POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Faculty of Computer Science

Chair of Intelligent Systems and Data Science

Data Science

Mateusz Góra
Register No. s26547

Spoken Polish Language Understanding

Master's thesis supervised by:

Grzegorz Wójcik, PhD

Advisor:

Dominika Wnuk, MSc

Warsaw, June, 2023

POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Systemów Inteligentnych i Data Science

Data Science

Mateusz Góra
Nr albumu s26547

Rozumienie Polskiego Języka Mówionego

Praca magisterska pisana pod kierunkiem:

dr hab. Grzegorz Wójcik

Promotor techniczny:

mgr Dominika Wnuk

Warszawa, czerwiec, 2023

Abstract

Title: Spoken Polish Language Understanding

The thesis explores the topic of natural language understanding in Polish language in the context of virtual assistants. In the work the methods of solving intent recognition and slot filling task are explored, as well as the current status of Polish natural language processing. In the research several models based on BERT architecture are compared on two representative datasets: MASSIVE and Leyzer.

Key words: Natural Language Processing, Natural Language Understanding, intent recognition, slot filling, BERT

Streszczenie

Tytuł: Rozumienie Polskiego Języka mówionego

Praca dotyczy dziedziny rozumienia języka naturalnego w języku polskim w kontekście wirtualnych asystentów. W pracy omówiono metody rozwiązywania zadania rozpoznawania intencji oraz wypełniania slotów, a także obecny stan przetwarzania polskiego języka naturalnego. W badaniu porównano modele oparte na architekturze BERT na dwóch reprezentatwynych zbiorach danych: MASSIVE oraz Leyzer

Słowa kluczowe: przetwarzanie języka naturalnego, rozumienie języka naturalnego, rozpoznawanie intencji, wypełnianie slotów, BERT

Contents

Chapter 1

Introduction

1.1 Research topic

In today's world artificial intelligence (AI) is ubiquitous. Machine learning solutions meant to make our life easier are becoming more and more popular. One of its fastest growing domains in recent years is Natural Language Processing (NLP). ChatGPT (OpenAI, 2022) is the latest biggest advance in that field. The chatbot earned its fame due to vast capabilities it offers: from holding a conversation to summarising texts and even solving programming problems. An another type of conversational AI are virtual assistants (VAs). Their key purpose is to perform tasks queried by the user, mainly with voice commands (Hoy, 2018). Among the most popular solutions available on the market are Apple's Siri, Amazon's Alexa, or Samsung's Bixby. However, none of these products currently support Polish language.

One of the main components of such tools relates to Natural Language Understanding (NLU). The two key NLU tasks it has to perform are intent recognition and slot filling, also known as Spoken Language Understanding (SLU) tasks. This topic is well known and researched for many years (Tur and De Mori, 2011)(Weld et al., 2022). Today, solutions based on language models, such as BERT (Devlin et al., 2018), give state-of-the-art results (Chen, Zhuo, and Wang, 2019). Polish language models were also built based on these advances (Rybak et al., 2020). However, to the best of my knowledge, there is no research about solving the SLU tasks purely in Polish language. The only evaluation of this problem was performed in a multilingual setting.

1.2 Purpose and scope of the thesis

The purpose of present work is to investigate and compare how well language models based on BERT architecture, that are pre-trained with Polish corpora, can deal with the problem of joint intent recognition and slot filling.

The thesis is composed of 7 chapters and its layout presents as follows:

- In Chapter 2 I will introduce the methodology applied to virtual assistants. However, before I discuss VAs I am first going to introduce the topic of natural language processing and understanding, and then describe what a virtual assistant is and define the NLU tasks: intent recognition and slot filling, as well as a joint SLU task. In the second section the methods used over the years of solving this problem are described. Lastly, I am going to concentrate on the Polish language and its processing.

- In Chapter 3 I am going to introduce the two datasets used for evaluating intent recognition and slot filling in Polish language: MASSIVE and Leyzer.

- The description of the experiment scenario is in Chapter 4. In my research I am going to compare 8 BERT-based models for each dataset on predefined metrics.

- In Chapter 5 the results of the research are presented.

- Chapter 6 contains the conclusion of the thesis and summarises the effects of the research.

- Lastly, in chapter 7 I discuss ideas for future work on this and other related topics.

Chapter 2

Methodology

2.1 Natural Language Processing

Natural language, by definition provided by the Cambridge Dictionary, is "a language that has developed in the usual way as a method of communicating between people, rather than language that has been created, for example for computers". In other words, it is a system of communication that enables humans to express their thoughts, feelings, and ideas to each other using words, grammar, and syntax.

Natural languages are characterized by their flexibility, ambiguity, and complexity, and are constantly evolving, growing and changing over time. Natural Language Processing is a set of computational techniques that are used for learning, analyzing, representing and generating natural language in a form of text or speech. These techniques include machine learning, deep learning, and statistical analysis. The goal of this area of research is to achieve human-like language processing in order to perform useful tasks. (Liddy, 2001)(Chowdhury, 2003)(Hirschberg and Manning, 2015). NLP has already many real-world applications, such as language translation, social media analysis, automated text summarisation, and aforementioned chatbots and voice assistants,.

2.1.1 Natural Language Understanding

Natural Language Understanding is a subfield of Natural Language Processing that focuses on the capability of a machine to interpret and comprehend the meaning and context behind text or speech, rather than just processing the language itself. (Liddy, 2001) believes that the synchronous model of language encapsulates the task of understanding language best. The model is constructed of different "levels of language", which are not processed in any particular order. This classification distinguishes:

(a) Phonology

Phonology is responsible for interpreting speech in terms of the arrangement of sound (Khurana et al., 2023). Crystal (2005) writes that "phonology studies the way in which a language's speakers systematically use a selection of these (speech) sounds in order to express meaning". In NLP the spoken input is first converted into a digitalized signal and further passed into a rule-based or a machine learning model for interpretation (Liddy, 2001).

(b) Morphology

Morphology is a field of study, which analyzes the structure of words. Each word is composed out of morphemes, the smallest units of meaning (Liddy, 2001). For example the Polish word *odczasownikowy*, which means *derived from a verb* can be dissected into a prefix *od-*, the root *czasownik* and the suffix *-owy*. While there are morphemes that keep their interpretation regardless of the word, like the English suffix *-ed*, which indicates the past tense of a verb, others may have a different meaning depending on the rest of morphemes. Continuing the analysis of the previous example, the prefix *od-* has the meaning of the preposition *from*. However, it can also be used in conjunction with other words where it can convey: an antonym (e.g. odbarwić), an amplification (e.g. odkarmić), a separation (e.g. odbiec), recreating something from a pattern (e.g. odrysować), a repetition in response to an action (e.g. odpowiedzieć), a recreation (e.g. odbudować), a decrease or removal (e.g. odczulić) or restoration of a characteristic (e.g. odmłodzić) (PWN, 2023). An NLU system should be able to interpret these morphemes accordingly to retrieve the meaning of each word.

(c) Lexical

Lexical understanding refers to the interpretation of individual words. The first type of processing at this level is part-of-speech tagging, where, given the context of the sentence, words are defined with the most feasible part-of-speech tag. Secondly, words with only one meaning can be described with it's semantic representation (Liddy, 2001). In Natural Language Processing often a lexicon is used - a lexical database with a computerized database of words, their meanings, and other relevant information, such as their pronunciation, part of speech, and usage examples. The most popular lexical databases are WordNet and

the Oxford English Dictionary. It is used for a range of text preprocessing techniques e.g. stemming or lemmatization (Khurana et al., 2023).

(d) Syntactic

Syntactic analysis involves identifying the grammatical relationships between the words in a sentence, such as subject-verb agreement, noun phrases or verb phrases. Also known as parsing, it outputs the structure of the sentence (Khurana et al., 2023). Such processing can be performed with a rule-based parser or, recently more popular, with deep learning models, where syntactic structures of language are learned automatically using large amounts of data.

(e) Semantic

This level of processing focuses on extracting the meaning of a text beyond its literal definition by identifying the relationships between the words and phrases used in the text. Analogically to how syntactic processing helps with determining the part-of-speech of an individual word, semantic analysis helps with understanding the context and context-dependent meaning of words (Liddy, 2001). For example, the word *country* can have different meanings depending on the context in which it is used. It can refer to a music genre, a nation or a land that is used for farming or left in its natural condition. There are many methods which enable such disambiguation, e.g. the frequency with which given interpretation is found in the whole corpus or statistical methods that capture the meaning of words based on their co-occurrence with other words.

(f) Discourse

The discourse level is responsible for analysing text longer than one sentence, as opposed to syntactic and semantic levels. It interprets the connections between sentences by focusing on features of the whole document or statement that portray its meaning (Liddy, 2001). One of the most frequent types of such processing is anaphora resolution, where e.g. pronouns are substituted with the word they are referring to. Other important techniques include discourse/text structure recognition coreference resolution (Khurana et al., 2023). This level of analysis is especially useful for tasks such as text summarization or information retrieval.

(g) Pragmatic

At this level the knowledge and context from the outside of a given text is used to gather the understanding of it. The goal is to achieve the implicit meaning that is not written (Liddy, 2001). It involves understanding the speaker's intention, the listener's interpretation, and the relationship between the two. In applications like conversational agents, such as chatbots, pragmatic analysis is used to understand and respond to user queries effectively.

Each level can have an effect of how we absorb information and the more levels are incorporated in an NLU system, the more comprehensive it is. Current solutions, such as BERT (Devlin et al., 2018), are already capable of syntactic and semantic analysis (Jawahar, Sagot, and Seddah, 2019). Moreover, the recently published products like ChatGPT, based on GPT model (OpenAI, 2023), are capable of performing tasks which do require discourse (e.g. text summarization) or pragmatic (e.g. recognizing sarcasm or irony) analysis, although their performance is still limited.

2.1.2 Virtual assistants

Virtual assistant (VA) is a dialog system, which purpose is to help the user with every day tasks. Firstly put forward in the early 60s, they now enable users to e.g. play music or make recommendations and are embedded into devices like smartphones, TVs or fridges (Pardela et al., 2022).

The usual pipeline of such system consists of speech recognition, natural language understanding, dialog manager, task manager, natural language generation and text-to-speech synthesis modules. First, user's spoken utterance is analyzed and converted from speech to text. The text is then processed by an NLU engine, where intent recognition and slot filling are performed. The output from the NLU component is then passed to the dialog manager, which communicates with a task manager. The task manager performs the requested task based on the NLU output and gives the response back to the dialog manager. After that the dialog manager passes it to the Natural Language Generation (NLG) component, where a natural language response is generated. This response is then converted from text to speech by text-to-speech synthesis module. For example, when user says "Play an Elton John song on Spotify", assuming the speech recognition module understood it correctly, the utterance should be recognized by the NLU engine with an intent with an id like "PlayMusicOnSpotify" with a slot value "artist: Elton John". The dialog manager passes it as a valid input to task manager, which should open the Spotify application and start playing a requested song. If it completes this task, the confirmation is send

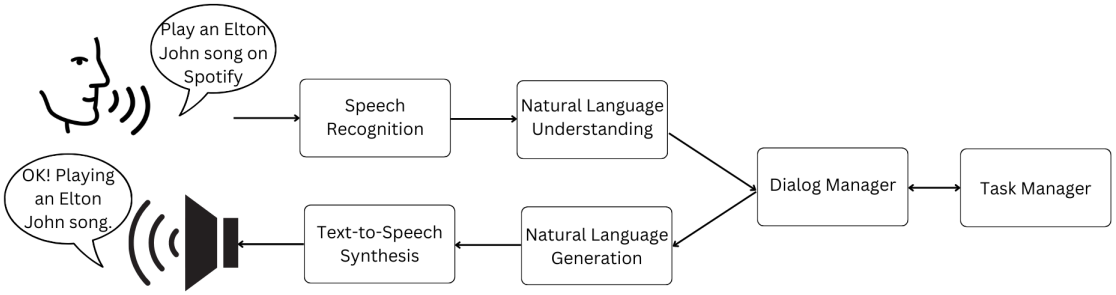


Figure 2.1: Virtual assistant pipeline (own elaboration)

back to the dialog manager and the NLG unit can respond with a dialog such as "OK! Playing an Elton John song.", which is then synthesized into speech and played from the device.

2.1.3 Intent recognition

Intent recognition, also called detection or classification, is one of the most critical problems to solve in dialog systems, especially virtual assistants (Liu, Li, and Lin, 2019). The is to find the underlying intention or aim behind a user's natural language input, usually expressed as a verb and noun combination, for example "play a song" or "set an alarm". The task itself is usually defined as a multiclass classification problem with multiple semantic intent classes (Asli et al., 2011). In the context of virtual assistants it is the process of understanding what the user is willing to achieve or is asking the machine to perform in a general sense.

Intent detection can be very challenging to solve due to the characteristics of natural language (Khan and Meenai, 2021). For example, users often use colloquialisms such as "play some rock". The statement refers to playing rock music, but the lack of music related words like "song" or "tune" might cause problems for an NLU system to correctly predict the goal. Another significant challenge is utterance ambiguity. It may come from expressions being too short or they don't specify the exact context, e.g. in an utterance "order a ticket" the word "ticket" can refer to ticket for a

movie, a train etc (Liu, Li, and Lin, 2019). In a conversational system though, this problem can be solved by prompting the user to specify the domain or utilising the dialogue context. Also the lack of labeled or imbalanced data brings a great struggle for researches as labeling the data usually comes at a great cost. Models trained on such corpus are prone to over-training. The same intent can be expressed in multiple different ways, which may not be accounted for in a small dataset. There may also occur conflicts between intents coming from usage of the same words, such as the word "play", which can be a part of an utterance with intent for playing music or playing videos (Weld et al., 2022).

The history of solving the intent recognition problem started with a rule-based approach and algorithms based on statistical features. Later solutions based on traditional machine learning algorithms were proposed, which utilised e.g. Naive Bayes or Support Vector Machines (SVMs). However, all of these solutions were insufficient when it came to actually understanding the semantic meaning of natural text input. Nowadays modern solutions revolve around deep learning methods such as word embeddings, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Language Models (LMs) and transfer learning (Liu, Li, and Lin, 2019)(Weld et al., 2022).

2.1.4 Slot filling

Slot filling is the second critical task performed by the NLU component in a dialogue system. While intent classification refers to classifying the meaning behind the whole utterance, slot filling focuses on tagging each word in a sentence. Specifically, it can be considered as a sequence labeling task, where the input word sequence $x = (x_1, x_2, ..., x_T)$ is mapped to output slot label sequence $y^S = (y_1^S, y_2^2, ..., y_T^S)$ (Goo et al., 2018a). A slot can be interpreted as a parameter or an argument of a query (Huang, Chen, and Bigham, 2017), while the slot label is the type of semantic information held by a word (Weld et al., 2022). A slot value might be represented by more than one word, or to be more precise, more than one token, where a token is the smallest entity analyzed by a Natural Language Understanding system. Slot label sequences can be encoded with the Inside/Outside representation, such as the IOB2 (BIO) format, which I will be using in my paper. The format consists of three types of tags: **B** tag marks a beginning of a chunk, **I** a token inside of a chunk that is not the first token, and **O** stands for a token outside of a chunk (Krishnan and Ganapathy, 2005). For example, a sentence *Show me the restaurants in New York City* could be labelled as *O O O B-place O B-location I-location I-location.*

The slot filling task is a well researched topic with many known issues that may be encountered. As a sequence to sequence task, it adapts similar problems such as label dependency and long distance dependency (Weld et al., 2022). Label dependency refers to the fact that some slots have a higher probability of occurring in the same sentence than others. Similarly some words, which are not part of the slot, may indicate that a certain slot value should be present before or after it. This information would ideally be later utilized to achieve better performance of the system. Long range dependency is associated with the fact that gradients tend to vanish or explode when propagated over multiple stages, which may result in lesser significance for the long-term interactions between the words in longer utterances. The diminishing and exploding gradients is a problem usually concerning very deep neural networks or recurrent neural networks and may also be the reason for the weights of neurons not changing during back propagation or the network being unstable respectively. Other problems generally come from the characteristics of the data. Slot filling task can have similar problems to intent recognition if the dataset is too small or imbalanced. Additionally the model may have problems with recognising slots that are not present in the training set.

Originally, with the task being similar to the Named Entity Recognition (NER) task, approaches like Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs) were utilised (Krishnan and Ganapathy, 2005)(Weld et al., 2022). Later, with the emergence of neural networks, RNNs have been a popular way to handle the task, although often with an addition of CRFs. Nowadays, alike intent classification, attention-based models, LMs and transfer learning are the state-of-the-art solutions.

2.1.5 Joint intent recognition and slot filling

Originally both sub-tasks were tackled separately. However, more recent solutions approach the issue as a multi-task learning problem. In multi-task learning information that comes from a related task is utilised to achieve better generalization, by sharing the information encapsulated in the training signals of corresponding tasks (Ruder, 2017). In this case, the sentence-level intent recognition and the word-level slot filling can be processed concurrently, which results in a synergistic effect. On a higher level, the motivation to combine the two may, for example, come from the idea that word-level signals might entail hints about the utterance-level intent (Zhang et al., 2018a). For instance, "Elton John" recognised as an *artist* indicates that the intent is more likely to be *PlayMusic* rather than *SetAlarm*. In general, techniques vary between implicit and explicit learning, getting the information about the intent

from slot filling and the slot labels from intent recognition, as well as explicit training of the joint distribution. While concatenating the tasks should promise better results, the joint task also has to find an answer to the same problems and limitations that both sub-tasks have (Weld et al., 2022).

2.2 Solving intent recognition and slot filling

In this section methods used for solving the separate tasks as well as the joint task are explained. First, different types of text representation are explored. In the further subsections classic approaches to modelling, and then neural network-based solutions are explained.

2.2.1 Text representation

Natural Language Processing consists of mainly two steps. The first one is focusing on outputting the textual raw data into numerical format that is understandable by computers. The second one is the modelling phase, where the designed approaches process the numerical data in order to get a desired result (Patil et al., 2023). Below I am going to give a brief summary of selected text representation methods, which can be used to tackle the problems described in present work.

2.2.1.1 One Hot Encoding

In this approach, a dictionary of unique words in a corpus is first created. The unique words are then sorted and indexed. Given a sentence, a matrix is constructed, where each row corresponds to a word in a sentence and each column to a unique word found in the corpus, thus, the number of rows is the number of words in a sentence, and the number of columns is the number of words in a vocabulary. If the value in a given row and column index is the same, a value of '1' is assigned. Otherwise, the value is set to '0'. For example, given a sentence "the quick brown fox jumps over the lazy dog", the vocabulary set created from it would be ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'lazy', 'dog']. The resulting one-hot encoding matrix is depicted in Table 2.1.

