Karlsruhe Institute of Technology

A Personalised Dialogue System based on Person Identification

Master's Thesis of

Lukas Frank

at the Interactive Systems Lab
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT)

Reviewer:	Prof. Dr. Alexander Waibel
Second reviewer:	Prof. Dr. Tamim Asfour
Advisor:	M.Sc. Stefan Constantin

16. November 2019 – 15. May 2020

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, 15.05.2020

. . . . . .

(Lukas Frank)

Abstract

Interacting with computer systems using natural language is getting more and more common due to the virtual assistant systems. However, when using such dialogue systems one can notice that frequently general responses are given and the systems often cannot interactively learn facts about the user. A personalized and to the user adapted dialogue is the key for a natural feeling conversation.

In this work a Transformer model based approach to adapt the dialogue of a chat system based on the knowledge about the user is presented. A computer vision pipeline is used to identify the user, create a user profile and assign facts gathered through the dialogue to this profile. Because of the stored facts the system can adapt the dialogue and retrieve the facts later if needed.

These capabilities are demonstrated through an automatic evaluation, attention weight visualization and a human evaluation. A generated validation dataset is used for the automatic evaluation in order to check if the system behaves correctly in different scenarios. The capabilities like greeting a user, asking for the name, saving and querying a food preference of the user or saving and querying a location of an object are covered by the evaluation. The experiments show that the system is capable of storing and retrieving facts of a user and generating a proper response with an average F1 score of 89%.

Zusammenfassung

Die Interaktion und Steuerung von Computer Systemen durch natürliche Sprache wird wegen der raschen Entwicklung und Verbreitung von virtuellen Assistenten immer üblicher. Durch das Nutzen solcher Dialog Systeme kann man jedoch feststellen, dass häufig allgemeine Antworten gegeben werden. Außerdem sind die Systeme oft nicht in der Lage interaktiv Wissen über einen Nutzer aufzubauen. Ein personalisierter und an den Nutzer angepasster Dialog ist jedoch wichtig für eine sich natürlich anfühlende Konversation.

In dieser Arbeit wird ein Transformer Modell basierter Ansatz vorgestellt, der den Dialog eines Chat Systems personalisiert basierend auf dem Wissen über einen Nutzer. Eine Bilderkennung wird verwendet um den Nutzer zu identifizieren und um ein Profil zu erstellen. Informationen die der Nutzer während einer Konversation preisgibt, werden dem Profil zugeordnet. Da das System Informationen über einen Nutzer behalten kann, können diese in dem weiteren Gesprächsverlauf ebenfalls genutzt werden.

Die Fähigkeiten des Systems werden durch eine automatische Evaluation, eine menschliche Evaluation und durch die Visualisierung der Gewichte des Netzes belegt. Ein synthetisch generierter Validierungsdatensatz wird bei der automatischen Evaluation verwendet, um zu prüfen, ob sich das System in verschiedenen Szenarien richtig verhält. Fähigkeiten wie das Begrüßen des Nutzers, das Erfragen des Namens, das Speichern und Verwenden von Essenspräferenzen oder das Speichern und Verwenden von präferierten Ortern von Gegenständen werden durch die Evaluation abgedeckt. Die Experimente zeigen, dass das System die Fähigkeiten besitzt Fakten zu speichern und diese im Dialog verwenden kann. Außerdem wird eine angemessene Antwort mit einem durchschnittlichen F1-Score von 89% gegeben.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In the recent years computer science research made a big progress in developing machine learning based systems. The society gets in touch with intelligent systems in many areas. The field of application of machine learning, especially deep neural networks, is ranging from search engines or diseases prediction to self driving cars or virtual personal assistants like Amazon Alexa or Siri. This technological influence also changes the interfaces between humans and technical systems and shifts it to a more natural manner like using natural language. However, while interacting with such a dialogue system one can notice a difference between the responses of a human or a machine. Often the dialogue does not feel natural because the systems are not capable of adapting the dialogue to facts mentioned by the user earlier in the conversation. Instead of that, general responses are given in the majority of the cases.

Below is an example of a usual conversation between a human and a dialogue system. The dialogue raises from a conversation with the voice assistant Amazon Alexa. It demonstrates the lack of the capability to save and recall information about the interlocutor. For humans it is a natural task to associate newly obtained information with an interlocutor and to use it in the later course of conversation.

HUMAN: Hi, my name is lukas.

SYSTEM: Hey.

HUMAN: I like to eat spaghetti

SYSTEM: Bingo, there are several recipes on chefkoch.

LATER IN THE DIALOGUE:

HUMAN: Do you remember my name?

SYSTEM: I don't know exactly who is talking.

HUMAN: I'm hungry, what can I eat?

SYSTEM: I am happy to help you. [1]

This work is describing the approach and the findings of the research to personalize a dialogue system using an explicit memory representation. The system utilizes a transformer

[1] Transcript of a sample dialogue with the voice assistant Amazon Alexa (https://alexa.amazon.de)

model based architecture to generate natural language responses. A computer vision component is used to manage the identities and the facts belonging to them. In combination the system is able to assign facts to an identity and to use them in the further course of the dialogue.

1.2 Goal

In this thesis the goal is to develop a prototype of a personalized dialogue where the domain is restricted to a kitchen scenario. The problem space was chosen to have the possibility to test the system later using the KIT assistance robot ARMAR as a platform. The idea is to empower the system with the capability to incrementally build up knowledge about different people interacting with the robot. For instance, the system should learn the name of the identity and associate further information with it. Information such as the preferred location of objects like a cup, or a plate, or the favorite food can be interactively learned. At a later time point in the dialogue the user is able to query this information.

The entire system contains a computer vision and a natural language processing component. The usage of information from an image input is needed to identify a person and to select accordingly a memory which contains facts about this person. The facts are injected into the natural language processing component in order to adapt the dialogue using this bias. To interact with the dialogue system itself, a chat interface is provided to allow the user to have a natural language conversation.

