.38
dob	0.85	0.65	1.31	0.14
nyúl	0.99	0.39	2.54	0.11

Table 5.3: 768-dimensional embeddings with cosine similarity

The average cosine similarity was 0.36, which combined with the fact that the ratios were not even close, indicates the necessity for PCA. If we look back at the analogy tasks, the performance peak was at 300 dimensions, but even at 150 dimensions, the embedding performed slightly better than at 768 dimensions. So I ran the same tests on 300 and 150 dimensions too, to see if things change. In average cosine similarity the two cases were identical rounding the numbers to two decimals and the ratios of the obtained coefficients were also consistent, so I don't present both cases, only the results obtained using the 150-dimensional embeddings. The next table shows how the embeddings behaved using Euclidean distance.

Word	coeff1	coeff2	ratio1	ratio2
nyelv	0.19	0.14	1.36	5.38
dob	0.14	0.16	0.88	0.14
nyúl	0.12	0.24	0.5	0.11

Table 5.4: 150-dimensional embeddings with Euclidian distance

Now, the same thing with cosine similarity:

Word	coeff1	coeff2	ratio1	ratio2
nyelv	0.04	0.03	1.33	5.38
dob	3.61	4.28	0.84	0.14
nyúl	2.09	4.30	0.49	0.11

Table 5.5: 150-dimensional embeddings with cosine similarity

The average cosine similarity in this case was 0.77, which is much more promising than the previously measured 0.36. Also we can see that after the PCA, the ratios correctly show which is the more frequent sense, but the values are still not close. These test results suggest that the linearity assertion doesn't hold when dealing with huBERT. Although, it could be investigated further with more ambiguous words, more example sentences, therefore more accurate embeddings.

Conclusion

When investigating analogy tasks, Word2Vec proved to be competitive with huBERT, although huBERT performed slightly better. In order to extract word vectors from huBERT, the most effective approach was to perform mean pooling on all the layers and subsequently applying mean pooling on these vectors. To reduce the dimensionality of the word vectors, Principal Component Analysis was employed, which proved to be a highly efficient tool for retaining as much information as possible while reducing the dimensions.

During the analysis, it was discovered that as the dimensionality was reduced, the scores for the analogy tasks improved, reaching their peak at 300 dimensions, after which they started decreasing. However, even at reduced dimensions of 150-200, the results remained slightly superior to the original ones.

Experiments were conducted to test the Linearity Assertion mentioned in *Chapter 5.4*, but the results did not support this assertion. One possible explanation for this is the use of a Transformer-based model like huBERT, as opposed to Word2Vec or GloVe, since in Transformers there is no fixed context because of the use of attention. for which the Linearity Assertion was empirically proven in the original paper. Additionally, examining the word frequencies associated with different senses could provide further insights, which would require a Hungarian corpus where the different senses are appropriately tagged. An area for further research could involve creating a self-made multi-sense embedding, which would enable precise control over word sense frequencies using a known corpus.

The codes necessary to replicate the results in this thesis are available on GitHub [26].

Bibliography

[1] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 1990.

[2] Subhasis Dasgrupta. Latent Semantic Analysis and its Uses in Natural Language Processing. `https://www.analyticsvidhya.com/blog/2021/09/latent-semantic-analysis-and-its-uses-in-natural-language-processing/`, 2021. [Online; accessed 31-May-2023].

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

[4] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[5] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (*EMNLP*). Association for Computational Linguistics, 2014.

[6] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors, 2018.

[7] Philip Gage. A new algorithm for data compression. *C Users J.*, 1994.

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

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1* (*Long and Short Papers*). Association for Computational Linguistics, 2019.

[10] Dávid Márk Nemeskey. Introducing huBERT. In *XVII. Magyar Számítógépes Nyelvészeti Konferencia* (*MSZNY 2021*), pages 3–14, Szeged, 2021.

[11] Davide Liu. Word similarity and analogy with Skip-Gram. https://davideliu.com/2020/03/16/word-similarity-and-analogy-with-skip-gram/, 2020. [Online; accessed 31-May-2023].

[12] Márton Makrai. Comparison of distributed language models on medium-resourced languages. In Attila Tanács, Viktor Varga, and Veronika Vincze, editors, *XI. Magyar Számítógépes Nyelvészeti Konferencia* (*MSZNY 2015*), 2015.

[13] Efnilex project. http://corpus.nytud.hu/efnilex-vect/.

[14] Hungarian webcorpus. http://mokk.bme.hu/resources/webcorpus/.

[15] Hungarian national corpus. http://corpus.nytud.hu/mnsz/index_eng.html.

[16] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *Philosophical Magazine Series 1*, 1901.

[17] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 1933.

[18] JU. D. Apresjan. Regular polysemy. 1974.

[19] Ladislav Zgusta. *Manual of lexicography*. De Gruyter Mouton, 1971.

[20] John Lyons. *Semantics*, volume 2. Cambridge University Press, 1977.

[21] Gábor Borbély, Márton Makrai, Dávid Márk Nemeskey, and András Kornai. Evaluating multi-sense embeddings for semantic resolution monolingually and in word translation. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Association for Computational Linguistics, 2016.

[22] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. Linear algebraic structure of word senses, with applications to polysemy, 2018.

[23] Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th International Conference on Machine Learning*. Association for Computing Machinery, 2007.

[24] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to PMI-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4, 2016.

[25] English corpora. https://www.english-corpora.org/coca/.

[26] Github repo. https://github.com/gedeonmate/Thesis.