This simple representation ensures that the information about a sentence is not lost and the ordering of the words is maintained. However, the vectors do not carry any contextual information and the relationship between the words and their meaning is not captured. Moreover, such method creates a very sparse matrix which, with a bigger vocabulary, would also have many dimensions (Patil et al., 2023).

	the	quick	brown	fox	jumps	over	lazy	dog
the	1	0	0	0	0	0	0	0
quick	0	1	0	0	0	0	0	0
brown	0	0	1	0	0	0	0	0
fox	0	0	0	1	0	0	0	0
jumps	0	0	0	0	1	0	0	0
over	0	0	0	0	0	1	0	0
the	1	0	0	0	0	0	0	0
lazy	0	0	0	0	0	0	1	0
dog	0	0	0	0	0	0	0	1

Table 2.1: One Hot Encoding example (own elaboration)

2.2.1.2 Bag of Words

Here, similarly to One Hot Encoding, a vocabulary set is created and each column corresponds to a unique word found in the corpus. However, each row represents a whole sentence instead of one word. The values in each cell denote the number of occurrences of each word in a sentence. This approach drastically reduces the number of rows, however, at a cost of losing the word ordering information. While it allows for looking for similarity between sentences or documents, it makes it unusable for sequence labeling tasks. Nevertheless, it may be used for text classification tasks such as sentiment analysis or intent recognition (Patil et al., 2023).

2.2.1.3 TF-IDF

TF-IDF can be looked at as an advanced Bag of Words representation. One of the shortcomings of the previous method is that interpreting more frequent words as more important for sentence classification, is not viable when it comes to e.g. stop words. Again a matrix with unique words as columns and documents (sentences) as rows is created, but the value assigned to each cell as $TF(w,d) \cdot IDF(w)$. Here, $TF(w,d)$ stands for term frequency of word w in a document sentence d and $IDF(w)$ for inverted document frequency. IDF is obtained as $IDF(w) = \log(\frac{N}{df(w)}$, where N is the number of documents in a corpus, and $df(w)$ is the number of documents that contain the word w. That way the noise in the matrix is reduced, however, the ordering of the words is still not retained (Patil et al., 2023).

2.2.1.4 N-grams

To capture the order of words in a sentence N-grams can be used. N denotes the number of words in a group. A vocabulary set created that way is a set with unique

n-sized chunks. For example, in a 2-gram representation, "not like" can encapsulate the correct sentiment of a sentence. Such information would not be explicitly shown in the previous two approaches. This method is a generalisation to bag of words, which can be looked at as 1-gram representation. Unfortunately, it still is not a viable option when it comes to sequence labelling. Also it does not tackle problem with less and very frequent vocabs not conveying much information. Moreover, the dimensionality of the feature space increases exponentially with the value of N (Patil et al., 2023).

2.2.1.5 Word Embeddings

Word embeddings are vector representations of words that can capture the context of a word and its similarity to other words. They are most commonly computed from a large corpus of unlabeled data using neural network models and unsupervised training. The models automatically capture the semantic and syntactic relationships between words in an implicit manner, which means no further feature extraction is needed. There are two main classes of embeddings: static and dynamic. The difference between the two is the fact, that in the latter ones, a word has a different representation based on the context of a sentence. This change can further improve the semantic language understanding. Most commonly known static embeddings are Word2Vec (Mikolov et al., 2013), GloVe (Pennington, Socher, and Manning, 2014), and FastText (Bojanowski et al., 2016). The more complex, dynamic models include ELMo (Peters et al., 2018), GPT(Radford et al., 2018), and BERT (Devlin et al., 2018), which is the leading solution for NLU tasks and I am going to describe it in greater detail in a later section of present work (Patil et al., 2023).

2.2.2 Classic methods

Similar problems to intent recognition and slot filling have been researched for many years. It is possible to apply these classic approaches to solve the two problems. In this subsection I am going to outline the main solutions that have been applied over the years.

2.2.2.1 Rule-based methods

While the first NLU systems were based on rule-based template semantic recognition (such as Dowding et al. (1993)), in this work I will focus on machine learning methods. However, it is worth noting that hybrid approaches that utilise both human driven and

machine learning models can be beneficial and are utilized to enhance the performance of the system e.g. by Rasa (2023).

2.2.2.2 Classic machine learning methods

Classic machine learning methods were mostly employed for separate processing of these tasks. As intent recognition is usually solved as a text classification task, it has a long history of solutions. Below I present just a few chosen approaches to summarise how the objective can be achieved using traditional techniques.

- Naive Bayes

  Naive Bayes is a simple classification model based on the Bayes theorem (Joyce, 2021) and "naive Bayes assumption", which assumes that the fact a particular feature exists in a class is not related to other features being present as well. Even though the assumption does not have a real-world application, empirical studies show that the method can achieve surprisingly good results, while greatly simplifying the training process. McCallum, Nigam, et al. (1998) present two Naive Bayes methods for text classification: one using multi-variate Bernoulli model and the other a uni-gram language model with integer word counts. Later Helmi Setyawan, Awangga, and Efendi (2018) describe an application of Multinomial Naive Bayes for intent classification, which utilises TF-IDF text vectorisation.

- AdaBoost

  Boosting is based on an idea that a series of "weak" classifiers (e.g. single-node decision trees) can together be combined into a "strong" classifier. At each iteration a weighted training set is shown to a "weak" classifier, with the weights being adjusted, so that a classifier focuses on distinguishing "harder" training examples (Freund and Schapire, 1997). For instance, with text represented as n-grams, each "weak" classifier can be used to check whether an n-gram exists in given text (Tur and De Mori, 2011). Schapire and Singer (2000) proposed an application of the multi-class classification algorithm AdaBoost.MH for the AT&T How May I Help You? system.

- Support Vector Machines (SVMs)

  SVM is a supervised learning method, that constructs an optimal hyperplane, in other words with maximum distance from the data points, in an N-dimensional

space which distinctly separates them. In the most simple 2-dimensional example the hyperplane is a straight line that divides the feature space in two (Gandhi, 2018). Similarly to Adaboost, SVMs were found to be a feasible solution for text classification due to large dimensions of the input space in such task (Tur and De Mori, 2011). For example, Haffner, Tur, and Wright (2003) introduce a multi-class call classifier based on SVMs. To adapt the algorithm to a multi-class problem they divided it into 1-vs-others heterogeneous binary classifiers, with 48 different classes in the dataset and word n-grams as features. Lekic and Liu (2019) outline more modern approach with Word2Vec and BERT embeddings and training a support vector machine intent classifier using them.

As mentioned in 2.1.4, traditionally the problem was tackled by Hidden Markov Machines and Conditional Random Fields:

- Hidden Markov Models (HMMs)

  Hidden Markov Model is an extension of a Markov model that incorporates a situation in which the modelled event can only be observed through another set of stochastic processes which output the sequence of observations (Rabiner, 1989). Weischedel et al. (1993) propose an implementation of HMM for part-of-speech tagging task. Based on the success of this solution, Bikel, Schwartz, and Weischedel (1999) describe a Named Entity Recognition system that uses a Hidden Markov Model. In their application the HMM tags each word with one of the predefined classes (e.g. person or organization) or NOT-A-NAME class (similar to O in BIO notation). The idea revolves around creating an independent language model for every label based on bigrams with an addition of a next label prediction based on the previous word and associated class to it. However, it was argued that HMMs have its limitations and to handle some of them, e.g. inability to handle non-independent or hard to specify observation features, conditional models should be used such as Maximum Entropy (Borthwick, 1999), Maximum Entropy Markov Models (McCallum, Freitag, and Pereira, 2000) or Conditional Random Fields (Lafferty, McCallum, and Pereira, 2001).

- Conditional Random Fields (CRFs)

  Conditional Random Fields is a machine learning method created specifically to tackle sequence labeling problem (Lafferty, McCallum, and Pereira, 2001). CRF's basic idea revolves around creating a conditional probability distribution over tag sequences in an observation. Comparing it to Hidden Markov Models,

Conditional Random Fields do not require an assumption of independence to achieve tractability and eliminate the label bias problem. CRFs model the dependencies between adjacent labels in the sequence, leading to more accurate and robust predictions. They incorporate all the benefits of Maximum Entropy Markov Models, while only having one exponential model. Their usage and advantages is widely described in research by e.g. (Li and Mccallum, 2003) or (Chopra et al., 2017).

A basic CRF layer, called linear-chain CRFs, can be used for NLP tasks such as named entity recognition or part of speech tagging (Sutton, McCallum, et al., 2012). The model assumes that an output label y_t is dependent on both the input sequence X and the previous tag y_{t-1} (see Figure 2.2). Formally, the conditional probability for an output label sequence y can be defined as:

$$p_\lambda(y|X) = \frac{1}{Z(X)} \prod_{t=1}^{T} \phi_t(y_t, y_{t-1}, X), \tag{2.1}$$

where X is a sequence of input feature vectors $(x_1, ..., x_T)$, $Z(X)$ is an instance-specific normalization (partition) function, and ϕ_t is the local function (factor) at time t. ϕ_t can be further factorised as:

$$\phi_t(y_t, y_{t-1}, X) = \phi_t^1(y_t, X) \cdot \phi_t^2(y_t, y_{t-1}), \tag{2.2}$$

and each local function has the form:

$$\phi_t^1(y_t, X) = \exp\{\sum_k \lambda_k^1 f_k^1(y_t, x_t)\}, \tag{2.3}$$

$$\phi_t^2(y_t, y_{t-1}) = \exp\{\sum_k \lambda_k^2 f_k^2(y_t, y_{t-1})\}, \tag{2.4}$$

where λ^1, λ^2 are real-valued parameter vectors, $f_k^1(y_t, X)$ is a feature function which encodes a feature of the input, and $f_k^2(y_t, y_{t-1})$ is a feature function which encodes a feature of the transition at time t (Sutton, McCallum, et al., 2012)(Jeong and Lee, 2008).

Based on the described approaches researchers have also tried to find a way to utilise them in a joint setting. Jeong and Lee (2008) applied a triangular-chain CRF model, in which the classic CRF method was used. In this solution the intent is recognised based on both the input sequence and the annotated slots to improve the results of the latter task. Wang (2010) also discuss a two-pass solution, where first

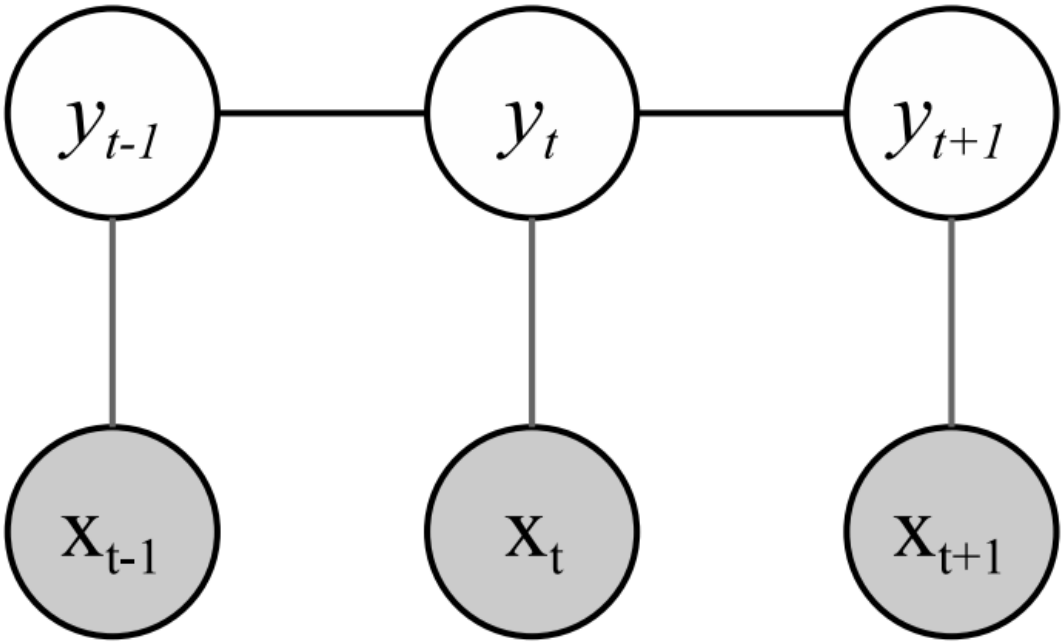


Figure 2.2: Linear-chain CRFs. (Jeong and Lee, 2008)

the intent could be classified with a Maximal Entropy model, and later a slot filling CRF model is trained so that the predicted label is used as a part of the conditional probability CRF maximizes. Celikyilmaz and Hakkani-Tur (2012) describe a method which utilises multi-layer HMM to improve the domain and intent recognition with slot predictions. Later, Lee, Ko, and Seo (2015) proposed a solution for virtual assistants based on CRFs, where they defined four tasks, three of them related to user's intent, which their system recognised simultaneously. The framework achieved higher performance than the triangular-chain CRF model.

2.2.3 Deep neural network methods

Deep neural networks (DNNs) have become a very common way of solving text classification and slot filling problems since their rise in popularity. Before I describe how they are used for these tasks, I am first going to introduce some basic concepts related to it, to better understand how neural networks (NNs) work.

2.2.3.1 Perceptron

Perceptron is a simple mathematical model of a neuron, first invented by Mcculloch and Pitts (1943) and implemented by Rosenblatt (1957). A single perceptron can be

used as a binary classificator or a regressor. It is also a building block of more complex models called artificial neural networks (Marsland, 2015). An artificial neuron consists of a number of basic components:

- n inputs $x_1, ..., x_n$
- n weights $w_1, ..., w_2$
- bias Θ
- output y

The output is defined as:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i \cdot x_i = W^T X \geq \Theta \\ 0 & \text{otherwise} \end{cases}, \tag{2.5}$$

where $W, X \in R^n$ stand for the weight and input vectors respectively. The output values can be interpreted as perceptron being activated for $y = 1$ ("positive" class) and not activated when $y = 0$ ("negative" class). This can be further used for training it in a supervised setting as a binary classifier. With the training set presented as a pair of input vector and the correct answer (0 or 1), the perceptron can "learn" by adjusting its weights using a simple method called "delta rule". The algorithm looks as follows:
At each iteration:

1. Feed an observation to the perceptron,
2. Compare the predicted value with the correct value.
    - If the values are the same, continue;
    - else adjust the weight vector:

$$W' = W + (d - y)\alpha X, \tag{2.6}$$

where d is the expected output, y is the actual output and α is an experimentally tuned parameter.

Bias Θ can also be incorporated in training by moving it to the left side of the equation 2.5 and by treating it as an additional weight with fixed input -1 (Marsland, 2015).

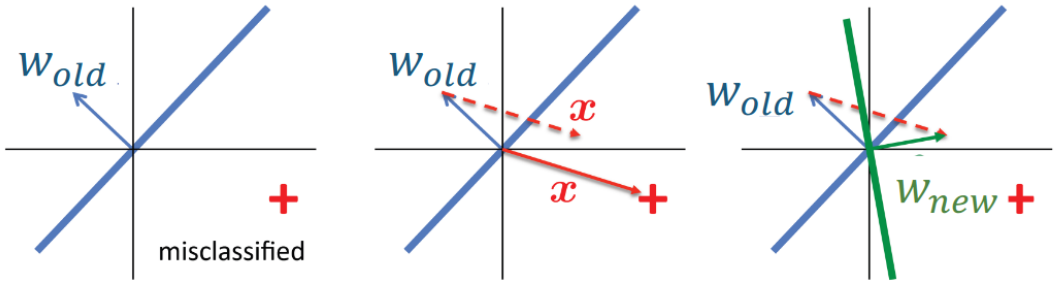


Figure 2.3: Geometric interpretation of the perceptron (Carpuat, 2017)

2.2.3.2 Geometric interpretation

In geometric interpretation we consider an n-dimensional input space, where each input vector X is represented as a point. Assuming that the decision boundary is a hyperplane defined by a normal vector W, the training can be interpreted as finding a hyperplane that distinguishes "positive" and "negative" observations. During each training step the hyperplane either stays the same or moves towards "positive" cases (when $d = 1$ and $y = 0$) and away from "negative" cases in the opposite scenario (see Figure 2.3). The role of the bias is to move the hyperplane from the origin of the input space, otherwise the hyperplane would pass through it (Sydow, 2019).

2.2.3.3 Multiclass classifier

A single perceptron is only capable of distinguishing 2 classes and even that is constrained by the classes being linearly separable in the feature space. To handle predicting more than 2 classes, a layer of neurons should be used, usually with them being fully connected with input (Sydow, 2019). The output of each neuron is then combined and the predicted class is determined based on it. There are two ways of presenting such input:

- local - there are as many perceptrons as there are classes: i-th perceptron active $\Rightarrow$ i-th class predicted
- global - the decision is inferred from the combined output. Usually there are less output neurons than in the local representation and each class is represented as a binary code.

For example, when there are three classes the local representation would look like:

1. $[1, 0, 0]$,

2. $[0, 1, 0]$,

3. $[0, 0, 1]$,

and a global representation could be encoded as:

1. $[1, 0]$,

2. $[0, 1]$,

3. $[0, 0]$.

2.2.3.4 Activation function

So far only the output of the perceptron with a discrete activation function has been considered in this work. The output of the activation of a perceptron $net = \sum_{i=1}^{n} w_i \cdot x_i - \Theta$ is usually passed as an argument to an activation function to prevent linearity (Sydow, 2019). Some of the most frequently used functions are:

- step function $y = \lfloor net > 0 \rceil$
- sign function $y = signum(net)$
- sigmoid function $y = \frac{1}{1+e^{-net}}$
- linear function $y = net$
- ReLU $y = max(0, net)$
- softmax $y_i = \frac{e^{net_i}}{\sum_{j=1}^{N} e^{net_j}}$. It is used in multiclass classification problem and can be interpreted as the probability of the input belonging to class i (where N is the total number of classes).