1.3 Structure

In Chapter 2 the relevant fundamentals are provided with a focus on natural language processing topics. Different architectures like recurrent neural networks as well as the transformer model are introduced including important concepts like the attention mechanism. Furthermore, a brief introduction on convolutional neural networks is given.

Related research is discussed in Chapter 3.

Chapter 4 introduces the developed approach. The base model GPT-2 is described along with the dialogue and the personalization component. The Chapter covers the data generation, training and inference of the model. The introduced special tokens and the approach how to utilize the memory is explained.

The used frameworks are described in Chapter 5. Apart from that the final dataset and the characteristics of it are given.

In Chapter 6 the results are presented including the methodology how the model was evaluated. Besides that, the used hyperparameters and end-to-end examples are provided.

Finally, in Chapter 7 the work is summarized, discussed and an outlook with further ideas to proceed with this research is given.

2 Fundamentals

This chapter provides a fundamental overview of the structure of neural networks, activation functions, learning problems and the training of neural networks. However, a basic knowledge is assumed as given in the history of machine learning approaches like the Bayes' Theorem, Rosenblatt perceptron [22], x-or problem, Support Vector Machines [23] and backpropagation. The following sections give a rough overview of relevant approaches in the field of computer vision and natural language processing.

The most basic architecture is called fully connected feedforward neural network or multilayer perceptron. Figure 2.1 visualises a graphical representation of a network with 3 layers. Every neuron (circle) is connected with all neurons of the next layer. A specific function f^* which maps input values x to output values y like $y = f^*(x)$ should be usually represented or approximated by a neural network. An example is, for instance, a function which takes a day of the year and maps the day to the temperature of this specific day. The training of a neural network f means to find a set of parameters θ which solves the problem: $\bar{y} = f(x, \theta)$ with $\bar{y} - y$ minimal according the ground truth data.

Once the network is used for more complex tasks like for image classification, usually a deeper model is chosen. A deep network means that several layers ($f^{(3)}$, $f^{(2)}$, $f^{(1)}$) are stacked on top of each other

$$f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$$

to apply the layers consecutively. The layers of a network usually contain a non-linear function σ, also called activation function. A network containing only linear layers can be contracted to a one linear layer network. This implies that the network can only approximate linear functions which means that, for instance, the xor problem is not solvable with such an architecture. A sigmoid $\sigma(x) = \frac{1}{1+e^{-1}}$, tanh $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ or relu $\sigma(x) = max(0, x)$ function are common activation functions to build a layer f^{hidden} such as $f^{hidden}(x) = \sigma(w^T x)$ with $w \subseteq \theta$.

The training of a neural network depends on the data and the chosen learning approach like supervised learning, unsupervised learning or reinforcement learning. In case of supervised learning the idea is to use data points including attached label attributes to check if the prediction of the neural network is correct. If the prediction is not correct, the learnable parameters (weights) are updated with the goal to have a smaller error in the next iteration. A loss function, also called error function $E(t, f(x))$, is used to measure the performance of a network f on a specific sample x. The L1 loss

$$E(t, f(x)) = |t - f(x)|$$

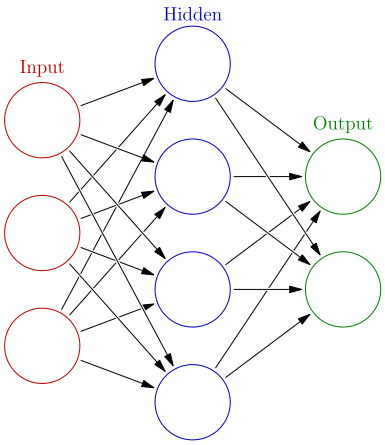

Figure 2.1: Fully Connected Network with three inputs, four hidden and two output neurons. [5]

is one of the simplest error functions which measures the absolute distance between the target t and the prediction $f(x)$. The backpropagation algorithm calculates the gradient δ regarding the weights in all the layers. Every weight is getting updated with a factor η which is the learning rate:

$$w \leftarrow w + \eta * \delta * x$$

In the training process the samples are pushed through the network, the error gets calculated and then the gradient gets propagated back through the network in order to optimize the parameters iteratively.

In order to test the performance of the trained neural network on unseen data, the available data gets split in several disjoint sets like training and test data. The training data set is only used to optimize the learnable parameters, to find feasible values for the hyperparameters like a good learning rate or to decide for the network architecture itself. The amount of data which is needed to optimize a neural network depends mainly on the complexity of the neural network (for instance, amount of trainable parameters) and the complexity of the problem which should be learned. The test data is exclusively used for testing the performance once the entire training process is done. It is quite common to split the training set again in two subsets (training and validation set). The data samples of the validation set are used during the training, for example, to prevent overfitting. The Figure 2.2 visualizes the error of the network during the training using data points from the training and the validation set. The error made on the training data (blue line) is constantly decreasing. Starting at the marked position, the predictions on unseen data from the validation dataset are getting worse. The error (red line) increases which means that the model tries to memorize the training data instead of solving the problem generalized.

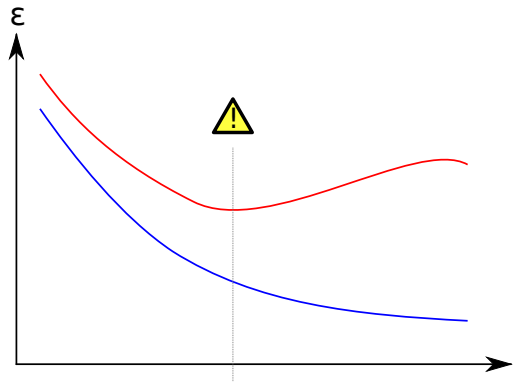

Figure 2.2: Training (blue) and validation (red) error plotted over training steps. Model starts to overfit at the marked point. [13]

2.1 Natural Language Processing

This section provides an overview of common network architectures for natural language processing problems. The recurrent neural network architecture and the related challenges will be presented in Section 2.1.1. In Section 2.1.2 the Transformer Network is discussed. The representation of language data, backpropagation through time and parameter sharing will be also briefly introduced.

Natural language processing is a research direction in artificial intelligence with the approach to use natural language like English as interface between humans and machines. Using language as interface brings challenges for computer systems like the representation of the language, the need to resolve ambiguities and the capture of the actual intent of the users.

There are several ways to feed written language into a neural network. Depending on the task the characters, pieces of words, words or entire sentences/paragraphs are encoded as input. In the following, a naive word representation and the byte-pair-encoding is introduced.

Taking all unique words of a corpus (i.e. Wikipedia crawls) and assigning each unique word an index is forming a word vocabulary. The input of the neural network is now the index of the word instead of the word itself. One major issue of this approach is that the vocabulary usually gets quite huge because subwords or related words have different indices. For instance, *sun* and *sunny* would have a different index in the dictionary. Furthermore, it can happen that words with typos are not in the dictionary at all and, therefore, they have to be assigned to an index which represents an unknown token.

The byte-pair-encoding overcomes these problems by composing words together out of subwords. The vocabulary contains a symbol for every seen character plus merges of symbols. To calculate the merges, the entire corpus gets replaced through symbols. Afterwards, the occurrence of adjacent symbol pairs is counted and the most frequently

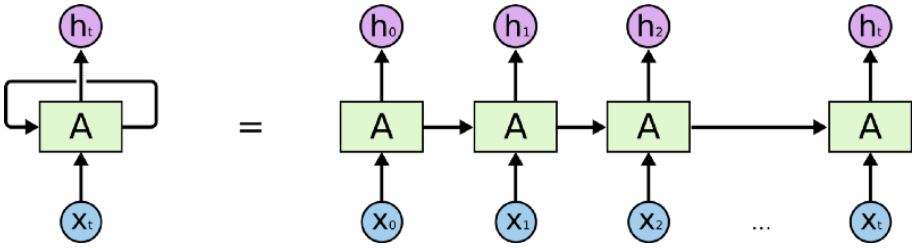

Figure 2.3: In time unfolded Recurrent Neural Network. [19]

seen pair gets replaced through a new symbol. The mapping of the pair and the new symbol is a merge.

2.1.1 Recurrent Neural Networks

Processing natural language implies inherently the necessity of the model to care about the variable input length of the sequence. On the one hand, the data can be padded to the same input length and then passed into a network which is dependent on the length like, for example, a feed forward network. On the other hand, a recurrent neural network can be used to process data of variable length.

The idea of a recurrent neural network is that the sequence $x^{(1)}, ..., x^{(T)}$ is read token by token in T time steps. A further input of the network is the internal state of the network itself. The Figure 2.3 shows two equivalent graphical representations of a recurrent network which maps the input sequence to an output sequence. On the right side the recurrence of the network is unfolded in time, however, the left one still contains the cycle.

The definitions below will be used to formalize an instance of an Elman recurrent network:

- Weight matrix: U, V, W
- Bias: b, c
- Input at time t: $x^{(t)}$
- Hidden state at time t: $h^{(t)}$
- Output at time t: $o^{(t)}$
- Loss at time t: $L^{(t)}$
- Prediction at time t: $\bar{y}^{(t)}$
- Target at time t: $y^{(t)}$

The hidden state $h^{(t)}$ is actually the point where the RNN differs from a feed forward architecture because of the recurrence. The weighted previous hidden state $Wh^{(t-1)}$ together with the the weighed input $Ux^{(t)}$ are getting activated. The output $o^{(t)}$ is a linear

transformation of the hidden state and can be used to calculate, for instance, the discrete prediction $\bar{y}^{(t)}$. The activation functions used here are interchangeable.

- $h^{(t)} = tanh(b + Wh^{(t-1)} + Ux^{(t)})$
- $o^{(t)} = c + Vh^{(t)}$
- $\bar{y}^{(t)} = softmax(o^{(t)})$

The recurrent network is trained with the backpropagation through time algorithm. The computational graph gets unfolded in time and the weights U, V, W are shared which means that each weight matrix is updated with the aggregated and the same value. The overall error of the network is the sum of the errors of all time steps $L^{(t)}$. To update one weight matrix ,the error from one sample is calculated, propagated back and summed up. Every path from the loss to the weight is included in the sum and the weight is updated only once using the calculated sum.

One major issue of RNNs is the exploding and vanishing gradient. Essentially, the problem occurs because the gradient is a product that contains multiple times the same term. Repeatedly multiplying gradients with values smaller or larger than 1.0 will lead to the vanishing or exploding gradients. In the worst case the network is not trainable because of the number under-/overflows or the training is not stable due to the large changes of the weights in each update. The problem can be fixed by using another recurrent architecture like Long Short-Term Memory Network.

In general the natural language problems can be divided into three major classes: variable length input with fixed output, variable length input with same length output and variable length output with variable length. The first two can be trivially implemented with a RNN.

An example for the variable length input with fixed output problem is, for instance, sentiment analysis. A sentence/paragraph is feed token by token into a recurrent network and the last prediction $\bar{y}^{(t)}$ can be interpreted as a semantic summarization. The fixed length output can be used to classify the sentiment.

However, the part of speech tagging problem has a variable length input with the same length as the output. The goal is to assign to every token a grammatical tag like noun or verb. A plain RNN can be trained so that the prediction $\bar{y}^{(t)}$ corresponds to the grammatical tag of the token t.