2.2.3.5 Gradient Method

Gradient Method is a technique used for calculating the weights of a neuron with a continuous activation function (Sydow, 2019). First, we define the error measure for a single neuron:

$$E = \frac{1}{2}(d - y))^2 = \frac{1}{2}(d - f(w^T x))^2 = \frac{1}{2}(d - f(net))^2, \tag{2.7}$$

where d is the desired continuous output, y is the actual continuous output, w is the weight vector, x is the input vector and f is the activation function. The goal is to minimise the error by properly adjusting the weight vector. The method is based

on the idea that the direction towards a minimum of the function is opposite to the gradient vector. The gradient of the error E is computed as follows:

$$\nabla E(w) = \frac{\partial E}{\partial w} = -(d - y)f'(net)x. \tag{2.8}$$

The new weights of a neuron are then calculated as:

$$w_{new} = w_{old} + \eta(d - y)f'(net)x, \tag{2.9}$$

where η is the learning rate coefficient.

This can now be extended to changing weights in a single-layer neural network (SLNN) (Sydow, 2019). With the layer having J inputs and K continuous neurons, the variables are defined as:

- input vector: $y^T = (y_1, ..., y_J,$
- output vector: $z^T = (z_1, ..., y_K,$
- desired output vector: $d^T = (d_1, ..., d_K,$
- weight matrix: $W = [w_{kj}]$, where w_{kj} is the weight of k-th neuron connected with j-th input,
- activation functions matrix: $\Gamma = diag[f(\cdot)]$ with $K \times K$ dimension,

where $z = \Gamma[Wy]$. The error E is then computed for a single input vector:

$$E = \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2 \tag{2.10}$$

Applying the gradient method we get a modification of a single weight:

$$\Delta w_{kj} = -\eta\frac{\partial E}{\partial w_{kj}}. \tag{2.11}$$

$\frac{\partial E}{\partial w_{kj}}$ is an equivalent of:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} \tag{2.12}$$

and:

$$\delta_{z_k} = -\frac{\partial E}{\partial net_k} = (d_k - z_k)f'(net_k), \tag{2.13}$$

$$\frac{\partial net_k}{\partial w_{kj}} = y_j, \tag{2.14}$$

thus:

$$W_{new} = W_{old} + \eta\delta_z y^T. \tag{2.15}$$

As a result a learning algorithm for a single-layer network is obtained:

1. Choose arbitrarily η and E_{max}, set weights in W randomly, $E = 0$
2. for each observation x in training set:
   - calculate the output vectors z
   - modify the weights:
   $$w_k \leftarrow w_k + \eta(d_k - z_k)f'(net_k)y$$
   - accumulate the error:
   $$E \leftarrow E + \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2$$
3. If $E < E_{max}$ then finish training, else $E = 0$ and repeat step 2.

2.2.3.6 Multi-layer network

Single-layer neural network is capable of splitting the input space into linearly separable zones. Adding next layers enables transforming the space even more. Therefore, a multi-layer neural network (MLNN) can, in theory, approximate any input space to output space transformation, at least to a certain extent (Marsland, 2015). Any layer that is not an output layer is called "hidden", as its desired output is not known. This limitation meant that the training algorithm had to be adjusted. The technique that enables it is referred to as the back-propagation method. Back-propagation is a specific instance of reverse-mode differentiation, first described in Linnainmaa (1970), and first implemented in the context of neural networks by Werbos (1981). It allows computing gradients in DNNs. Below I am going to present how it applies to two-layers NN, however, the algorithm extends to any MLNN.

Expanding the reasoning for Gradient Method, an additional hidden layer with J neurons and I inputs is added before the layer described previously. The variables are now denoted as:

- input vector: $x^T = (x_1, ..., x_I,$
- first layer's weight matrix: $V = [v_{ji}]$, where v_{ji} is the weight of j-th neuron connected with i-th input,
- first layer's output vector (second layer's input vector): $y^T = (y_1, ..., y_J,$
- second layer's weight matrix: $W = [w_{kj}]$, where w_{kj} is the weight of k-th neuron connected with j-th input,

- second layer's output vector: $z^T = (z_1, ..., y_K,$
- desired output vector: $d^T = (d_1, ..., d_K,$
- activation function operator: $\Gamma = diag[f(\cdot)]$ with $J \times J$ or $K \times K$ dimension,

where $z = \Gamma[Wy] = \Gamma[W\Gamma[Vx]]$. The idea of the method revolves around updating the weights of the neurons backwards, so starting from the last layer and moving towards the first one. We begin by modifying the weights of the first layer as described in the previous section. Then we apply the gradient method again, this time to update the weights of the hidden layer:

$$\Delta v_{ji} = -\eta \frac{\partial E}{\partial v_{ji}}. \tag{2.16}$$

Analogically:

$$V_{new} = V_{old} + \eta \delta_y x^T, \tag{2.17}$$

where $\delta_y^T = (\delta_{y_1}, ..., \delta_{y_J})$ is the error signal vector of the first layer and is calculated accordingly:

$$\delta_{y_j} = -\frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial net_j} = -\frac{\partial E}{\partial y_j} \cdot f'(net_j) = \sum_{k=1}^{K} \delta_{z_k} wkj \cdot f'(net_j). \tag{2.18}$$

The adjusted algorithm then goes as follows (Marsland, 2015):

1. Choose arbitrarily η and E_{max}, set weights in W and V randomly, $E = 0$
2. for each observation x in training set:
   - calculate the output vectors y and z
   - accumulate the error:
   $$E \leftarrow E + \frac{1}{2}\sum_{k=1}^{K}(d_k - z_k)^2$$
   - calculate the error signals:
   $$\delta_{z_k} = (d_k - z_k)f'(net_k),\ \delta_{y_j} = f'(net_j)\sum_{k=1}^{K}\delta_{z_k} w_{kj}$$
   - modify the weights (starting from the last layer):
   $$w_k \leftarrow w_k + \eta \delta_{z_k} y$$
   $$v_j \leftarrow w_j + \eta \delta_{y_j} x$$
3. If $E < E_{max}$ then finish training, else $E = 0$ and repeat step 2.

2.2.3.7 Regularization

One of the most important problems in machine learning is making sure that the model is able to generalise, i.e. to perform adequately not only on the training set, but on the testing set as well. As defined in Bengio, Goodfellow, and Courville (2017), "regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error". Below I am going to describe some popular methods of regularization:

- Early stopping

  High capacity models trained on big datasets are prone to over-training, which means that the validation set error at some point in time starts rising even though the training error keeps going down. Early stopping is simply restoring the weights of the model to the moment when the validation error was at the lowest. Usually the algorithm has a *patience* parameter, which decides on how much more time should the model be trained for after a rise in validation error is noticed. This method is one of the most commonly used regularization techniques due to its effectiveness and simplicity and is often used alongside other approaches. (Bengio, Goodfellow, and Courville, 2017)

- Bagging

  Bagging (bootstrap aggregating) is a method for decreasing generalization error by training and aggregating a number of models. The technique is based on an observation that averaging from multiple measurements lowers the variance. First, k different datasets are created from the original dataset by sampling with replacement. That way each dataset is of the same size, however, each is missing some records and has other duplicated. Then, k models are trained separately and the final output (on the testing set) is achieved by having them vote on it. Neural networks usually greatly benefit from this method, though it can be very expensive due to the necessity of training many models. (Bengio, Goodfellow, and Courville, 2017)

- Dropout

  Dropout might be perceived as a practical implementation of bagging for an exponential number of models. In this technique an ensemble made of all sub-networks created by removing neurons from the base network is trained. It can be applied by multiplying the weights of a randomly selected set of neurons by

zero. This method is less expensive than bagging, although it still has its price. As removing neurons from a network reduces it capacity, bigger networks are used to compensate for it. Usually finding the best solution also requires more iterations. Especially with larger datasets these costs may outweigh the gain achieved from regularization. (Bengio, Goodfellow, and Courville, 2017)

2.2.3.8 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks were created as a result of studies on the visual cortex - part of the brain that is responsible for image recognition. The research was conducted in the 1950s by David Hubel and Torsten Wiesel, where they discovered simple and complex cells (Yalçın, 2021). The first ones are responsible for recognising simple things, e.g. edges and bars, while the latter ones respond to more complex features. The more complex the cell is the more specific it becomes. Moreover, it was shown that the complex neurons are spatially invariant, thus able to detect a pattern independently of its location or orientation, which they achieve by pooling information from many simple cells. This lead to the creation of a first artificial convolutional neural network by Fukushima (1980), and later to a first application of a modern CNN by LeCun et al. (1989).

Convolutional Neural Networks are mostly composed of three types of layers: convolutional, pooling and fully-connected (O'Shea and Nash, 2015). Usually CNN is build with a number of convolutional (sometimes a few in a row) and pooling layers stacked alternately and one or more fully-connected layers at the end. Bellow I am going to explain in more detail how the two first ones, which are specific to these types of NNs, work:

- Convolutional layer

  In a convolutional layer neurons are organised into planes called feature (activation) maps. Each neuron in a feature map is connected only to a small, local regions of the input and has shared weights with all the other neurons in the map, thus recognising the same pattern, but in a different part of the input. Each pattern is implemented as a small kernel. The output of the neurons is achieved through the mathematical operation of convolution: the inputs from the region are multiplied by weights of the kernel and summed (as illustrated in Figure 2.4). The output is then passed to an activation function. This procedure can be looked at as passing the input through a filter. The neurons that are responsible for each receptive region will get activated if the input is

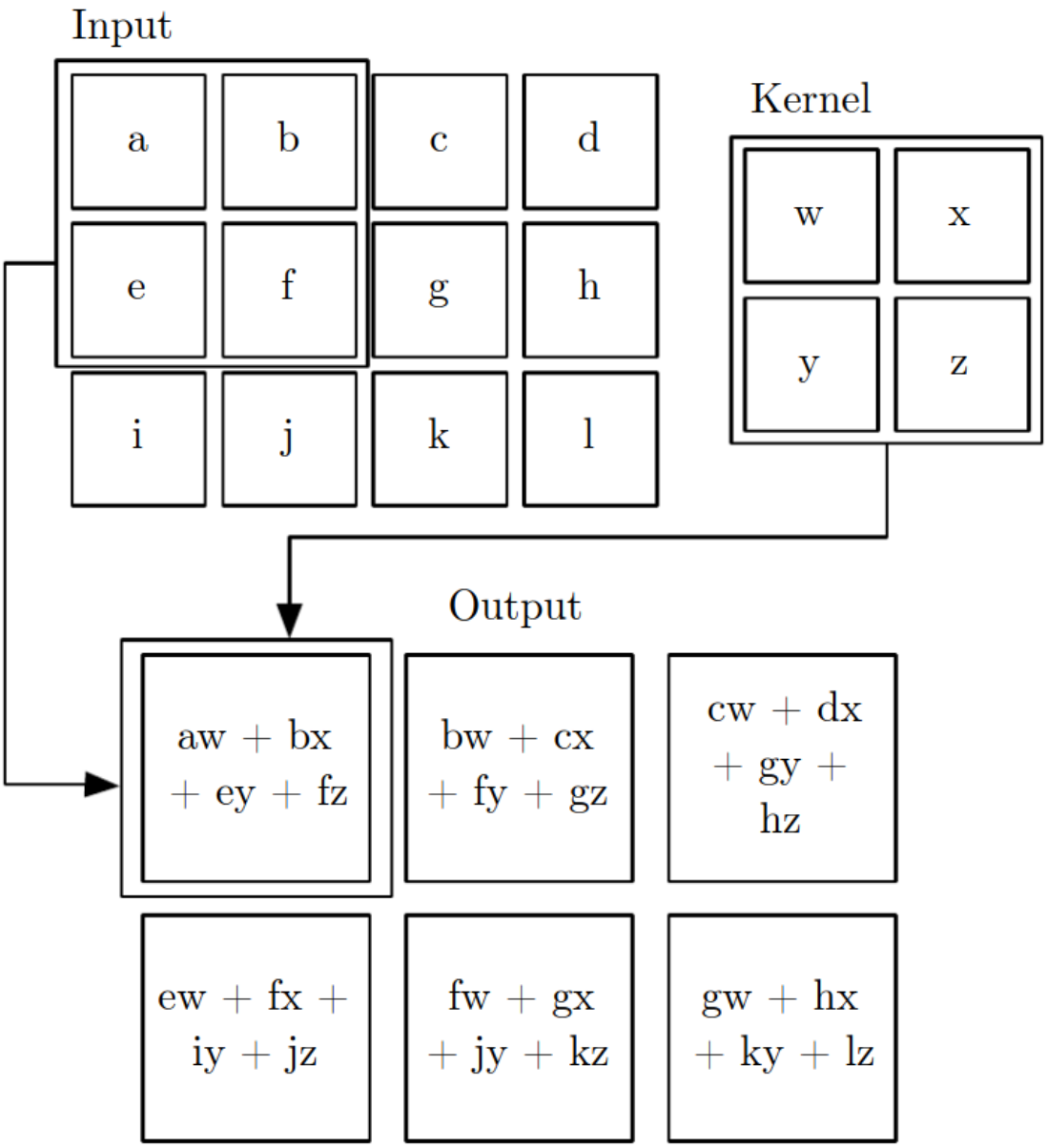


Figure 2.4: 2-D convolution operation example (Bengio, Goodfellow, and Courville, 2017)

similar to the kernel. At the end of a convolutional layer an activation map is produced. For example, in image processing, a kernel with a shape 1×2 and weights $[-1, 1]$ is used to detect vertically oriented edges in a picture (see Figure 2.5).

This approach has two main advantages. Firstly, it reduces the number of parameters the model has. To process an image with input shape $32 \times 32 \times 3$ (32 by 32 pixels in RGB representation) each neuron in a DNN would have 3,072 weights. With a kernel set to a size of 2×2 each neuron in a feature map would consist of $2 \times 2 \times 3 = 12$ weights. Secondly, the complexity of the model can be reduced by the optimisation of the output through the use of hyperparameters: receptive field size, stride and zero-padding. Continuing the previous example, if we set the stride, so the amount of movement over the input, to 2, we would get an activation map of size 15×15. Zero-padding is an another way to control the shape of the output. By surrounding the input with zeroes we can minimise the reduction of size caused by the kernel size and the stride.

- Pooling layer

  The second typical element of CNNs is the pooling layer. The input of a pooling layer is an activation map produced by a convolutional layer connected to it. Each neuron in the pooling layer is connected to a local region (usually) of one feature map. Typically, max pooling (computation of the maximal input from local region) or average pooling (computation of an average of the input from local region) is used. In general the stride is set so that the receptive fields do not overlap. This architecture ensures that the dimensionality is further reduced, which causes even further reduction of the number of parameters, and the output is insensitive to minimal shifts of the feature map pattern.

CNNs for intent recognition and slot filling

While CNNs were originally created for computer vision, they can also be used in NLP (Jacovi, Sar Shalom, and Goldberg, 2018). There, one-dimensional Convolutional Neural Networks are used, with the one dimension being time. Usually, each word (token) is represented as a vector (e.g. word embeddings). The convolutional kernel will then have a size of $n \times m$, where n is the number of tokens and m is the size of a vector. In NLP convolution operation can be looked at as window-based feature

Figure 2.5: Edge detection feature map achieved with $[-1, 1]$ kernel (Bengio, Goodfellow, and Courville, 2017)

extraction (Camacho, 2019). The features then are patterns in sequential n-grams, which can indicate e.g. an intent.

Convolutional Neural Networks were applied for intent recognition by Hashemi (2016). They used CNNs to automatically extract query features. The model was trained on text vectorised with *word2vec* (Mikolov et al., 2013). The query vectors were taken after a max pooling layer and passed on to a fully-connected classification layer. Xu and Sarikaya (2013) proposed a joint intent detection and slot filling model, where features are also extracted with CNNs. However, in their approach they used the TriCRF (Jeong and Lee, 2008) on top of the neural network. Not only did the model achieve better results than the traditional method, it also allowed to skip the manual feature extraction, but at the cost of the model becoming more complex.

2.2.3.9 Recurrent Neural Networks (RNNs)

Recurrent neural networks are one of the best solutions when it comes to dealing with analysing a sequence of objects or events (Bengio, Goodfellow, and Courville, 2017). Moreover, it can handle inputs and outputs of variable length, making it a great tool for NLP tasks. The main difference between aforementioned feed-forward neural networks (FFNN) and RNNs is that outputs of a neuron can be used as an input to the ones of lower layers or even to itself. Through the structure of hidden states, the recurrent neural networks are able to store, memorise and process past signals over long periods of time (Salehinejad et al., 2017). Below I am going to

briefly present basic concepts needed to understand how RNNs work and later talk about their implementation to the discussed problem.

A typical simple recurrent layer consists of:

- sequential input signal $x = \{..., x_{t-1}, x_t, x_{t+1}, ...\}$, where x_t is an input vector at a discrete time t
- hidden layer $s_t = (s_1, ..., s_M)$, where each of M units is connected recurrently to one another
- output signal o_t at a discrete time t
- weight matrices U, W, V for input-to-hidden, hidden-to-hidden and hidden-to-output connections, respectively

Its operations can be described mathematically as follows:

$$s_t = f(Ux_t + Ws_{t-1} + b_s) \tag{2.19}$$

$$o_t = g(Ws_t + b_o), \tag{2.20}$$

where b_s and b_o are bias vectors, and $f(\cdot)$ and $g(\cdot)$ are activation functions. As the input is sequential through time, the described steps are iteratively repeated over time t. The hidden state s_t is a function of all the past states and inputs, thus it can be interpreted as a part of the recurrent layer that represents its "memory".

Such layer can also be represented as a (potentially infinite) feed-forward neural network, where each layer corresponds to one iteration, has the same parameters and its hidden output is connected to the input of the subsequent layer. This representation is called unfolding RNN through time and can be seen in Figure 2.6. It enables a training technique named Back-propagation through time (BPTT), which is a generalisation of the training algorithm in FFNNs.