Translation is a sequence-to-sequence (seq2seq) problem where the source and target sentence of the two languages usually do not have the same length. Figure 2.4 shows an encoder-decoder architecture build out of two recurrent networks. The input sequence is getting condensed into the vector s using the last hidden state $h^{(t)}$ of the encoder RNN. A secend RNN - the decoder - is initialized with the vector s to decode the semantic representation to the target sequence. The sequence gets decoded token by token whereby the already generated tokens are used as next inputs. Since there is no input at state $t = 0$, a special token like <*BOS*> (begin of sentence) is used to start the decoding process. A

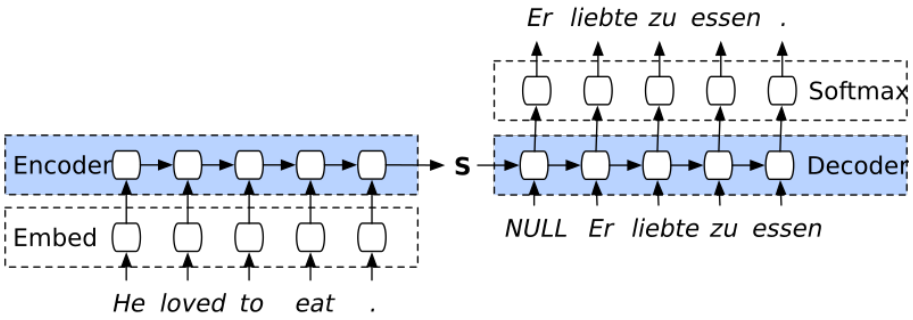

Figure 2.4: Encoder-Decoder model composed out of two RNNs modeling a translation task. [4]

special token like <*EOS*> (end of sentence) is used to indicate the end of the sentence and to abort the decoding process.

2.1.2 Transformer Network

In this chapter the Transformer Network is introduced. The general idea, different dependency types and the building blocks of the architecture are discussed.

The Transformer model is a sequence-to-sequence model which relies fully on attention. It is simpler and more computing efficient due to removed recurrent connections which leads to shorter training times. Dependencies between the source and target sequence are well captured by the model.

2.1.2.1 Dependency types

A linguistic dependency describes the relation between two words. In the following example the word *they* resolves to *two girls. Two girls are in the restaurant. They order the same dish.* It is for many tasks essential to capture these kind of dependencies to perform well. Essentially, there are 3 main types of dependencies in sequence to sequence tasks: dependencies between the source and target sequence, within the source sequence and within the target sequence. Whereas a basic RNN model can capture the dependencies between the source and target sequence reasonable well, the Transformer model captures better the dependencies within the source and the target [27].

2.1.2.2 Position-Encoding

Since the model has no recurrent structure to encode the relative or absolute position of the tokens, this information is injected manually. To the input and output embedding a positional encoding is added. Usually a sine and cosine function with different frequencies are used as a positional encoding.

2.1.2.3 Attention

In this section the idea of the attention mechanism will be introduced. Furthermore, the main building block of the Transformer model, the multi-head attention, will be defined.

A plain RNN applied to the translation task will encode the entire sequence to a vector v and then decode it. The intuition behind the attention mechanism is to provide the decoder with a context vector to focus on certain tokens of the source. Each context vector is a basically a weighted sum of the hidden states of the encoder. The attention weights are learned during the training. Figure 2.5 visualizes that for the current decoding step the token I in the input sequence is more relevant then the others. The weighted sum of the hidden states multiplied by the attention weight represents the context vector.

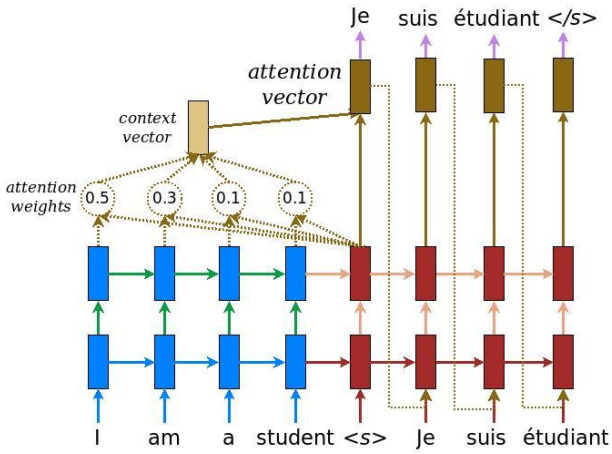

Figure 2.5: RNN with attention to utilize the context vector to focus on specific tokens of the input. [20]

However, the Transformer model is based on a multiplicative attention which is called Scaled Dot-Product Attention. The following equation represents the attention and can be visualized like in Figure 2.6.

$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$ with Query Q, Key K and Values V.

K and V has the dimension d_k which is used to scale the dot-product since it tends to grow with the dimensionality. The mechanics of this attention is the same like in the above described additive attention that certain values V are more relevant regarding the query Q and the current keys K which are comparable to a context. Multiple Scaled Dot-Product Attention blocks are used in parallel, the results are concatenated and then projected through a linear layer to the dimensionality of the value V (Figure 2.6).

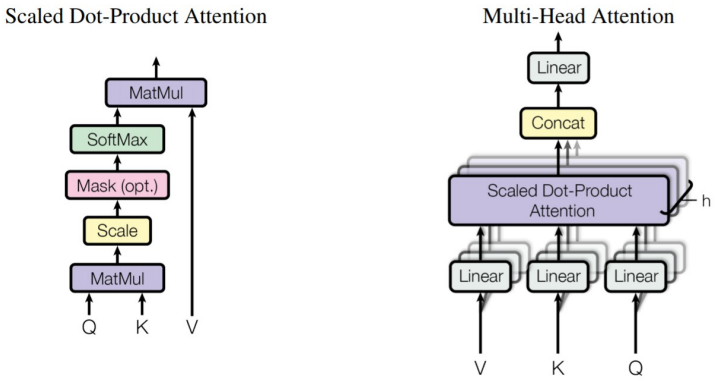

Figure 2.6: Left: The structure of a single Scaled Dot-Product Attention block. Right: Multi-Head Attention block composed out of h Scaled Dot-Product Attention blocks. [27]

2.1.2.4 Encoder & Decoder

The entire architecture can be divided into an encoder and decoder stack. N identical layers are stacked like visualized in Figure 2.7. In the following the focus is put on how the Multi-Head Attention is reused and connected.

The encoding layer is built out of two sub-layers. The first layer includes the Multi-Head Attention, a residual connection and a layer normalization. The second layer is built out of a fully connected network, a residual connection and a layer normalization. In the encoding stack Q, K and V of the Multi-Head Attention block is set to the same value in order to implement a self-attentive behaviour.