Recurrent layers can be connected to each other creating a deep recurrent neural network. Such network has a potential to become very complex upon unfolding, even more so when the length of the input sequence is big. Because of these aspects, RNNs are susceptible to the same problems as very deep neural networks, the main ones being overtraining, and vanishing and exploding gradients. The vanishing gradient problem is related to the exponential decrease of the gradients as they are propagated back, while in exploding gradient problem the gradients grow exponentialy fast to infinity. Both oh these problems make it hard for the RNN to learn long-term

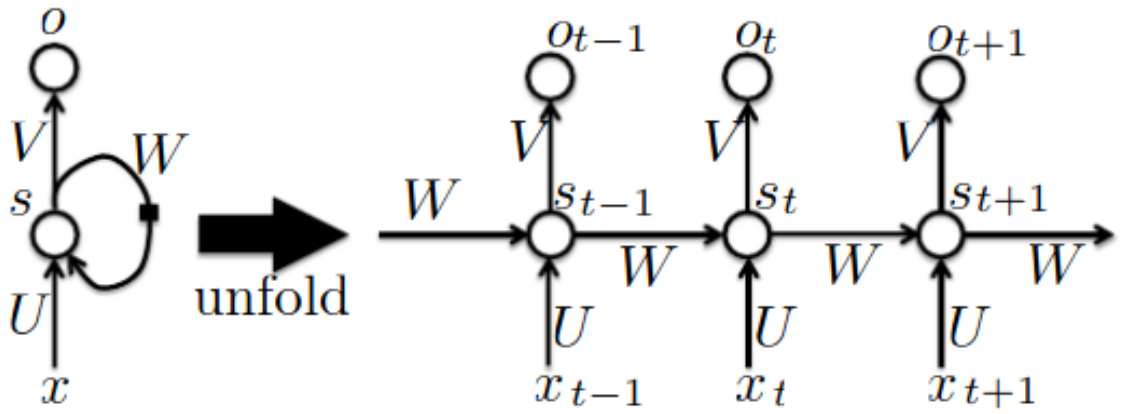


Figure 2.6: Unfolding simple recurrent layer through time (Bengio, Goodfellow, and Courville, 2017)

dependencies in the sequence (Bengio, Simard, and Frasconi, 1994). To tackle overtraining, dropout was found to be a good solution (Gal and Ghahramani, 2016). For vanishing/exploding gradient problems truncated BPTT can be used, where only a part of the sequence is used in training. Additionally, gradient clipping can be used to handle exploding gradients.

LSTM

However, even with the assumption that the recurrent network is stable, the long-term memory problem still arises, as the older inputs have lower significance then the newer ones, because of smaller weights given to long-term interactions (Bengio, Goodfellow, and Courville, 2017). To further deal with this problem a Long Short-Term Memory (LSTM) cell was created (Hochreiter and Schmidhuber, 1997). Comparing it to a simple neuron with memory it has bigger capacity, learns faster and can "memorise" longer dependencies in sequential data. A standard LSTM holds two separate states c (cell) and h (hidden), that are passed on between iterations. They are responsible for long-term and short-term memory, respectively. Mathematically, LSTM can be described as follows:

$$i_t = \sigma(W_{xi}^T x_t + W_{hi}^T h_{t-1} + b_i) \tag{2.21}$$

$$f_t = \sigma(W_{xf}^T x_t + W_{hf}^T h_{t-1} + b_f) \tag{2.22}$$

$$o_t = \sigma(W_{xo}^T x_t + W_{ho}^T h_{t-1} + b_o) \tag{2.23}$$

$$g_t = \tanh(W_{xg}^T x_t + W_{hg}^T h_{t-1} + b_g) \tag{2.24}$$

$$c_t = f_t \times c_{t-1} + i_t \times g_t \tag{2.25}$$

$$y_t = h_t = o_t \times \tanh(c_t), \tag{2.26}$$

where σ is the sigmoid activation function, $W_{\cdot\cdot}$ are the weight matrices controlling corresponding connections, and $b_{\cdot}$ are the biases. Current input x_t and the previous hidden state h_t are passed as an input to four parallel fully connected layers. g_t is the output of the main layer, while i_t, f_t, o_t denote the output vectors of input, forget and output gate, respectively. Main layer's output is partially stored in the long term state c. The other three layers act as gate controllers. The usage of the sigmoid activation function means that their output has values from 0 to 1, which corresponds to the gates being "open" or "closed". At each iteration state c_t:

- goes through the forget gate, which decides what part of the long-term state should be forgotten,
- memorises some information from g_t, where the output i_t controls what information is important,
- adds some information to o_t, that is then used to compute the current final output y_t (which is the same as the hidden state h_t),
- passed on to the next iteration.

Thus, the LSTM can recognise which input is important (through the input gate), store this input (in the long-term state) for as long as it is needed (through the forget gate) and use it when it is needed (through the output gate). The aforementioned characteristics make it a great tool for processing long sequences.

GRU

A simplified version of LSTM, the Gated Recurrent Unit (GRU), was proposed by Cho et al. (2014). Here, instead of forget and input gates, one gate z is used. The output of this gate equal to 1 is an equivalent of the forget gate being open and input gate being closed, and the other way around when the output is 0. Moreover, only one state h is present and passed across the iterations. Also the output gate is not present, instead, the hidden state output vector h_t is used as the output. There is an

additional controller r, whose task is to decide what information from the previous state should be passed on to the input. Mathematically, it is expressed as:

$$z_t = \sigma(W_{xz}^T x_t + W_{hz}^T h_{t-1} + b_z) \tag{2.27}$$

$$r_t = \sigma(W_{xr}^T x_t + W_{hr}^T h_{t-1} + b_r) \tag{2.28}$$

$$g_t = \tanh(W_{xg}^T x_t + W_{hg}^T (r_t \times h_{t-1}) + b_g) \tag{2.29}$$

$$h_t = z_t \times h_{t-1} + (1 - z_t) \times g_t. \tag{2.30}$$

Even though it is a simplified version of the LSTM, there seem to be no bigger differences in how they work, and thus it is also very popular. Both cells have shown to achieve high results, especially in natural language processing tasks.

Bi-directional RNNs

An another extension to RNNs are bi-directional recurrent neural networks (BRNNs). While a normal RNN is only capable of capturing the previous context of data, the BRNNs also utilise the future context (Salehinejad et al., 2017). It achieves that by having two RNNs - one processing the input forward in time from start to end, and the other backwards from end to start. There are no interactions between the hidden states of the RNN layers and only their outputs are accumulated to obtain the output sequence. This method can be also applied with the aforementioned LSTM (BiLSTM) and GRU (BiGRU) cells . The downsides of this approach are that it doubles the complexity of the network and it is only applicable to input sequences with known start and end.

Types of RNN architectures

RNNs can be categorised into four types based on the input and output type (Chandrakesan, 2021):

- sequence-to-vector - the input is a sequence $(x)_t$ and only the last output of the output sequence $(y)_t$ is used. This architecture is useful for classification tasks such as sentiment analysis or intent recognition.

- vector-to-sequence - only the first input x_0 has a value, the rest of the input sequence is equal to 0 and the output is a sequence $(y)_t$. An example of such model is image description, where x_0 is the image.

- sequence-to-sequence - both the input and the output are a sequence. Such architecture can be used for sequence labelling tasks.
- encoder-decoder - here, the model can be split into two phases: sequence-to-vector and vector-to-sequence. The output vector of the first phase is the input vector of the second one. Models with such schema might perform better than sequence-to-sequence models, as the information about the whole input is passed before generating the output sequence (Bengio, Goodfellow, and Courville, 2017).

RNNs for intent recognition and slot filling

As mentioned before, the recurrent neural networks are a great tool for solving NLP tasks. There are many papers discussing their use for intent recognition, slot filling, and the joint task.

Ravuri and Stolcke (2015a) proposed both RNN and LSTM based approaches to solve the intent detection problem on ATIS corpus, with LSTM showing better results than RNN and both improving on the more traditional approaches. Later, the same authors showed a comparative study between RNN, LSTM and GRU solutions, where LSTM and GRU achieved comparably good performance (Ravuri and Stoicke, 2015b). Sreelakshmi et al. (2018) described a system that utilises a deep BiLSTM with enriched GloVe embeddings, which further improved the results of intent recognition.

For the slot filling task Peng et al. (2015), following up on Yao's previous work on RNNs and language understanding (Yao et al., 2013) (Yao et al., 2014), he compares a simple RNN, LSTM and RNN with external memory, with the latter achieving the best results. In other works Mesnil et al. (2013) and Mesnil et al. (2015) investigated different types of RNNs, also achieving higher performance than the traditional approaches.

Zhang and Wang (2016) were the first to introduce RNNs, namely GRU cells, to tackle the joint task problem. They highlight two main advantages of that approach over separate tasks, first one being the lower time needed to train the model and the other being the higher performance. Later, more authors researched different architectures. Zhou et al. (2016) propose a hierarchical LSTM solution with two LSTM layers - the first layer's final state was used for intent detection and the second layer's hidden states for slot labelling. Hakkani-Tür et al. (2016) explored different architectures utilising LSTMs to construct a multi-domain joint model, with BiLSTM

showing best results. More alternatives emerged based on BiLSTMs: a model with separate losses for both tasks (Zheng, Liu, and Hansen, 2017) or a model with shared BiLSTM layer trained on word embeddings and slot tags and intent derived from hidden states matrix (Kim, Lee, and Stratos, 2018). Wen et al. (2018) investigated usage of both contextual and hierarchical approaches and their various combinations. It is worth noting that in these recurrent methods the information sharing is implicit between two problems - usually only through the learning process using a joint loss. A different technique is proposed by e.g. Wang, Shen, and Jin (2018), where two paths based on BiLSTMs are defined. The hidden states are shared between the paths. The loss however is calculated separately for the two tasks and the training is done in an asynchronous manner: first intent recognition is done on the training batch, and then slot tagging.

2.2.3.10 Attention

Bahdanau, Cho, and Bengio (2014) introduced attention to improve machine translation. It was meant to tackle the limitations of a typical decoder-encoder RNN architecture, where the decoder would not get the whole information that comes with an input. Instead of passing just the last hidden state from the encoder, all hidden states are transferred. Each hidden state is associated with a certain word in the input sequence. During decoding, an attention score α_{ti} is computed for each hidden representation of a word h_i in order to compute context vector c_t for output at current position t (Hu, 2020):

$$e_{ti} = a(s_{t-1}, h_i) \tag{2.31}$$

$$\alpha_{ti} = \text{softmax}(e_{ti}) \tag{2.32}$$

$$c_t = \sum_i \alpha_{ti} h_i, \tag{2.33}$$

where s_{t-1} is the previous hidden state of the decoder, $a(\cdot)$ is an alignment function that measures how similar are the two tokens. The target token y_t and decoder hidden state are computed as follows:

$$y_t = f_y(s_{t-1}, yt - 1, c_t) \tag{2.34}$$

$$s_t = f_h(s_{t-1}, y_t) \tag{2.35}$$

where f_y and f_h denote the output layer and hidden layer in the encoder. The procedure is done until the last token y_t is generated. Through this architecture insufficient memory of RNNs is addressed. Additionally, the attention score ensures

that the focus is on the right part of a sequence when decoding. It is worth noting that Luong, Pham, and Manning (2015) also proposed a major implementation of attention mechanism for machine translation, however, the underlying concept of attention remained the same.

Attention mechanism can be generalised and applied to any sequence-to-sequence problem. Consider $V = v_i \in \mathbb{R}^{n \times d_v}$, a sequence of vector elements. The previous steps are then rewritten as:

$$e_i = a(u, v_i) \tag{2.36}$$

$$\alpha_i = \text{softmax}(e_i) \tag{2.37}$$

$$c = \sum_i \alpha_i v_i, \tag{2.38}$$

where $u \in \mathbb{R}^{d_u}$ is a problem-specific pattern vector (a query) that is matched with each vector from V via the alignment function $a(u, v)$. The alignment function outputs a scalar $e_i \in \mathbb{R}$ that signals the quality of match. Assuming $d_v = d_u$, the usual choices for the alignment functions are:

- Multiplicative:

$$u^T v \tag{2.39}$$

$$u^T W v \tag{2.40}$$

- Additive:

$$w_2^T \tanh(W_1[u; v]) \tag{2.41}$$

- Applied with a neural network:

$$\sigma(w_2^T \tanh(W_1[u; v] + b_1) + b_2) \tag{2.42}$$

Attention score $\alpha_i \in \mathbb{R}$ is the normalised weight for each v_i. It is then used to encode the whole sequence V into a context vector $c \in \mathbb{R}^{d_u}$. The context is later passed as an additional input to the downstream task. The v_i that matches u best is assigned the highest weight, and thus dominates the encoded context.

Attention for intent recognition and slot filling

Attention mechanism has been also explored in the context of NLU. Costello et al. (2018) proposed a multi-layer ensemble model for intent classification that utilises CNN character-level embedding and bidirectional CNN with attention mechanism.

Zhu and Yu (2016) investigated the attention mechanism with BiLSTMs for slot filling. However, a joint task was already solved before by Liu and Lane (2016). They explored the encoder-decoder model with aligned inputs and an attention-based RNN model, where the weighted sum of BRNN hidden states was passed to slot tagger and intent classifier. Later, Goo et al. (2018b) proposed a solution with more explicit attention. Here, slot and intent context vectors are computed from hidden states of a BiLSTM. Additionally, a slot-gated mechanism is introduced to improve slot labelling with the use of intent context. It is one of early examples of intent2slot circuits, where the intent prediction information is explicitly used for slot prediction. Li, Li, and Qi (2018) utilised self-attention (explained further in section 2.2.3.12) to achieve context-aware representations of the embeddings. Context vectors and embeddings are the passed to a BiLSTM. Based on the weighted average of hidden states the intent label is predicted. To predict slot labels a slot-gated mechanism is used, where the word gate is an output of a linear transformation of another self-attention layer and intent embedding.

2.2.3.11 Capsule Neural Networks

Capsule Neural Networks (CapsNets) emereged as an alternative to CNNs and were first introduced by Hinton, Krizhevsky, and Wang (2011). They were designed to tackle the shortcomings of CNNs such as inability to recognise pose or parts of the image (Kwabena Patrick et al., 2022). A single capsule consists of a network of neurons. Each neuron is responsible for outputting a different characteristic of a feature. The input to a capsule is a feature map produced by a CNN and the output are vectors, contrary to CNNs scalar values. To be precise, a capsule returns a probability that the feature it is responsible for is present, and instantiation parameters, which is a set of vector values. The first part of the output ensures invariance of the network, while the latter one equivariance. The fact that a model is invariant makes it so that the feature is recognised no matter where it is on the input. On the other hand, equivariance enables the model to identify pose, deformations or texture. It also guarantees that the spatial location of a feature is considered when making the final decision. Thus, comparing it to CNNs, not only the sole presence of a feature is taken into account, but also where it is e.g. on an image.

Capsule Neural Networks for intent recognition and slot filling

The solution proposed in Hinton, Krizhevsky, and Wang (2011) was built with pose recognition in mind, instead of object recognition. As an improvement, Sabour, Frosst, and Hinton (2017) proposed a capsule network which was not only constrained to that task. It incorporated dynamic routing algorithms to estimate the pose features. Based on it, Zhao et al. (2018) investigated using CapsNets in text classification tasks. He empirically showed that capsule networks can perform well in such problems. Xia et al. (2018) utilised capsules for intent detection in a zero-shot setting. They construct three capsules. First, SemanticCaps, is built on BRNN with multiple self-attention heads (explained in more detail in the next section) and its purpose is to get semantic features from utterances. After that DetectionCaps processes the low-level information from SemanticCaps by aggregating it through dynamic routing-by-agreement protocol and returns a high-level prediction vector. The protocol helps teach the model a hierarchy of feature detectors. In the algorithm the output of lower-level capsules is passed on to the higher-level one only if there is a strong agreement that a feature is informative to a higher-level capsule. Additionally, a Zero-shot DetectionCap is used to detect emerging intents. Later, Zhang et al. (2018b) proposed a hierarchical model for the joint task. Here also three capsules are defined. The architecture is composed of:

1. WordCaps, which learn contextual word representations

2. SlotCaps, that based on these representations tag words with slot types using the dynamic routing-by-agreement algorithm and build slot representation by aggregating words associated with each slot

3. IntentCaps, which detect the intent based on the predicted slot representation and utterance contexts. After that the slots are once again recognised utilising the inferred intent.

The proposed solution showed very good performance on both tasks, matching intent detection results of the other CapsNets approach and surpassing others, such as Hakkani-Tür et al. (2016) or Liu and Lane (2016) on SNIPS and ATIS datasets.

2.2.3.12 Transformer

The previous definition of attention can be looked at as external attention. The context vector is computed based on matches between elements of the input vector v_i and an external pattern u (Hu, 2020). In self-attention the external pattern is replaced with parts of an internal pattern taken from the sequence. By computing

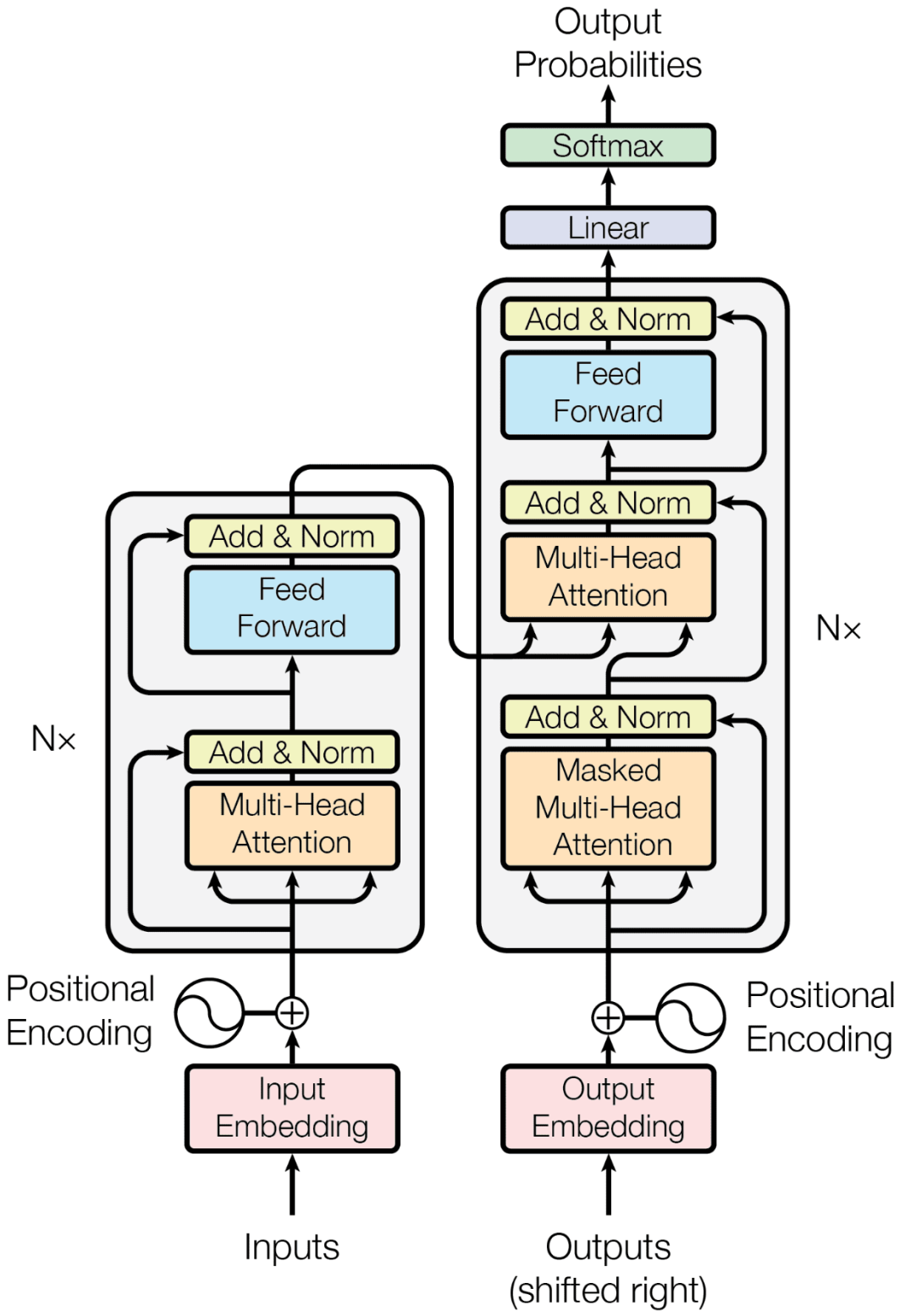


Figure 2.7: The Transformer model architecture (Vaswani et al., 2017)

pairwise self-attention we receive a representation of a sequence, that encapsulates the relationships between the elements. The most notable use of self-attention was described by Vaswani et al. (2017), where the Transformer model was proposed.

Transformer was the first model that relied purely on self-attention, without the usage of RNNs or CNNs, to calculate input and output representations. The original encoder is built with six layers on top of each other, each one made of two sub-layers: multi-head self-attention layer and position-wise fully connected FFNN. The output of each sub-layer is added its input through residual connections and normalised. The decoder is built with six layers as well. However, apart from the two sub-layers described before, it also incorporates a third one between the other two, which implements a multi-head attention over the output of the encoder. Here, the self-attention layer is also masked, which guarantees that the predictions can only depend on the decoder outputs computed before. All sub-layers, as well as the embeddings, output vectors of dimension $d_{model} = 512$. For a graphical representation see Figure 2.7.

The authors define the attention "as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors". In such definition the alignment score is computed between the query and a key corresponding to the value. The attention function used in the Transformer is called scaled dot-product attention. Its illustration is presented in Figure 2.8a. Mathematically, it can be described as follows:

$$e_i = a(q, k_i) = \frac{q \cdot k_i}{\sqrt{d_k}} \tag{2.43}$$

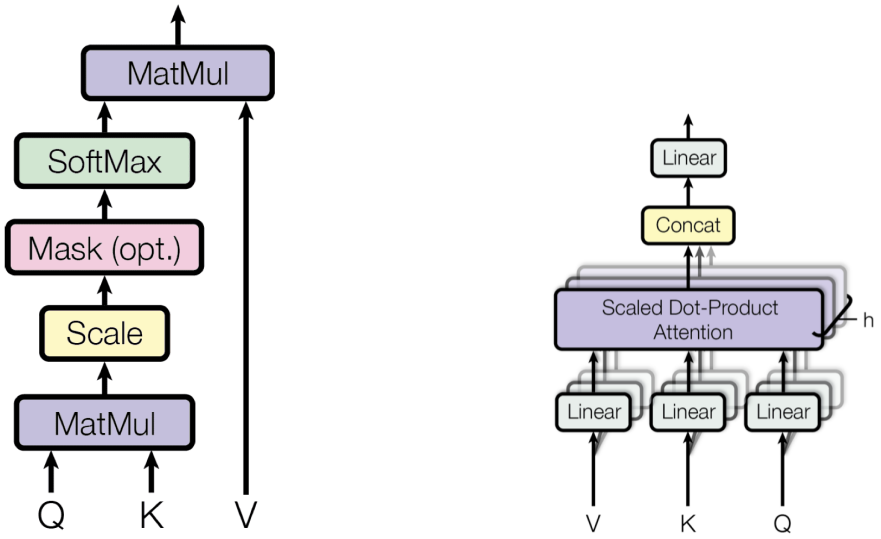
$$\alpha_i = \text{softmax}(e_i) \tag{2.44}$$

$$c = \sum_i \alpha_i v_i, \tag{2.45}$$

where $q \in \mathbb{R}^{d_k}$ is the query, and $k_i \in \mathbb{R}^{d_k}$ and $v_i \in \mathbb{R}^{d_v}$ are a corresponding key-value pair. In practice, this operation can be done on a set of queries simultaneously. With the queries, keys, and values packed into Q, K, V matrices, respectively, we can compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}). \tag{2.46}$$

Vaswani et al. also introduced multi-head attention in their model, which allowed it to collectively process the information from different representation subspaces at distinct positions. The queries, keys and values are projected h times, each with a different, linear projection into a d_k, d_k, d_v dimmensions. Then, the attention function



(a) (b)

Figure 2.8: (a) Scaled Dot-Product Attention; (b) Multi-Head Attention (Vaswani et al., 2017)

is parallelly computed on the projected inputs, after which they are concatenated and projected again to get the final result. Formally, it can be denoted as follows:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O, \tag{2.47}$$

where

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.48}$$

and $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ indicate the projection matrices. Graphical representation of multi-head attention can be seen on Figure 2.8b.

In both the encoder and decoder, a FFNN is used. Each position from the input passes through it separately, but the transformations are the same. The FFNN consists of two sub-layers with ReLU function between them, and can be looked at as two convolutions with kernel size 1.

Since the model does not use RNNs or CNNs, the information about the order of the sequence has to be passed in some other way. To preserve it, positional encoding is used. The encodings are added to the embeddings at the bottom of the encoder and the decoder. The authors proposed using sine and cosine functions for that:

$$PE_{(pos,2i)} = \sin(pos/n^{2i/d_{model}}) \tag{2.49}$$

$$PE_{(pos,2i+1)} = \cos(pos/n^{2i/d_{model}}), \tag{2.50}$$

where pos is the position, i is the dimension, and n is a user defined scalar (set to 10000 in the Transformer).

BERT

This novel architecture led to huge advances in NLP. One of the next big milestone models, that works especially well with NLU tasks is BERT (Devlin et al., 2018). The model is built from a stack of L Transformer encoder layers. The base version consists of 12 layers with 768 hidden states and 12 attention heads. The input sequence to the model can be either one or two sentences. The elements of the sequence are tokens vectorised with WordPiece embeddings. The first token is always a special classification token `[CLS]`. Its last hidden state can be later used for sequence level classification tasks. To separate two sentences an another special token `[SEP]` is introduced. Moreover, the information about which part of the sequence corresponds to which sentence, additional information is added to the embedding (apart from the positional encoding).

There are main steps defined in the framework: pre-training and fine-tuning. The pre-training step consists of two unsupervised tasks. The first one is Masked Language Modelling (MLM). During this phase deep bidirectional representations of tokens are learned. Before only unidirectional Language Models were explored (e.g. GPT (Radford et al., 2018)), as bidirectional conditioning would mean that each token would be able to indirectly see itself in a multi-layer context, which would lead to the model being able to easily predict the target word. To counteract this obstacle, a random small portion of the tokens is masked, and the objective is to predict them based on their context. The second task is binary next sentence prediction. The training data is generated from a monolingual unlabeled corpus. 50% of the inputs are paired sentences A and B, where B comes after A, and in the other 50% B is an another random sentence from the corpus. This objective helps the model understand the relationship between two sentences, which is not ensured with just language modelling.

In the fine-tuning step BERT is being adjusted to perform a specific task. For text classification and sequence tagging this is done by passing the objective-related as the first sentence A and masking the second sentence B with ∅ tokens. Then the output representation of the `[CLS]` token is used for predicting the sentence level class, and the token representations for sequence labelling. During training all of the models

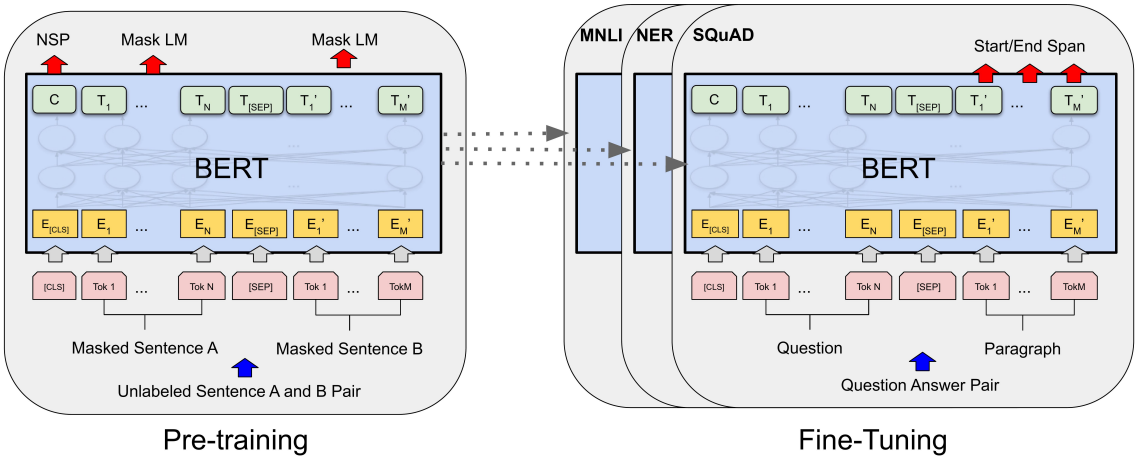


Figure 2.9: The pre-training and fine-tuning phases of BERT (Devlin et al., 2018).

parameters are fine-tuned end-to-end. This operation is usually very inexpensive compared to the pre-training phase. The two steps are illustrated in Figure 2.9.

Joint slot filling and intent classification with BERT

Chen, Zhuo, and Wang (2019) proposed a joint intent classification and slot filling model based on BERT. In their solution they simply extended the pre-trained BERT with two classification heads. The intent was predicted based on the hidden state corresponding to the special `[CLS]` token h_1 passed as an input to a softmax layer:

$$y^i = \text{softmax}(W^i h_1 + b^i), \tag{2.51}$$

while the slots were tagged by passing hidden states of the sequence tokens to another softmax layer:

$$y^s_n = \text{softmax}(W^s h_n + b^s), n \in 1...N \tag{2.52}$$

where h_n is a hidden state corresponding to a first WordPiece sub-token of word x_n. The model was fine-tuned end-to-end by minimising the joint cross-entropy loss of the two tasks. The authors they also explored adding a CRF for slot tagging on top of the joint BERT model instead of the softmax layer. The approach achieved a new state-of-the-art performance at the time, surpassing the results of previously mentioned solutions.

It is also worth noting that later a supposedly upgraded version of BERT was created called RoBERTa (Liu et al., 2019). In this approach, next sentence prediction was removed from the pre-training step. At the same time the masked language

modelling was performed with modified hyperparameters: much larger mini-batches and learning rates. The authors reported improvements on many downstream task benchmark sets. However, as written in (Hardalov, Koychev, and Nakov, 2020) there were no improvements on using it for the joint SLU task, same as with the addition of a CRF layer.

2.3 Polish Language Understanding

Most of natural language understanding research is done on English and Chinese (Rybak et al., 2020). All of the aforementioned solutions were evaluated on strictly English datasets. However, there might be differences in how they work with other languages. As present work's focus is on Polish language, below I am going to present known difficulties one may encounter in natural language processing of Polish language, as well as current research and advances made in that field.

2.3.1 Characteristics of Polish language

Polish language is the sixth biggest official language in European Union, with more than 40 million native speakers world-wide (Ogrodniczuk et al., 2022). It is the official language in the Republic of Poland, a country with over 38 million residents. Polish is categorised as a Slavic language, and as such it shares most of the common characteristics associated with that group. In the context of NLP there are a few features that may make it more complicated to process it compared to Germanic and Romance languages (Przepiórkowski, 2007). The two regarded as the most problematic are rich nominal inflection and free word order.

2.3.1.1 Rich nominal inflection

In Polish language there are 7 defined cases: nominative (mianownik), genitive (dopełniacz), dative (celownik), accusative (biernik), instrumental (narzędnik), locative (miejscownik), and vocative (wołacz). The inflected form depends e.g. on the gender type, of which there are 5 in Polish. Moreover, a noun or an adjective can have the same morphological suffix in more than one declension. This inflection system is famously complex and hard to master, even for native speakers. As reported by (Ogrodniczuk et al., 2022), there are roughly 180 thousand base forms of words and almost 4 million inflected word forms. Przepiórkowski (2007) also mentions the different inflection of homonymous common and proper nouns or the inflection of foreign

names as other difficulties in the context of Named Entity Recognition. As slot tagging is a very similar task, these may affect the performance of the system.

2.3.1.2 Free word order

Relative free word order in Polish means that the words in a sentence can occur in different places without changing its meaning. Nominal inflection keeps the information about sentence roles like subjects, objects, or matters (Przepiórkowski, 2007). For example, the sentence *Alice has a cat* is naturally translated as *Ala ma kota*. However, *Kota ma Ala* is also a viable way to say it, just the focus of the sentence turns from *Alice* to *cat*. Other versions, so: *Ala kota ma*, *Kota Ala ma*, *Ma Ala kota*, or *Ma kota Ala*, although not as natural, still preserve the essence of it. In English changing the order to *A cat has Alice* would completely change the meaning of the sentence. It means that the positions of the words do not entail as much information, which might make it harder to model the language or at least require more training data.

2.3.2 State of Polish NLP

According to Ogrodniczuk et al. (2022) Polish language belongs to *fragmentary support* group based on the availability language specific resources in European Language Grid (ELG)[1], which means that on average "the language is present in $\geq 3\%$ and $< 10\%$ of the ELG resources of the same type". That puts it on par with most of the other European languages. In comparison, English is the only language in the *good support* group, as it can be found in $\geq 30\%$ of the resources on average.

However, the state of Polish NLP is constantly improving, especially in recent times. The report attributes it to considerable funding of researchers by research infrastructures, as well as the deep learning breakthrough that accelerated the development of the NLP field. The architectures and tools, that were originally created for and tested with English language, are easily convertible to other languages.

Lately, many pre-trained word embeddings or language models have emerged for Polish language, e.g. Polish versions of BERT (Polbert (Kłeczek, 2020) and HerBERT (Rybak et al., 2020)) or RoBERTa (Dadas, Perełkiewicz, and Poświata, 2020). Moreover, multilingual models are available, such as Multilingual BERT (mBERT) (pretrained on the top 104 languages with the largest Wikipedia) or SlavicBERT

[1] https://live.european-language-grid.eu/

(Arkhipov et al., 2019), which was pre-trained on Russian, Bulgarian, Czech and Polish, but the weights were initialised with the ones from Multilingual BERT.

In terms of data, the most well known corpus NKJP (National Corpus of Polish) is not freely available for use because of copyright issues. Because of that researchers turn to use a mix of openly accessible datasets, such as Wikipedia dumps or Open Subtitles for pre-training purposes. For evaluation, KLEJ (Rybak et al., 2020) benchmark was created as an answer to an English benchmark GLUE (Wang et al., 2018). It consists of three sentiment analysis and singular paraphrase, NER classification, semantic relatedness textual entailment, cyberbullying detection and question answering tasks. For the intent recognition and slot filling problem there are two datasets available: MASSIVE (FitzGerald et al., 2022) and Leyzer (Sowański and Janicki, 2020), which I will explore in greater detail in the next chapter of present paper. To the best of my knowledge this problem was not previously explored in Polish language. Most of publicly available Polish NLP resources are gathered by (Dadas, 2019) on Github.

Chapter 3

Data

In this chapter I am going to describe in greater details the two datasets that are used in the experiment. Both of them are annotated textual datasets for the SLU task in Polish language. The domains represented in the data are strictly connected with virtual assistants.

3.1 MASSIVE

MASSIVE (*M*ultilingual *A*mazon *S*LU Resource Package (SLURP) for *S*lot filling, *I*ntent classification, and *V*irtual assistant Evaluation) is a human-created virtual assistant utterance evaluation dataset. Created by FitzGerald et al. (2022), a team from Amazon, it consists of over one million examples spanning across 51 languages (52 after Catalan was added). The corpus was constructed based on SLURP (Bastianelli et al., 2020), a Spoken Language Understanding Resource Package. While SLURP consists of both written and audio data, MASSIVE is a purely textual dataset. The multilingual one was created by parallelly translating the original English seed by human workers. Utterances represent 18 domains and 60 intents, with 55 unique slots defined. An example utterance with its slot tagging in BIO format and intent class is shown in Table 3.1.

For my experiment only the Polish part of the corpora was chosen. Its statistics per domain are presented in Table 3.2. As depicted, there are 10641, 1945 and 2916 utterances in train, development and test sets, respectively. Number of intents per

Word	obudź	mnie	o	dziewiątej	rano	w	piątek
Slot	O	O	O	B-time	I-time	O	B-date
Intent	alarm_set						

Table 3.1: Example utterance and its tagging from MASSIVE. (own elaboration)

domain	# intents	# unique slots	# train utt.	# dev utt.	# test utt.
alarm	3	13	361	61	92
audio	4	8	216	28	58
calendar	3	20	1578	274	396
cooking	2	14	198	42	68
datetime	2	8	337	63	102
email	4	17	918	152	267
general	3	18	621	119	187
iot	9	13	589	100	202
lists	3	17	476	105	139
music	4	15	315	55	80
news	1	10	477	80	123
play	5	26	1297	251	385
qa	5	15	1151	213	286
recommendation	3	14	419	68	93
social	2	14	389	68	106
takeaway	2	14	248	44	57
transport	4	16	554	109	124
weather	1	10	497	113	151
Total	60	55	10641	1945	2916

Table 3.2: Statistics of intents, possible slots, and utterances (utt.) per dataset across domains in Massive. Notice that slots can occur in more than one domain. (own elaboration)

domain spreads from 1 to 9, with the majority of domains having between 2 and 4 intents. There are the most utterances in *calendar*, *play* and *qa* domains, while the least numerous domains are *audio*, *cooking* and *takeaway*, as depicted on Figure 3.1. There is also a wide range of examples per intent, ranging from 4 to 796 in the train dataset. The slot labels are not exclusive to the domain, with most of them being shared. The number of words in utterances also varies, with most sentences having 5 words. All of these characteristics represent a real life scenario of a virtual assistant, where some intents can have less representative queries than the other.