The encoding layer is built out of three sub-layers. The first layer includes the Multi-Head Attention, a residual connection and a normalization layer. However, in the decoder the Multi-Head Attention is masked to prevent that the model can attend tokens which will be generated in later time steps of the decoding process. The second sublayer consists of the Multi-Head Attention, a residual connection and a layer normalization. The Multi-Head Attention uses in this layer the output from the encoder as K and V and Q from the layer below. The third layer is built out of a fully connected network, a residual connection and a normalization layer equivalent to the second layer in the encoder.

2.1.2.5 Training & Inference

The training and the inference of the Transformer model is slightly different in terms of how the data is fed into the model. The main difference is that the inference needs several runs to generate the entire sequence.

The model is trained using input sequence $I = I_0, ..., I_n$ and output sequence $O = O_0, ..., O_n$ tuples. The embedding of I is fed into the encoder. A special token to indicate the begin of a sentence is prepended to the sequence O, then embedded and fed into the decoder. The training objective is to predict the sequence O postpended with a special token to

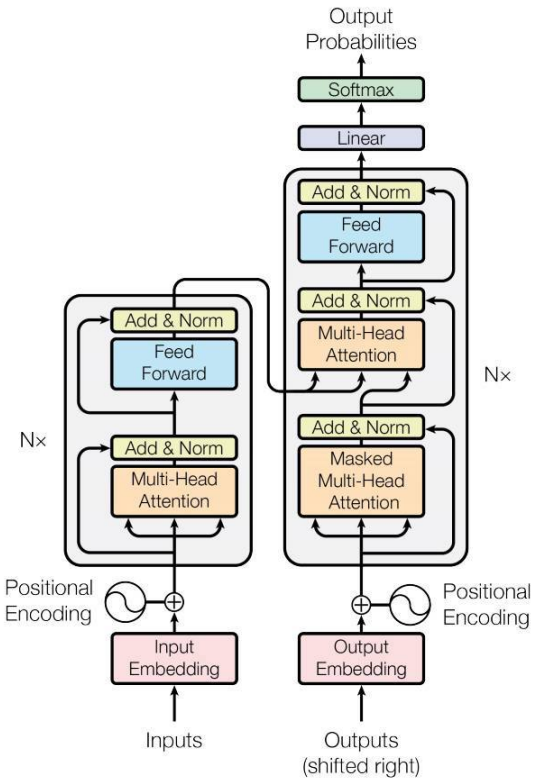

Figure 2.7: Visualization of the entire Transformer model which is composed using multi-head attention as main building block for encoder (left stack) and decoder (right stack). [27]

indicate the end of the sentence. The begin of sentence token <BOS> is basically used to introduce a shift that the first token (begin of sentence) of the input of the decoder maps to O_0. During the training additionally an attention mask is used to prevent leftward information flow in the decoder.

At inference time the sequence I is fed into the encoder and the output gets iteratively generated. The begin of sentence token *<BOS>* is fed into the decoder to predict the first token. The generated token gets attached to the token <BOS> input token to predict the next token. The procedure is repeated until the end of the sentence token *<EOS>* is outputted. This greedy decoding strategy depends on the first predicted token. To lower the chance of decoding a wrong sequence, for example, due to the wrongly predicted first token, algorithms like Beam Search are used. Beam Search keeps while decoding the N most probable sequences. After all sequences are fully decoded, the most probable sequence is taken.

2.2 Computer Vision

In the field of computer vision algorithms are developed to process visual data such as images or videos. In the recent years the approaches evolved from algorithms which depend on hand-crafted features to end-to-end learned neural networks. Some of the models achieve in specific tasks close to human level performances. In the following, the convolution operation and the Convolutional Neural Networks (CNNs) are briefly introduced.

Usually images are represented in a grid-like data structure where every grid entry belongs to a pixel value. While a gray scale image can be represented with only one matrix containing the gray values, an RGB color image is represented with 3 matrices where each matrix represents one color channel. Features of an image like edges can be extracted with a mathematical operations called convolution. They are used to describe the content of the image. In the recent approaches these convolution functions are learned through training.

2.2.1 CNN

In this section the use of Convolutional Neural Networks is motivated and the main building blocks are briefly introduced.

Convolutional Neural Networks are introduced here because plain feed-forward networks have practical limitations in processing visual data. They can also be interpreted as an extension of the time delayed neural networks [29]. The issues like heavy computing complexity, the many needed parameters and shift-variant learned knowledge are approached using convolutional layers, pooling layers and a classification layer.

In the convolutional layer small matrices are learned which are often called kernel or filter. By sliding a two-dimensional filter h over the input f the convolved output G is calculated.

The pixel representation of the input is addressed with the index m and n and j, k denotes the index to the value of the filter.

$$G(m,n) = \sum_j \sum_k h[j,k] * f[m-j, n-k]$$

The filters are learned during training and the parameters of a filter are shared which means that a single filter has only $j * k$ parameters. In comparison to a fully-connected approach, the representation is sparse and invariant to the position. A convolutional layer usually contains multiple filters activated by a ReLu function.

Usually a pooling layer is stacked on top of a convolutional layer. A pooling function summarizes the characteristics of the input in a rectangular neighborhood. For example, a max pooling only outputs the maximum of the input. Because of this layer a local translation and rotation invariance can be introduced and no parameters are added because the function is not learned. If no padding is added to the input, the repeated application of convolutional and pooling layers will reduce the dimensionality of the initial input.

Once a specific dimensionality is reached, a fully-connected layer is applied as the last processing step. Depending on the task, the output is used, for example, to classify the content of the image.

2.2.2 ResNet

The ResNet architecture utilizes residual connections in order to build deeper computer vision models. In this chapter the idea of the approach is presented.