In the preprocessing stage I have removed duplicated utterances. If an utterance was present in both train and test examples, I decided to remove the ones from the training set. I have also removed one example from the training set that had 50 words, which was more than twice as much as the longest sentence in the train part of the dataset. Apart from that I also needed to create the BIO annotations for the slot filling task, as they were not provided with the corpus.

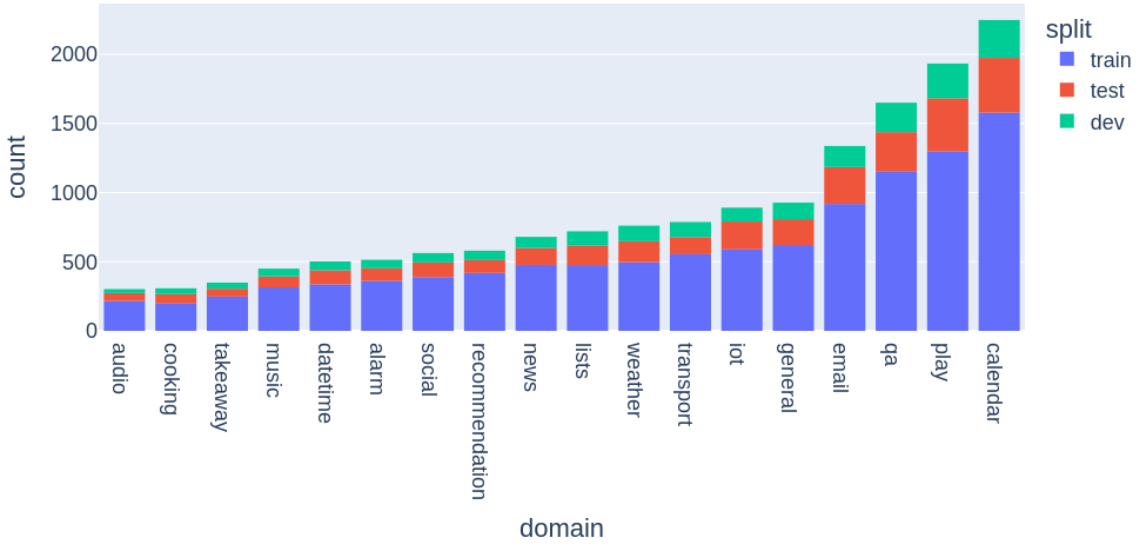


Figure 3.1: Distribution of examples per domain in MASSIVE in each split of the dataset: train, test and development. (own elaboration)

3.2 Leyzer

Leyzer is a multilingual text corpus, that was created to study cross-lingual transfer learning in NLU systems. It was designed by Sowański and Janicki (2020) and consists of parallel English, Spanish and Polish datasets. The utterances were generated using JSpeech Grammar Formats (JSGF). The authors assure that the fact that the dataset was created using grammar patterns does not influence the naturalness of the examples. At the same time it provides a cheap way to generate a lot of data that covers most of the variations. The corpora has also integrated slot expansion from slot dictionaries, which enables data augmentation. An example utterance with its slot tagging in BIO format and intent class is shown in Table 3.3.

Again, in my experiment I have only utilised the Polish part of the dataset. The subset grammars were created by linguists based on the English seed. In the original paper there were 186 intents and 86 slots across 20 domains reported. At the time of writing the thesis an updated version of the corpus was released, with 194 intents across 21 domains. My version of it was generated on that basis and its statistics are visible in Table 3.4. After the generation step and removal of duplicates, the whole corpus had 25814 labeled utterances with BIO annotations. My train-development-test split was created with 80-10-10 proportions. The number of intents per domain ranges from 4 to 18, with the most domains having 10 intents. Similarly to MASSIVE,

Word	włącz	moją	playlistę	crash	into	me
Slot	O	O	O	B-playlist	I-playlist	I-playlist
Intent	PlayPlaylist					

Table 3.3: Example utterance and its tagging from Leyzer. (own elaboration)

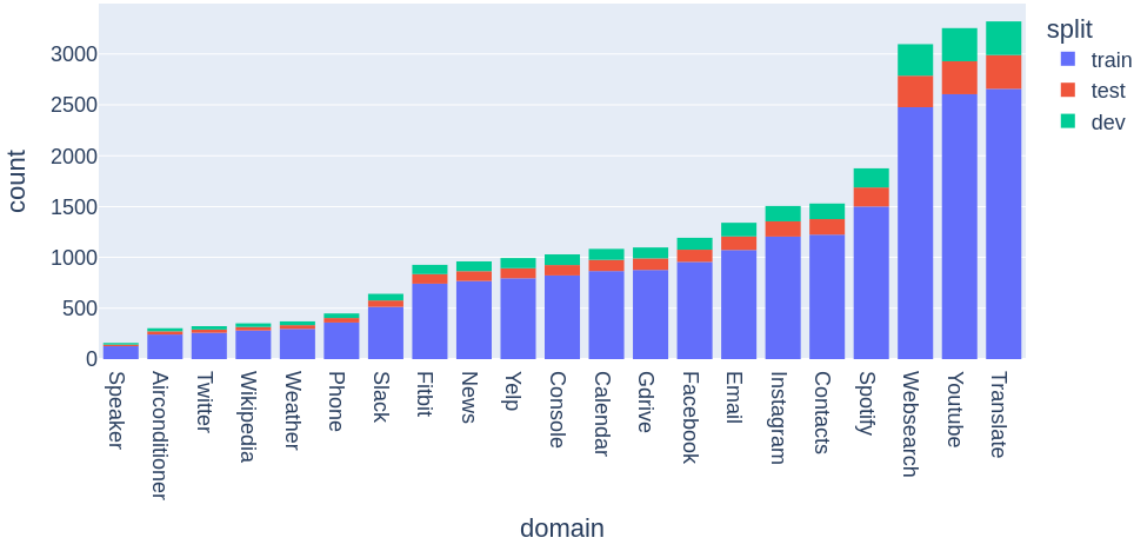


Figure 3.2: Distribution of examples per domain in Leyzer in each split of the dataset: train, test and development. (own elaboration)

the number of utterances per domain varies significantly, with *Speaker* (159 examples in total) and *Translate* (3321 examples in total) being the least and most abundant domains (also shown on Figure 3.2, while intent utterance count ranges from 10 to 2408. As described by the authors, these characteristics come from the design of the dataset and were supposed to capture the real-life scenario best.

domain	# intents	# unique slots	# train utt.	# dev utt.	# test utt.
Airconditioner	10	3	242	30	32
Calendar	10	3	866	108	110
Console	6	5	823	105	102
Contacts	11	4	1349	169	169
Email	11	7	1073	134	134
Facebook	7	4	956	117	120
Fitbit	5	2	742	91	94
Gdrive	13	5	877	107	114
Instagram	10	7	1205	151	150
News	4	2	768	96	97
Phone	6	3	349	44	42
Slack	14	8	514	65	63
Speaker	7	1	127	16	16
Spotify	18	6	1501	188	187
Translate	9	22	2658	331	332
Twitter	6	3	259	33	32
Weather	10	2	297	37	37
Websearch	7	2	2477	310	310
Wikipedia	8	1	282	36	34
Yelp	12	6	795	100	99
Youtube	10	3	2533	317	315
Total	194	81	20693	2585	2589

Table 3.4: Statistics of intents, possible slots, and utterances (utt.) per dataset across domains in Leyzer. Again slots can occur in more than one domain. (own elaboration)

Chapter 4

Experiment

4.1 Aim of the experiment

The aim of the experiment is to investigate how well Polish language models can deal with the joint task of intent recognition and slot filling. As the architecture for my experiment I picked the Joint BERT architecture (Chen, Zhuo, and Wang, 2019). I decided on this particular solution as it is a state-of-the-art method that emphasises the power of language modelling the most, which at the moment drives the development of Polish NLU. To the best of my knowledge, such experiment has not yet been performed in Polish language.

4.2 Experimental setup and scenarios

The experiment was implemented in Python using PyTorch (Paszke et al., 2019) and transformers (Wolf et al., 2019) libraries. The code is an adjusted version of the implementation specified on `https://github.com/monologg/JointBERT` and is published on `https://github.com/matgora/JointBERT`. As my contribution I first added the two corpora in a desired format and configurations for four new models: Polbert, HerBERT, Polish RoBERTa, and Multilingual BERT. The latter model is used as a baseline for comparison. Lastly, I implemented early stopping regularisation.

Apart from the steps described in the previous chapter, no significant preprocessing step is needed when using BERT, as it uses all of the context in the sentence. Alzahrani and Jololian (2021) showed that on an example of gender of the author classification, where BERT achieved best performance when no preprocessing technique was used.

To summarise, 16 models are trained: for each of the two datasets, both Joint BERT and Joint BERT+CRF architectures are tested on each of the four aforemen-

dataset	model	opt.	lr	batch size	max seq len	crf
Leyzer	**mBERT**	Adam	5e-5	128	72	False
	HerBERT	Adam	5e-5	128	78	False
	Polbert	Adam	5e-5	128	66	False
	RoBERTa	Adam	5e-5	128	78	False
	mBERT	Adam	5e-5	128	72	True
	HerBERT	Adam	5e-5	128	78	True
	Polbert	Adam	5e-5	128	66	True
	RoBERTa	Adam	5e-5	128	78	True
MASSIVE	**mBERT**	Adam	5e-5	128	64	False
	HerBERT	Adam	5e-5	128	35	False
	Polbert	Adam	5e-5	128	37	False
	RoBERTa	Adam	5e-5	128	57	False
	mBERT	Adam	5e-5	128	64	True
	HerBERT	Adam	5e-5	128	35	True
	Polbert	Adam	5e-5	128	37	True
	RoBERTa	Adam	5e-5	128	57	True

Table 4.1: Experiment cases. (opt. - optimiser, l.r. - learning rate, max. seq. len. - maximal sequence length, crf - whether the CRF layer was used (True) or not (False)) (own elaboration)

tioned language models. Each model is fine-tuned with most of the hyperparameters set as in Chen, Zhuo, and Wang (2019): Adam (Kingma and Ba, 2014) is used as an optimiser with an initial learning rate 5e-5, the dropout probability is 0.1, and batch size of 128. Maximal number of epochs is set to 40, but training is performed with early stopping with patience 7. The maximal length of input sequence is adjusted to each language model and dataset. Each BERT comes with it's own pre-trained tokenizer and the outputs of each tokenizer can differ. Moreover, the Leyzer dataset had many utterances which had over 60 tokens (mostly due to *console* domain), while in MASSIVE the sentences were much shorter (see Table 4.1). The models were trained on NVIDIA GeForce RTX 2080.

4.3 Evaluation metrics

4.3.1 Intent classification

For intent classification intent accuracy is used, as it is the most commonly used evaluation metric (Weld et al., 2022). Accuracy is calculated as the number of correct predictions divided by the number of all examples.

4.3.2 Slot filling

In the slot tagging task the span slot micro-averaged precision, recall and F1 is used. Span is the sequence of words with the same label. For example, a span of class VALUE could be represented in BIO notation as B-VALUE I-VALUE I-VALUE.

Precision and recall can be calculated for a span of class C as:

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN}, \tag{4.2}$$

where:

- TP is the number of True positives, i.e. spans of class C correctly labelled
- FP is the number of False positives, i.e. spans of another class incorrectly labelled as of class C
- FN is the number of False negatives, i.e. spans of class C incorrectly (partially) labelled as of a different class

Micro-averaged precision and recall are calculated with TP, FP and FN summed over all classes:

$$Precision = \frac{\sum TP}{\sum TP + \sum FP} \tag{4.3}$$

$$Recall = \frac{\sum TP}{\sum TP + \sum FN}. \tag{4.4}$$

Then, the F1 score is calculated as:

$$F1 = \frac{2 \times Precission \times Recall}{Precision + Recall}. \tag{4.5}$$

4.3.3 Joint task

Semantic frame accuracy is the joint evaluation metric. An utterance is predicted correctly as a whole when both intent and all slots have the right label assigned. Then, the semantic frame accuracy is the number of correctly evaluated sentences divided by the number of all examples in a test set. This metric can also be refered to as exact match accuracy.

Chapter 5

Results

Table 5.1 shows the models performance as intent recognition accuracy, slot filling F1, and semantic frame accuracy on test sets of the Leyzer and MASSIVE datasets. The column **crf** informs whether the CRF layer was used (True) or not (False). Figures 5.1 to 5.4 visualise the training process as the change of training and development sets' loss over training epochs.

As depicted in Table 5.1, Multilingual BERT-based model achieves the highest overall results on Leyzer with intent detection accuracy of 99.11%, slot filling F1 of 99.32%, and semantic frame accuracy of 98.68%. This might be surprising, as other models are solely Polish language models, and should be better fitted for this task. However, the second best model considering all metrics turns out to be Polbert with CRF layer, with higher intent recognition at 99.23%, but slightly lower slot filling F1 (99.30%) and semantic frame accuracy (98.61%). There is a very little difference between the two models and it may just come down to the randomness of weight initialisation at the start of the training. The other reason mBERT performs better or just as well as Polish models might be that in the corpus, there are English name entities like artists in *Spotify* domain or phrases from other languages in *Translate* domain, which a multilingual language model should deal with better. The worst performing, especially on the slot filling task, are models based on Polish RoBERTa, with 96.30% and 97.01% slot filling F1 score for models with and without the CRF layer, respectively. Comparing it with other solutions which achieve around 99% F1 on slot labelling task, this score is very low. However, overall all models achieve very high results on the Leyzer dataset. Looking and Figures 5.1 and 5.2, we can see that Polbert based model achieved the lowest training and development loss. In the case with the CRF layer it took the longest to train (19 epochs), however, without CRF early stopping halted the training the fastest at 14 epochs. The best model, based on Multilingual BERT, took 21 epochs to train.

dataset	model	intent acc.	slot F1	semantic frame acc.	crf
Leyzer	**mBERT**	99.11%	**99.32%**	**98.68%**	False
	HerBERT	98.96%	99.22%	98.37%	False
	Polbert	99.15%	99.24%	98.53%	False
	RoBERTa	98.72%	97.01%	95.67%	False
	mBERT	98.99%	99.11%	98.34%	True
	HerBERT	97.17%	98.93%	96.59%	True
	Polbert	**99.23%**	99.30%	98.61%	True
	RoBERTa	98.30%	96.30%	95.39%	True
MASSIVE	**mBERT**	83.61%	69.28%	57.99%	False
	HerBERT	85.46%	68.62%	58.74%	False
	Polbert	**85.97%**	69.36%	**60.32%**	False
	RoBERTa	80.59%	52.71%	44.51%	False
	mBERT	81.28%	69.57%	55.93%	True
	HerBERT	83.26%	**71.80%**	59.26%	True
	Polbert	84.26%	70.92%	59.95%	True
	RoBERTa	79.42%	56.60%	46.98%	True

Table 5.1: Slot filling and intention recognition results on two datasets. (own elaboration)

On MASSIVE, the best overall model turns out to be based on monolingual Polbert, with the highest intent recognition accuracy (85.97%) and semantic frame accuracy (60.32%), and slot filling F1 of 69.36%. In the latter metric, HerBERT with CRF achieves the highest performance (71.80%). However, the addition of CRF to all models lowered the intent classification accuracy. On this corpus multilingual BERT performs a bit worse than Polish monolingual BERT models. However, Polish RoBERTa again achieves the lowest results. The model without CRF scores 5.38% relative less on intent detection accuracy, 16.65% relative less on slot filling F1, and 15.81% relative less on semantic frame accuracy then Polbert.

Interestingly, in the results presented by FitzGerald et al. (2022), their model based on a multilingual RoBERTa, XLM-R (Conneau et al., 2020), with the same joint architecture and trained on 51 languages, achieved higher exact match accuracy (60.9%), with slightly lower intent classification accuracy (85.8%) and slot labelling F1 (69.0%) on Polish evaluation set. Their other models, which were based on mT5 (Xue et al., 2021), scored even better. Obviously, the bigger amount of training data, larger vocabularies and number of parameters, or different architectures have influenced the higher results, but this once again shows the power of multilingual models. However, especially the bigger size might be a concern when developing a virtual assistant. Ultimately, the models would be stored and used on the end device, which usually

does not have the capabilities to do so with a large model. Monolingual language models are smaller, but probably still too big to be utilised in that way. Nevertheless, there is still a lot of room for improvement when it comes to modelling this dataset, both for mono- and multilingual models.

On Figures 5.3 and 5.3 the training process on MASSIVE of all considered models. In both cases, with and without CRF, Polbert achieved the lowest development set loss. Moreover, early stopping halted the training the fastest for these models (after 10th epoch). RoBERTa with CRF took the longest to train (16 epochs). Second slowest was HerBERT, finding the local minimum after 7 epochs both with and without CRF.

Overall, out of the three monolingual Polish models, Polbert seems to work best on the joint task of intent recognition and slot filling. Polish RoBERTa definitely is the worst match for this problem. However, compared to Multilingual BERT, Polbert and HerBERT did not achieve much higher performance, with mBERT performing better than any other model on Leyzer, although only by a small margin. The addition of CRF shows mixed results. On Leyzer it helped Polbert achieve the highest overall score in intent accuracy, while on MASSIVE HerBERT with CRF has the best slot filling F1, but at the cost of lower intent detection accuracy.

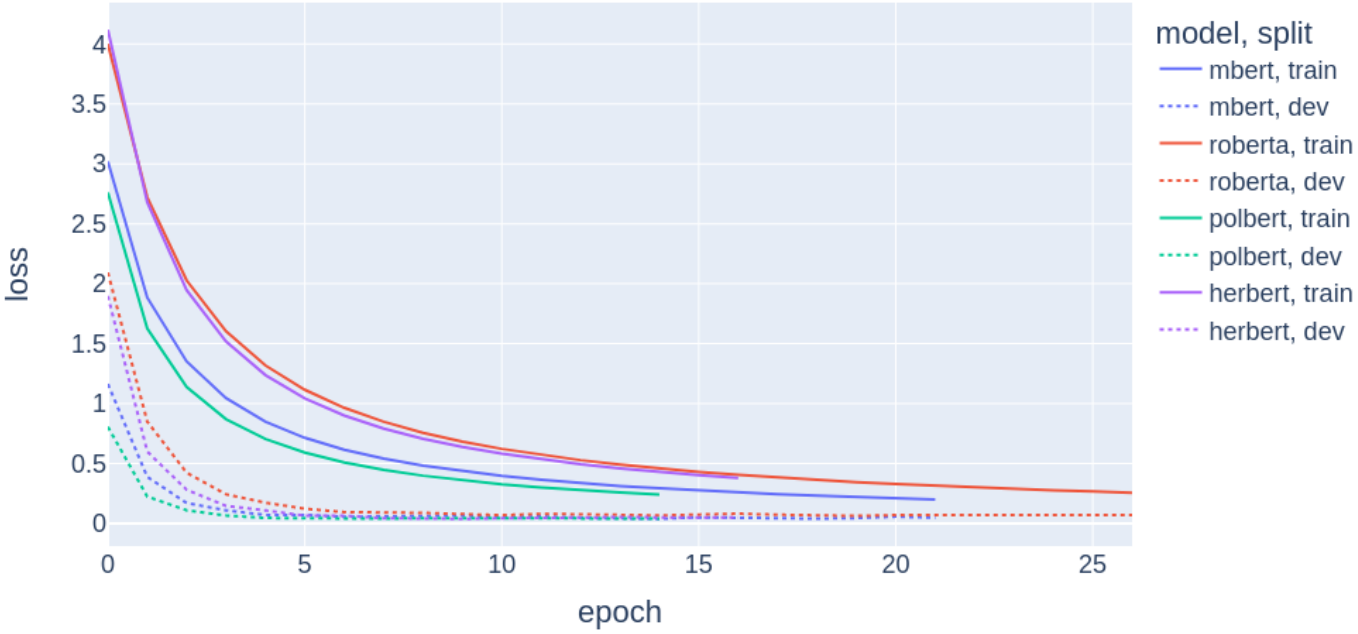


Figure 5.1: Train and development sets losses during training (Leyzer) (own elaboration)

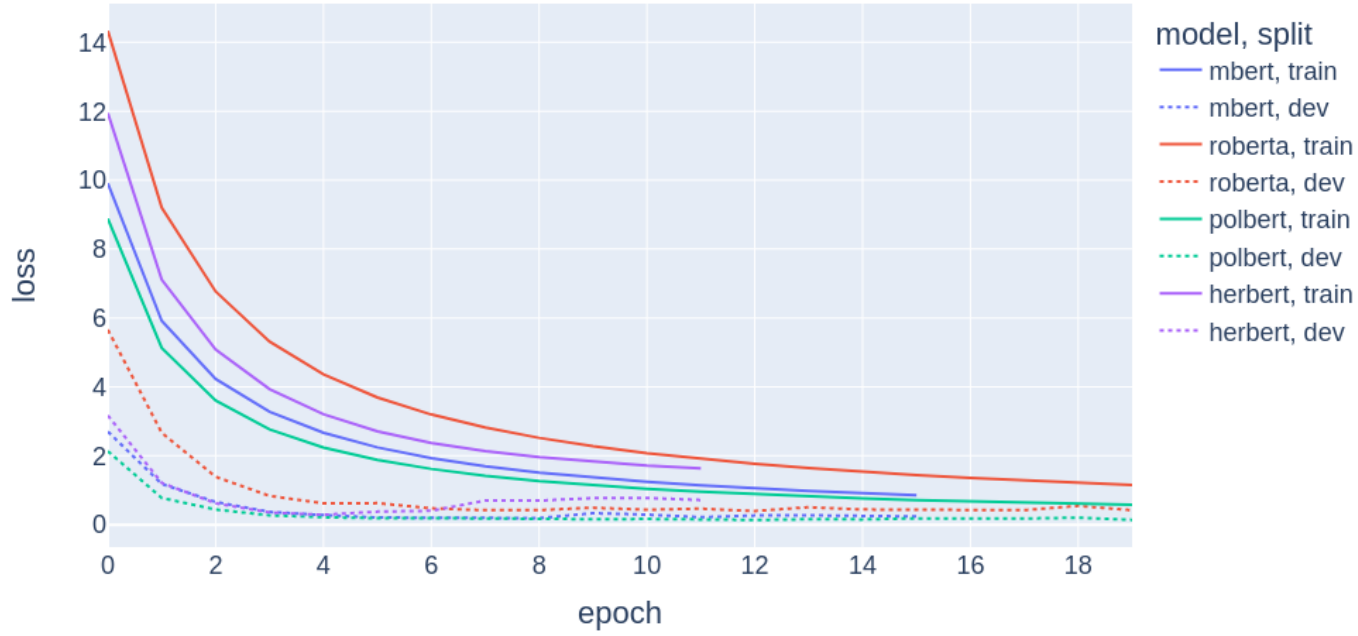


Figure 5.2: Train and development sets losses during training (Leyzer+CRF) (own elaboration)

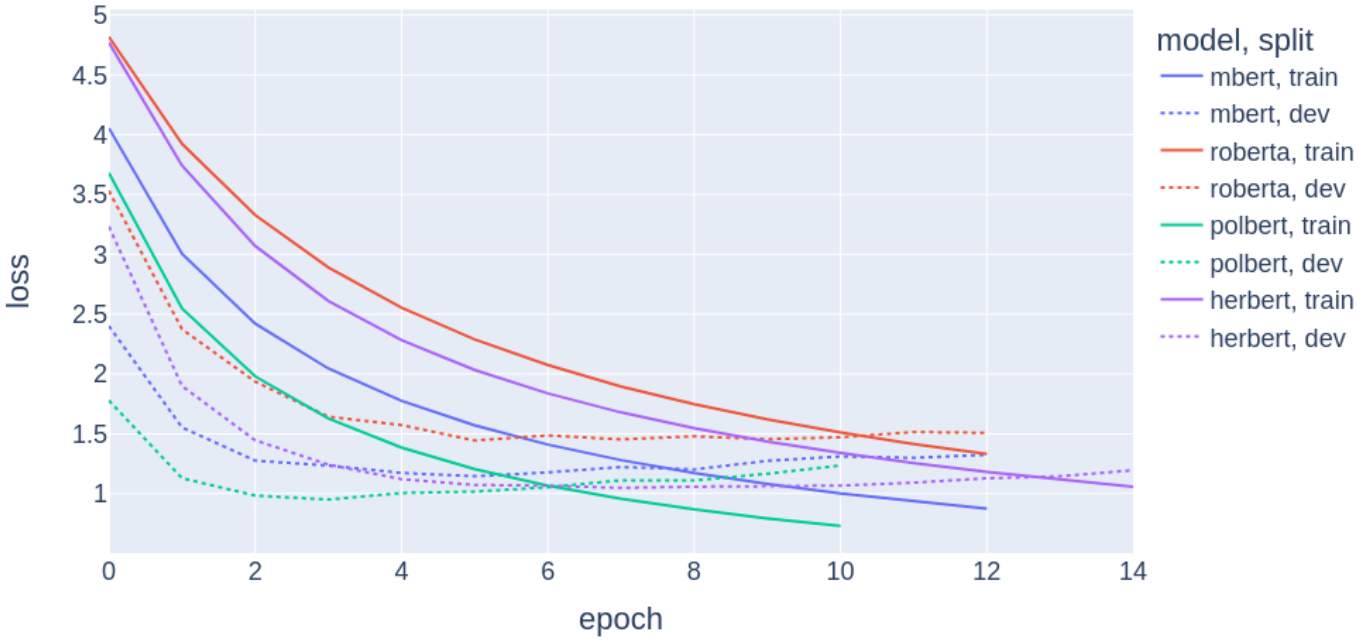


Figure 5.3: Train and development sets losses during training (MASSIVE) (own elaboration)

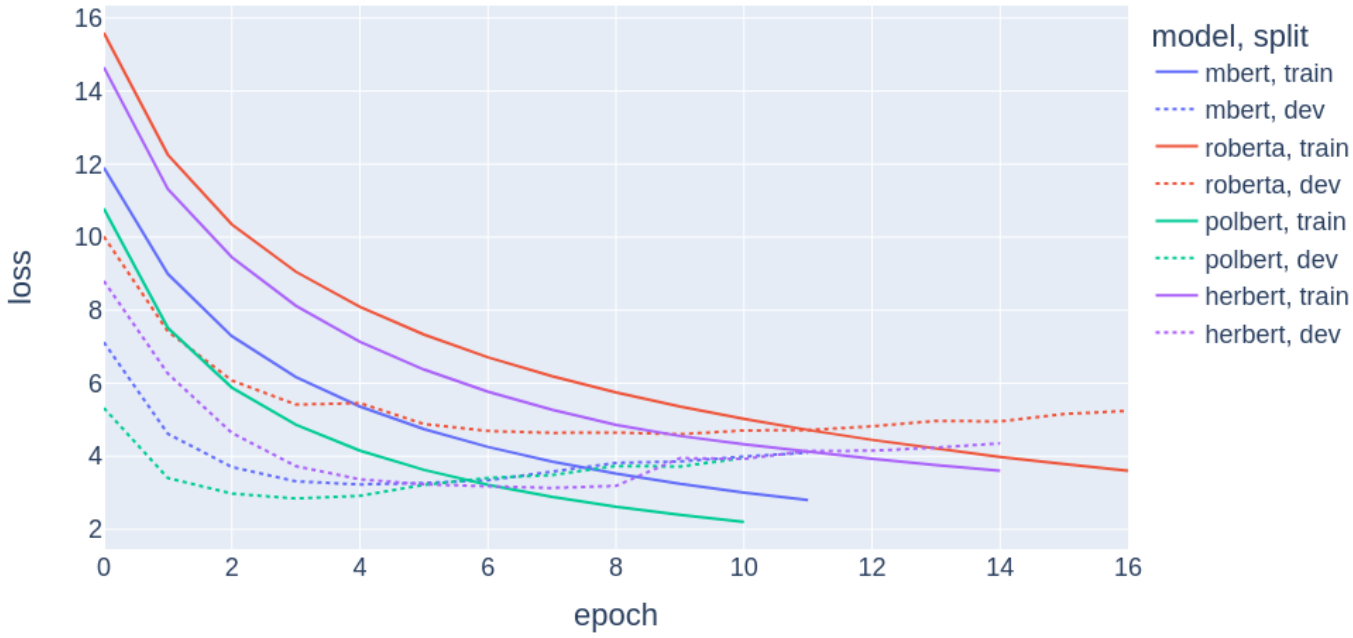


Figure 5.4: Train and development sets losses during training (MASSIVE+CRF) (own elaboration)

Chapter 6

Conclusions

In present thesis I presented the problem of spoken language understanding in Polish language. My contribution in this paper can be summarised as follows

- In the first part of Chapter 2, I described the basics of natural language understanding, how it is a vital part of virtual assistants and defined the tasks which are solved in that domain: intent recognition and slot filling.
- I reported a detailed literature review of the history of tackling these objectives in the second section of Chapter 2. Methods from classical to modern, neural network ones, for both separate and joint setting were analysed there.
- In the latter part of Chapter 2, a summary of Polish language and Polish language processing is showed.
- Chapter 3 presents the data used for the research experiment described in Chapter 4.
- In Chapter 5 the results of the experiment are described.

In my research I evaluated Polish language models and compared them to a multilingual one on joint intent detection and slot labelling task. The solutions were analysed on subsets of two representative corpora: MASSIVE and Leyzer. The results on Leyzer show that spoken Polish language understanding is solvable by both mono- and multilingual language models. However, the experiment performed on MASSIVE indicate that it still has some development potential. Polish monolingual resources can deal with that problem on par or better than multilingual ones.

Chapter 7

Discussion and future work

The results presented in this work show a great promise for building a purely Polish virtual assistant. However, looking at how well multilingual models are performing the question might appear, whether working on a unified solution would not be better. One direction for future work could be evaluating zero-shot and few-shot approaches on multilingual models, to see how much additional labour is needed to achieve the same results as in a monolingual model.

In present work I have only explored encoder-based models. There are, however, existing solutions that use text-to-text or generative models for solving NLU problems (FitzGerald et al., 2022)(Ahmad et al., 2021). As an extension of this paper I could explore these solution in the context of Polish language.

The other concern I mentioned in my thesis is the model size. Large language models are not yet suitable for running on the end devices. The way around it, is to set up a back-end service, for example on a cloud, and communicate with it over the network. Nevertheless, this approach is not optimal due to latency and privacy issues. Moreover, if each user would have their own version of a model it could enable better product personalisation. Thus, an another idea for future work could be investigating smaller models and how well they could deal with the SLU problem.

Lastly, as described in present paper, virtual assistants are composed of not only an NLU engine, but also other NLP components, like automatic speech recognition or natural language generation. To build a fully functioning Polish VA, all of these need to be at a high enough standard. This is an another area of research that should be explored in order to achieve a good full product.

References

Ahmad, Wasi Uddin et al. (2021). "Intent classification and slot filling for privacy policies". In: *arXiv preprint arXiv:2101.00123*.

Alzahrani, Esam and Leon Jololian (2021). "How Different Text-preprocessing Techniques Using The BERT Model Affect The Gender Profiling of Authors". In: *CoRR* abs/2109.13890. arXiv: 2109.13890. URL: https://arxiv.org/abs/2109.13890.

Arkhipov, Mikhail et al. (Aug. 2019). "Tuning Multilingual Transformers for Language-Specific Named Entity Recognition". In: *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*. Florence, Italy: Association for Computational Linguistics, pp. 89–93. DOI: 10.18653/v1/W19-3712. URL: https://www.aclweb.org/anthology/W19-3712.

Asli, Celikyilmaz et al. (Dec. 2011). "Exploiting distance based similarity in topic models for user intent detection". In: DOI: 10.1109/ASRU.2011.6163969.

Bahdanau, Dzmitry, Kyunghyun Cho, and Y. Bengio (Sept. 2014). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ArXiv* 1409.

Bastianelli, Emanuele et al. (2020). "SLURP: A Spoken Language Understanding Resource Package". In: *CoRR* abs/2011.13205. arXiv: 2011.13205. URL: https://arxiv.org/abs/2011.13205.

Bengio, Y., Patrice Simard, and Paolo Frasconi (Feb. 1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 5, pp. 157–66. DOI: 10.1109/72.279181.

Bengio, Yoshua, Ian Goodfellow, and Aaron Courville (2017). *Deep learning*. Vol. 1. MIT press Cambridge, MA, USA.

Bikel, Daniel M., Richard M. Schwartz, and Ralph M. Weischedel (1999). "An Algorithm that Learns What's in a Name". In: *Machine Learning* 34, pp. 211–231.

Bojanowski, Piotr et al. (2016). "Enriching Word Vectors with Subword Information". In: *CoRR* abs/1607.04606. arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606.

Borthwick, Andrew Eliot (1999). "A Maximum Entropy Approach to Named Entity Recognition". AAI9945252. PhD thesis. USA. ISBN: 0599472324.

Camacho, Cezanne (2019). *CNNs for Text Classification*. URL: https://cezannec.github.io/CNN_Text_Classification/ (visited on 06/02/2023).

Carpuat, Marine (2017). *The Perceptron*. URL: http://www.cs.umd.edu/class/spring2017/cmsc422/slides0101/lecture06.pdf (visited on 06/09/2023).

Celikyilmaz, Asli and Dilek Hakkani-Tur (July 2012). "A Joint Model for Discovery of Aspects in Utterances". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Jeju Island, Korea: Association for Computational Linguistics, pp. 330–338. URL: https://aclanthology.org/P12-1035.

Chandrakesan, Gopi (2021). *Day 43 – Sequence-to-Sequence, Sequence-to-Vector, and Vector-to-Sequence*. URL: https://www.gopichandrakesan.com/day-43-sequence-to-sequence-sequence-to-vector-and-vector-to-sequence/ (visited on 06/28/2023).

Chen, Qian, Zhu Zhuo, and Wen Wang (2019). *BERT for Joint Intent Classification and Slot Filling*. DOI: 10.48550/ARXIV.1902.10909. URL: https://arxiv.org/abs/1902.10909.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Chopra, Romansha et al. (2017). "Sequence Labeling using Conditional Random Fields". In: *International Journal of u-and e-Service, Science and Technology* 10.9, pp. 101–108.

Chowdhury, Gobinda G. (Jan. 2003). "Natural language processing". English. In: *Annual Review of Information Science and Technology* 37.1, pp. 51–89. ISSN: 0066-4200. DOI: 10.1002/aris.1440370103.

Conneau, Alexis et al. (July 2020). "Unsupervised Cross-lingual Representation Learning at Scale". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 8440–8451. DOI: 10.18653/v1/2020.acl-main.747. URL: https://aclanthology.org/2020.acl-main.747.

Costello, Charles et al. (2018). "Multi-Layer Ensembling Techniques for Multilingual Intent Classification". In: *CoRR* abs/1806.07914. arXiv: 1806.07914. URL: http://arxiv.org/abs/1806.07914.

Crystal, D. (2005). *How Language Works*. Penguin books. Penguin Books. ISBN: 9780140515381. URL: https://books.google.pl/books?id=-3tiAAAAMAAJ.

Dadas, Sławomir (2019). *A repository of Polish NLP resources*. Github. URL: https://github.com/sdadas/polish-nlp-resources/.

Dadas, Sławomir, Michał Perełkiewicz, and Rafał Poświata (2020). *Pre-training Polish Transformer-based Language Models at Scale*. DOI: 10.48550/ARXIV.2006.04229. URL: https://arxiv.org/abs/2006.04229.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Dowding, John et al. (June 1993). "GEMINI: A Natural Language System for Spoken-Language Understanding". In: *31st Annual Meeting of the Association for Computational Linguistics*. Columbus, Ohio, USA: Association for Computational Linguistics, pp. 54–61. DOI: 10.3115/981574.981582. URL: https://aclanthology.org/P93-1008.

FitzGerald, Jack et al. (2022). *MASSIVE: A 1M-Example Multilingual Natural Language Understanding Dataset with 51 Typologically-Diverse Languages*. DOI: 10.48550/ARXIV.2204.08582. URL: https://arxiv.org/abs/2204.08582.

Freund, Yoav and Robert E Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1, pp. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4, pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251. URL: https://doi.org/10.1007/BF00344251.

Gal, Yarin and Zoubin Ghahramani (2016). *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*. arXiv: 1512.05287 [stat.ML].