The trend in computer vision before the ResNet publication was to build deeper and deeper models. However, the authors of the ResNet paper have shown that deeper models might lead to worse performance compared to the models with lower capacity. Figure 2.8 is visualizing the result of an experiment where two models are compared. Both models are not overfitting because the training error and the test error is decreasing. However, the performance of the model with less layers still outperforms the other one. In order to be still able to train such deep networks, utility losses are often used to introduce a further learning signal.

The main idea of the ResNet architecture is to introduce an identity shortcut connection. The also called skip connection enables to bypass several convolutional layers. The authors claim that adding several identity layers do not influence the performance of the network. Furthermore, they argue that it is easier for the network to learn an identity function using the residual connection instead of adapting the weights of the kernel to achieve that.

This approach is picked up by many successor papers and gets extended. For instance, instead of using only two paths (convolutional layer path and residual path) the model gets extended through further convolutional paths with different kernel sizes. As a result the network is able to select the fitting kernel size or to skip the layers if they are not needed to solve the task.

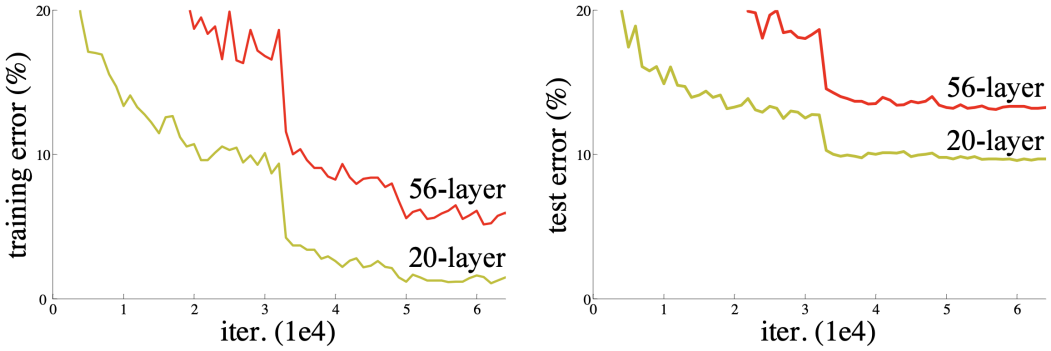

Figure 2.8: Comparison of two networks with different network depth. The accuracy of the deeper model saturates which results in a higher error rate. [7]

3 Related Work

In the survey of Chen et al. [2] an overview of the different dialogue systems and the related architectures is given. They group the approaches in task-oriented and non-task-oriented systems. The task-oriented systems are designed to solve a specific task like to book a flight or to find a restaurant. An approach is to interpret the dialogue as a pipeline of several steps like language understanding, dialogue state tracking, policy learning and natural language generation. Recent work is focusing on end-to-end approaches to avoid limitations like the credit-assignment problem or to get more adaptable to the new domains [35]. The non-task-oriented systems are sub-divided into generative, retrieval-based and hybrid methods. According to their differentiation this work is a non-task-oriented dialogue system with a way to partially incorporate memory.

In the work of Qian et al. [17] the goal is to generate more realistic and natural responses by tying a personality to the dialogue system. The idea is that the user can ask for the gender of the system and it acts according to the given personality. A sub-network predicts if the response should be generated by a general decoder or if some profile information is needed which is used by another decoder. The profile information contains key-value pairs where one tuple gets selected according to the key based on the user input. The value is used to generate a grounded response. One difference in this work is that they assign a personality to the dialogue system instead of having a profile for every single user. The system tries to imitate the behaviour of a real person. However, their dialogue system does not build a profile of the user to generate a personalized response based on this knowledge. That means a user is not able to ask facts about his identity.

In the paper of Ghazvininejad et al. [6] a knowledge-grounded conversation model is introduced. The model relies not only on the conversation history but also on external facts to generate a response. The facts that are available to the system are taken from external sources and are not covered by the conversational corpora. Facts which are relevant for the user input are selected through keyword matching. The conversation as well as the relevant facts are encoded and fed into the decoder. The focus of their network does not lie on task completion and is trained in a multi-task learning fashion. Their objective is comparable to the one in this thesis. Instead of person related facts they query world facts. However, their architecture does not allow to learn from a dialogue and does not store these facts.

The foundation of this thesis is the work of Wolf et al. [31]. Compared to the previously often used architectures, they used a transformer network based approach instead of a recurrent neural network. The network is trained on the persona-chat dataset in order to condition the model to a specific persona. Because of that, the network uses the injected

persona data to generate responses which are related to this data. This approach gives the model itself a personality, however, in this thesis knowledge of the user is injected to modify the dialogue.

Recently before the submission of this thesis the paper [21] from Facebook was published. Their work covers an extensive investigation on how to build open-domain chatbots. In particular, the work focuses on properties of the dialogue system like empathy, personality and maintenance of a consistent persona of the system. All the developed models of their work are based on the Transformer model. Using human evaluation, their model outperforms a comparison baseline model in terms of engagingness and humanness. Their work follows the same research direction like the work presented in this thesis which underlines even more the importance of building personalized dialogue systems.

An architecture with three different components is developed by Tanaka et al. [26] to incorporate external facts. The system consists out of a Reranker, a Fact Retrieval (FR) component and a Memory-augmented Hierarchical Encoder-Decoder (MHRED). The Reranker component selects the most probable response generated by the MHRED and FR and outputs it to the user. The dialogue context together with the encoded facts are in the MHREC used to generate a fact grounded response. The Facts Retrival component selects relevant responses based on a given metric from a database. In comparison to the model of this thesis they are using well prepared datasets to train every component.

In the work of Zhang et al. [34] they used a GPT-2 model which is fine-tuned on Reddit comment chains to generate more relevant and contentful responses. The data is extracted from Reddit through filtering out irrelevant information resulting in a dataset with 147,116,725 dialogues. They observed that the model performs better than an RNN counterpart since it pays more attention to the context. The objective of their work is not to inject external knowledge in order to adapt the dialogue.

Recent research activities show that there is interest in developing more engaging dialogue systems. It is approached through using high-capacity models or injecting additional knowledge into the model. However, the literature research resulted in a fact that there is no relevant or well-known work focusing on building up a representation of the user which is used later while interacting with the dialogue system.