Gandhi, Rohith (2018). *Support Vector Machine — Introduction to Machine Learning Algorithms*. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (visited on 05/15/2023).

Goo, Chih-Wen et al. (June 2018a). "Slot-Gated Modeling for Joint Slot Filling and Intent Prediction". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 753–757. DOI: 10.18653/v1/N18-2118. URL: https://aclanthology.org/N18-2118.

— (June 2018b). "Slot-Gated Modeling for Joint Slot Filling and Intent Prediction". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 753–757. DOI: 10.18653/v1/N18-2118. URL: https://aclanthology.org/N18-2118.

Haffner, Patrick, Gokhan Tur, and Jerry Wright (Oct. 2003). "Optimizing Svms For Complex Call Classification". In: *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on*. DOI: 10.1109/ICASSP.2003.1198860.

Hakkani-Tür, Dilek et al. (2016). "Multi-Domain Joint Semantic Frame Parsing using Bi-directional RNN-LSTM". In: *Proceedings of The 17th Annual Meeting of the International Speech Communication Association (INTERSPEECH 2016)*. ISCA. URL: https://www.microsoft.com/en-us/research/publication/multijoint/.

Hardalov, Momchil, Ivan Koychev, and Preslav Nakov (2020). "Enriched Pre-trained Transformers for Joint Slot Filling and Intent Detection". In: *CoRR* abs/2004.14848. arXiv: 2004.14848. URL: https://arxiv.org/abs/2004.14848.

Hashemi, Homa Baradaran (2016). "Query Intent Detection using Convolutional Neural Networks". In.

Helmi Setyawan, Muhammad Yusril, Rolly Maulana Awangga, and Safif Efendi (Oct. 2018). "Comparison Of Multinomial Naive Bayes Algorithm And Logistic Regression For Intent Classification In Chatbot". In: pp. 1–5. DOI: 10.1109/INCAE.2018.8579372.

Hinton, Geoffrey E, Alex Krizhevsky, and Sida D Wang (2011). "Transforming auto-encoders". In: *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*. Springer, pp. 44–51.

Hirschberg, Julia and Christopher D. Manning (2015). "Advances in natural language processing". In: *Science* 349.6245, pp. 261–266. DOI: 10.1126/science.aaa8685. eprint: https://www.science.org/doi/pdf/10.1126/science.aaa8685. URL: https://www.science.org/doi/abs/10.1126/science.aaa8685.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hoy, Matthew B (2018). "Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants". In: *Medical Reference Services Quarterly* 37, pp. 81 –88.

Hu, Dichao (2020). "An introductory survey on attention mechanisms in NLP problems". In: *Intelligent Systems and Applications: Proceedings of the 2019 Intelligent Systems Conference (IntelliSys) Volume 2*. Springer, pp. 432–448.

Huang, Ting-Hao 'Kenneth', Yun-Nung Chen, and Jeffrey P. Bigham (2017). *Real-time On-Demand Crowd-powered Entity Extraction*. arXiv: 1704.03627 [cs.HC].

Jacovi, Alon, Oren Sar Shalom, and Yoav Goldberg (Nov. 2018). "Understanding Convolutional Neural Networks for Text Classification". In: *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Brussels, Belgium: Association for Computational Linguistics, pp. 56–65. DOI: 10.18653/v1/W18-5408. URL: https://aclanthology.org/W18-5408.

Jawahar, Ganesh, Benoît Sagot, and Djamé Seddah (July 2019). "What Does BERT Learn about the Structure of Language?" In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3651–3657. DOI: 10.18653/v1/P19-1356. URL: https://aclanthology.org/P19-1356.

Jeong, Minwoo and Gary Geunbae Lee (2008). "Triangular-Chain Conditional Random Fields". In: *IEEE Transactions on Audio, Speech, and Language Processing* 16.7, pp. 1287–1302. DOI: 10.1109/TASL.2008.925143.

Joyce, James (2021). "Bayes' Theorem". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2021. Metaphysics Research Lab, Stanford University.

Khan, Vasima and Tariq Azfar Meenai (2021). "Pretrained Natural Language Processing Model for Intent Recognition (BERT-IR)". In: *Human-Centric Intelligent Systems* 1 (3-4), pp. 66–74. ISSN: 2667-1336. DOI: 10.2991/hcis.k.211109.001. URL: https://doi.org/10.2991/hcis.k.211109.001.

Khurana, Diksha et al. (2023). "Natural language processing: state of the art, current trends and challenges". In: *Multimedia Tools and Applications* 82.3, pp. 3713–3744. ISSN: 1573-7721. DOI: 10.1007/s11042-022-13428-4. URL: https://doi.org/10.1007/s11042-022-13428-4.

Kim, Young-Bum, Sungjin Lee, and Karl Stratos (2018). "OneNet: Joint Domain, Intent, Slot Prediction for Spoken Language Understanding". In: *CoRR* abs/1801.05149. arXiv: 1801.05149. URL: http://arxiv.org/abs/1801.05149.

Kingma, Diederik and Jimmy Ba (Dec. 2014). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations*.

Krishnan, Vijay and Vignesh Ganapathy (2005). "Named entity recognition". In: *Stanford Lecture CS229*.

Kwabena Patrick, Mensah et al. (2022). "Capsule Networks – A survey". In: *Journal of King Saud University - Computer and Information Sciences* 34.1, pp. 1295–1310. ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2019.09.014. URL: https://www.sciencedirect.com/science/article/pii/S1319157819309322.

Kłeczek, Dariusz (2020). "Polbert: Attacking Polish NLP Tasks with Transformers". In: *Proceedings of the PolEval 2020 Workshop*. Ed. by Maciej Ogrodniczuk and Łukasz Kobyliński. Institute of Computer Science, Polish Academy of Sciences.

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira (2001). "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 282–289. ISBN: 1558607781.

LeCun, Y. et al. (1989). "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4, pp. 541–551. DOI: 10.1162/neco.1989.1.4.541.

Lee, Changsu, Youngjoong Ko, and Jungyun Seo (July 2015). "A Simultaneous Recognition Framework for the Spoken Language Understanding Module of Intelligent Personal Assistant Software on Smart Phones". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Beijing, China: Association for Computational Linguistics, pp. 818–822. DOI: 10.3115/v1/P15-2134. URL: https://aclanthology.org/P15-2134.

Lekic, Sasa and Kasper Liu (2019). "Intent classification through conversational interfaces : Classification within a small domain". In.

Li, Changliang, Liang Li, and Ji Qi (2018). "A Self-Attentive Model with Gate Mechanism for Spoken Language Understanding". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 3824–3833. DOI: 10.18653/v1/D18-1417. URL: https://aclanthology.org/D18-1417.

Li, Wei and Andrew Mccallum (Sept. 2003). "Rapid Development of Hindi Named Entity Recognition Using Conditional Random Fields and Feature Induction". In: *ACM Trans. Asian Lang. Inf. Process.* 2, pp. 290–294. DOI: 10.1145/979872.979879.

Liddy, Elizabeth D. (2001). "Encyclopedia of Library and Information Science, 2nd Ed." In: NY. Marcel Decker, Inc. Chap. Natural Language Processing.

Linnainmaa, Seppo (1970). "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". In: *Master's Thesis (in Finnish), University of Helsinki*, pp. 6–7.

Liu, Bing and Ian R. Lane (2016). "Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling". In: *CoRR* abs/1609.01454. arXiv: 1609.01454. URL: http://arxiv.org/abs/1609.01454.

Liu, Jiao, Yanling Li, and Min Lin (2019). "Review of Intent Detection Methods in the Human-Machine Dialogue System". In: *Journal of Physics: Conference Series* 1267.1, p. 012059. DOI: 10.1088/1742-6596/1267/1/012059. URL: https://dx.doi.org/10.1088/1742-6596/1267/1/012059.

Liu, Yinhan et al. (2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *CoRR* abs/1907.11692. arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning (2015). "Effective Approaches to Attention-based Neural Machine Translation". In: *CoRR* abs/1508.04025. arXiv: 1508.04025. URL: http://arxiv.org/abs/1508.04025.

Marsland, Stephen (2015). *Machine learning: an algorithmic perspective*. CRC press.

McCallum, Andrew, Dayne Freitag, and Fernando C. N. Pereira (2000). "Maximum Entropy Markov Models for Information Extraction and Segmentation". In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 591–598. ISBN: 1558607072.

McCallum, Andrew, Kamal Nigam, et al. (1998). "A comparison of event models for naive bayes text classification". In: *AAAI-98 workshop on learning for text categorization*. Vol. 752. 1. Madison, WI, pp. 41–48.

Mcculloch, Warren and Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". In: *Bulletin of Mathematical Biophysics* 5, pp. 127–147.

Mesnil, G. et al. (Jan. 2013). "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding". In: pp. 3771–3775.

Mesnil, Grégoire et al. (2015). "Using Recurrent Neural Networks for Slot Filling in Spoken Language Understanding". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.3, pp. 530–539. DOI: 10.1109/TASLP.2014.2383614.

Mikolov, Tomás et al. (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *CoRR* abs/1310.4546. arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

Ogrodniczuk, Maciej et al. (2022). *Report on the Polish Language*. Instytut Podstaw Informatyki Polskiej Akademii Nauk.

OpenAI (2022). *ChatGPT*. URL: https://openai.com/blog/chatgpt (visited on 06/14/2023).

— (2023). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL].

O'Shea, Keiron and Ryan Nash (2015). "An Introduction to Convolutional Neural Networks". In: *CoRR* abs/1511.08458. arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

Pardela, Paweł et al. (2022). "Federated Learning in Heterogeneous Data Settings for Virtual Assistants – A Case Study". In: *Text, Speech, and Dialogue*. Ed. by

Petr Sojka et al. Cham: Springer International Publishing, pp. 451–463. ISBN: 978-3-031-16270-1.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Patil, Rajvardhan et al. (2023). "A Survey of Text Representation and Embedding Techniques in NLP". In: *IEEE Access* 11, pp. 36120–36146. DOI: 10.1109/ACCESS.2023.3266377.

Peng, Baolin et al. (2015). "Recurrent Neural Networks with External Memory for Spoken Language Understanding". In: *Natural Language Processing and Chinese Computing*. Ed. by Juanzi Li et al. Cham: Springer International Publishing, pp. 25–35. ISBN: 978-3-319-25207-0.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Peters, Matthew E. et al. (2018). "Deep contextualized word representations". In: *CoRR* abs/1802.05365. arXiv: 1802.05365. URL: http://arxiv.org/abs/1802.05365.

Przepiórkowski, Adam (June 2007). "Slavic Information Extraction and Partial Parsing". In: *Proceedings of the Workshop on Balto-Slavonic Natural Language Processing*. Prague, Czech Republic: Association for Computational Linguistics, pp. 1–10. URL: https://aclanthology.org/W07-1701.

PWN (2023). *Słownik języka polskiego*. URL: https://sjp.pwn.pl/sjp/od;2492621.html (visited on 04/23/2023).

Rabiner, L.R. (1989). "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2, pp. 257–286. DOI: 10.1109/5.18626.

Radford, Alec et al. (2018). "Improving language understanding by generative pre-training". In.

Rasa (2023). *Rasa NLU: Language Understanding for Chatbots and AI Assistants.* URL: https://rasa.com/docs/ (visited on 05/15/2023).

Ravuri, Suman and Andreas Stoicke (2015b). "A comparative study of neural network models for lexical intent classification". In: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pp. 368–374. DOI: 10.1109/ASRU.2015.7404818.

Ravuri, Suman and Andreas Stolcke (Sept. 2015a). "Recurrent neural network and LSTM models for lexical utterance classification". In: pp. 135–139. DOI: 10.21437/Interspeech.2015-42.

Rosenblatt, F. (1957). *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory.

Ruder, Sebastian (2017). *An Overview of Multi-Task Learning in Deep Neural Networks*. arXiv: 1706.05098 [cs.LG].

Rybak, Piotr et al. (July 2020). "KLEJ: Comprehensive Benchmark for Polish Language Understanding". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 1191–1201. DOI: 10.18653/v1/2020.acl-main.111. URL: https://aclanthology.org/2020.acl-main.111.

Sabour, Sara, Nicholas Frosst, and Geoffrey E Hinton (2017). "Dynamic routing between capsules". In: *Advances in neural information processing systems* 30.

Salehinejad, Hojjat et al. (2017). "Recent advances in recurrent neural networks". In: *arXiv preprint arXiv:1801.01078*.

Schapire, Robert E. and Yoram Singer (2000). "BoosTexter: a boosting-based system for text categorization". English (US). In: *Machine Learning* 39.2, pp. 135–168. ISSN: 0885-6125. DOI: 10.1023/a:1007649029923.

Sowański, Marcin and Artur Janicki (Sept. 2020). "Leyzer: A Dataset for Multilingual Virtual Assistants". In: pp. 477–486. ISBN: 978-3-030-58322-4. DOI: 10.1007/978-3-030-58323-1_51.

Sreelakshmi, K et al. (2018). "Deep Bi-Directional LSTM Network for Query Intent Detection". In: *Procedia Computer Science* 143. 8th International Conference on Advances in Computing & Communications (ICACC-2018), pp. 939–946. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.10.341. URL: https://www.sciencedirect.com/science/article/pii/S1877050918320374.

Sutton, Charles, Andrew McCallum, et al. (2012). "An introduction to conditional random fields". In: *Foundations and Trends® in Machine Learning* 4.4, pp. 267–373.

Sydow, Marcin (2019). *Tools of AI - Course Web Page*. URL: http://users.pja.edu.pl/~msyd/nai-eng.html (visited on 06/09/2023).

Tur, G. and R. De Mori (2011). *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech*. Wiley. ISBN: 9781119993940. URL: https://books.google.pl/books?id=RDLyT2FythgC.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

Wang, Alex et al. (2018). "GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding". In: *CoRR* abs/1804.07461. arXiv: 1804.07461. URL: http://arxiv.org/abs/1804.07461.

Wang, Ye-Yi (2010). "Strategies for statistical spoken language understanding with small amount of data - an empirical study". In: *Interspeech*.

Wang, Yu, Yilin Shen, and Hongxia Jin (2018). "A Bi-model based RNN Semantic Frame Parsing Model for Intent Detection and Slot Filling". In: *CoRR* abs/1812.10235. arXiv: 1812.10235. URL: http://arxiv.org/abs/1812.10235.

Weischedel, Ralph et al. (1993). "Coping with Ambiguity and Unknown Words through Probabilistic Models". In: *Computational Linguistics* 19.2, pp. 359–382. URL: https://aclanthology.org/J93-2006.

Weld, Henry et al. (2022). "A Survey of Joint Intent Detection and Slot Filling Models in Natural Language Understanding". In: *ACM Comput. Surv.* Just Accepted.

ISSN: 0360-0300. DOI: 10.1145/3547138. URL: https://doi.org/10.1145/3547138.

Wen, Liyun et al. (Jan. 2018). "Jointly Modeling Intent Identification and Slot Filling with Contextual and Hierarchical Information". In: pp. 3–15. ISBN: 978-3-319-73617-4. DOI: 10.1007/978-3-319-73618-1_1.

Werbos, P. J. (1981). "Applications of Advances in Nonlinear Sensitivity Analysis". In: *Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC*, pp. 762–770.

Wolf, Thomas et al. (2019). "HuggingFace's Transformers: State-of-the-art Natural Language Processing". In: *CoRR* abs/1910.03771. arXiv: 1910.03771. URL: http://arxiv.org/abs/1910.03771.

Xia, Congying et al. (2018). "Zero-shot User Intent Detection via Capsule Neural Networks". In: *CoRR* abs/1809.00385. arXiv: 1809.00385. URL: http://arxiv.org/abs/1809.00385.

Xu, Puyang and Ruhi Sarikaya (2013). "Convolutional neural network based triangular CRF for joint intent detection and slot filling". In: *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 78–83. DOI: 10.1109/ASRU.2013.6707709.

Xue, Linting et al. (June 2021). "mT5: A Massively Multilingual Pre-trained Text-to-Text Transformer". In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, pp. 483–498. DOI: 10.18653/v1/2021.naacl-main.41. URL: https://aclanthology.org/2021.naacl-main.41.

Yalçın, Orhan G. (2021). *The Brief History of Convolutional Neural Networks*. URL: https://towardsdatascience.com/the-brief-history-of-convolutional-neural-networks-45afa1046f7f (visited on 06/01/2023).

Yao, Kaisheng et al. (Aug. 2013). "Recurrent Neural Networks for Language Understanding". In: DOI: 10.13140/2.1.2755.3285.

Yao, Kaisheng et al. (2014). "Spoken language understanding using long short-term memory neural networks". In: *2014 IEEE Spoken Language Technology Workshop (SLT)*, pp. 189–194. DOI: 10.1109/SLT.2014.7078572.

Zhang, Chenwei et al. (2018a). "Joint slot filling and intent detection via capsule neural networks". In: *arXiv preprint arXiv:1812.09471*.

Zhang, Chenwei et al. (2018b). "Joint Slot Filling and Intent Detection via Capsule Neural Networks". In: *CoRR* abs/1812.09471. arXiv: 1812.09471. URL: http://arxiv.org/abs/1812.09471.

Zhang, Xiaodong and Houfeng Wang (2016). "A Joint Model of Intent Determination and Slot Filling for Spoken Language Understanding". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*. IJCAI'16. New York, New York, USA: AAAI Press, 2993–2999. ISBN: 9781577357704.

Zhao, Wei et al. (2018). "Investigating Capsule Networks with Dynamic Routing for Text Classification". In: *CoRR* abs/1804.00538. arXiv: 1804.00538. URL: http://arxiv.org/abs/1804.00538.

Zheng, Yang, Yongkang Liu, and John H.L. Hansen (2017). "Intent detection and semantic parsing for navigation dialogue language processing". In: *2017 IEEE 20th*

International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6. DOI: 10.1109/ITSC.2017.8317620.

Zhou, Qianrong et al. (Oct. 2016). "A Hierarchical LSTM Model for Joint Tasks". In: vol. 10035. ISBN: 978-3-319-47673-5. DOI: 10.1007/978-3-319-47674-2_27.

Zhu, Su and Kai Yu (2016). "Encoder-decoder with Focus-mechanism for Sequence Labelling Based Spoken Language Understanding". In: *CoRR* abs/1608.02097. arXiv: 1608.02097. URL: http://arxiv.org/abs/1608.02097.

List of Figures

List of Tables