4 Architecture

This chapter provides a general overview of the entire architecture and motivates the usage of the chosen components. The details of the dialogue component are elaborated. Topics like data generation, training, inference and special tokens are covered. Finally, the personalization component and the related functionalities like memory selection are introduced.

4.1 Overview

The developed architecture consists of two main components:

- The **dialogue component** generates personalized responses based on user input and facts of previous dialogues (memory).
- The **personalization component** keeps track of the facts gathered during conversations and returns a memory that is based on the recognized person.

The dialogue and the personalization components are not jointly trained, however, at the inference time the models run at the same time.

To build the dialogue system a Transformer Model is used because it performs well in seq-to-seq tasks and can handle dependencies well. Because of the small amount of training data the approach is to fine-tune a model which already learned the concepts of language like, for instance, the recently published models BERT and GPT-2. The BERT model is not autoregressive in contrast to the GPT-2 model. Since a dialogue system is based on language generation, the GPT-2 model is better qualified for this task.

The two neural networks which are used in the personalization component do several steps in order to transform a input image into a face embedding. Because of the assumption that the images from a camera are not perfectly aligned, a pre-trained [30] multi-task network[33] is used to do the pre-processing steps. The processed image is embedded using a pre-trained Inception ResNet[24] as a feature extractor.

The two components follow the same execution order for every conversation turn. In the following the steps are listed how a personalized response is generated:

1. Camera image gets preprocessed and embedded.
2. User profile gets loaded according to the embedding.
3. User input is expected.

4. Response generation based on input and user profile (memory).
5. Update user memory if needed.
6. Print response to the user.

A visual overview how a personalized response is generated is shown in Figure 4.1. The Figure basically visualizes the step 1 - 4 in the above mentioned list. The different steps are elaborated in Chapter 4.2 and 4.3 in detail.

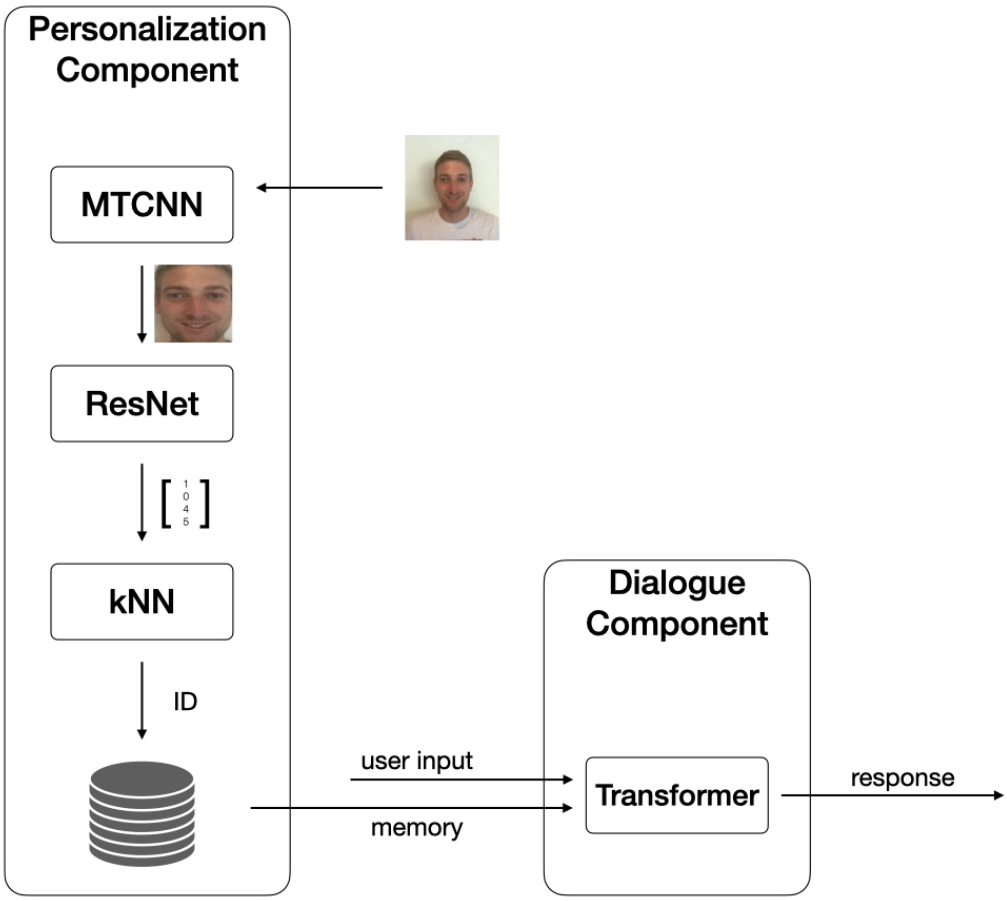

Figure 4.1: The dialogue component needs two inputs in order to generate a personalized response: the user input and the memory which is selected according to an image.

4.1.1 GPT2

The GPT-2 model developed by OpenAI [18] is essentially a language model and predicts the next word given a context. It is based on the decoder stack of the transformer model and was trained on a 40GB text dataset.

The authors of the GPT-2 model are targeting the lack of task-independent models in the natural language understanding domain. They claim that the currently developed approaches are "narrow experts rather than generalists"[18]. Furthermore, they identified that there will be a limit in scaling the datasets in order to be able to solve certain natural language understanding tasks.

The GPT-2 model is composed out of the Transformer decoder blocks ranging from 12 up to 48 layers. The Transformer architecture is only slightly changed. In each sub-block the layer normalization was moved to the input of the block. Furthermore, they added another layer normalization at the final attention block, increased the context size from 512 to 1024 tokens and scaled the initialization of the residual layers. For GPT, the successor model of GPT-2, exactly the same decoder block of the Transformer model is used.

The improvements of GPT-2 are focusing more on how to train the model itself rather than on architectural changes. The general approach is to split the training into two stages: unsupervised pre-training using the language modeling objective and the supervised fine-tuning to optimize the downstream objectives. The following likelihood is maximized and represents the language modeling objective:

$$L_1(U) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \theta)$$

U represents the set of all tokens in the dictionary, k the context window and θ are the model parameters. The conditional probability P is modeled by the transformer decoder where h_n is the output of the last decoder block and W_e is the token embedding matrix.

$$P(u_{-k}, \ldots, u_{-1}) = softmax(h_n W_e^T)$$

Text data out of multiple domains is used for the pre-training, although usually the dataset is carefully designed with samples of the same domain. The intention is to present many different aspects of natural language to the model while training to achieve a good adaptability to different tasks. A new dataset called WebText was created by the authors of GPT-2. The dataset contains web crawls of pages among which the link to the page was posted on Reddit. Karma points of the posts were taken as a relevance metric to filter out web pages with less then 3 points. After duplicate removal and heuristic cleanup the collection contains roughly 8 million documents which results in 40 GB of text.

For the supervised fine-tuning an annotated data set C is required where each sequence $x^1, \ldots, x^m$ is assigned to a target y. The pre-trained network is used to get the activation of the last decoder block which is then passed into the linear output layer with the weights W_y.

$$P(y | x^1, \ldots, x^m) = softmax(h_n W_y)$$

The loss of the fine-tuning is defined as following:

$$L_2(C) = \sum_{(x,y)} \log P(y | x^1, \ldots, x^m)$$

The authors of GPT-2 finally proposed to use a weighted loss $L = L_2(C) + \lambda L_1(C)$ since it turned out to be beneficial to use language modeling as an auxiliary objective.

4.2 Dialogue System

This section covers the implemented algorithm to generate training data, the model which is used as the dialogue component to respond to user queries, the training and inference of the model.

4.2.1 Data Generation

The objective of the model is to generate responses which should be grounded on the facts in the memory. To fulfill this requirement the samples in the training data set need to have at least the properties below:

- facts which represents the memory
- chat history
- response to be generated

Due to the special requirements of the samples, the data can not be easily crawled or extracted from public sources like movie subtitles or Reddit. In the most cases it is not easily possible to build up a memory of a user using crawled data. Furthermore it was not intended to use humans (Crowdworkers) for the data generation because it is often exhausting and expensive. Therefore an algorithm is used to generate artificial training samples. A formal grammar defines dialogue snippets and rules how to generate samples for the training and the test dataset. The data is structured similar to the samples in the persona chat dataset [31].

Listing 4.1 represents one sample where the dialogue has one turn. The *memory* key of the JSON includes an array of facts within the memory. These facts refer to the entire dialogue. The array referenced by the *utturances* key represents the turns of the dialogue. One turn entry has two more keys: the *history* key and response *candidates* key. The dialogue history is represented by an alternating sequence of user inputs utterances and utterances generated by the dialogue system. The *history* key contains this information. Furthermore, the *candidates* key includes N responses. Only one response is the intended response since the history and the memory is taken into account. This correct response is always the last element in the array. The $N - 1$ additional utterances serve as distractor responses and are random samples from other dialogues. The data generation algorithm ensures that the random samples are always distractors which means that they do not match the context of the user input. The multiple-choice head of the model is trained with the N candidates. However, only the correct response is used to train the language-modeling head.

The data point in listing 4.1 assumes that the user wrote *Hello* to the dialogue system. Since the dialogue system knows the name of the user (for example because of a previous dialogue) the system should greet the user with his name. The response candidate in line 11 serves as a distractor and is not the intended response because the dialogue system already knows the name.

Listing 4.1: Sample instance which is used to train the model.

```
{
  "memory": [
    "name: Peter",
  ],
  "utterances": [
    {
      "history": [
        "Hello"
      ],
      "candidates": [
        "<ask_name> <sep> Good afternoon, I haven't met you so far, what's
            your name?",
        "<greet> Peter <sep> Good afternoon, Peter"
      ]
    }
  ]
}
```

Listing 4.2: Excerpt of a two turn dialogue.

```
...,
"utterances": [
  {
    "history": [
      "Good evening"
    ],
    "candidates": [
      "<greet> Miguel <sep> Hi, Miguel",
      "<ask_name> <sep> Good afternoon, what's your name?"
    ]
  },
  {
    "history": [
      "Good evening",
      "<ask_name> <sep> Good afternoon, what's your name?",
      "It's Marcella"
    ],
    "candidates": [
      "<ask_name> <sep> Hi, I haven't met you so far, what's your name?",
      "<new_identity> Marcella <sep> Hello, Marcella"
    ]
  }
]
```

Listing 4.2 shows a training sample with a two turn dialogue. The intended network response is appended to the history of the next turn. For the training every sub-dialogue is used as a training sample. This means that every turn instance in the *utterances* key is used at least once in the training. That guarantees that the history of the conversation is also considered for the response generation.

The generated dataset is encoded with a byte-level Byte-Pair-Encoding tokenizer of Huggingfaces[1]. The tokenizer is pretrained and has a dictionary size of 50257 elements. Special tokens (<bos>, <eos>, <pad>, <sep>) are added to the dictionary to indicate the begin of sequence, end of sequence, to pad the sequence and to separate parts within a sequence. Furthermore, task-depended special tokens (<system>, <human>, <new_identity>, <ensure_identity>, <use_fact>, <new_fact>, <ask_name>, <greet>, <help>) are added. The usage of them is discussed in Section 4.2.4.

4.2.2 Training

The personalized dialogue system is built on top of the TransferTransfo[31] architecture introduced by Huggingfaces which is respectively a pre-trained GPT-2[18] architecture. The system consists of a GPT-model with different heads for training and inference. For the training two heads are attached to the GPT-2 backbone: a language modeling and a multiple-choice classification head. However, for the inference only the language modeling head is used. Hereafter the training procedure and the reason for using different heads is discussed.

The 12 pre-trained attention block layers of the GPT-2 backbone are trained or fine-tuned to work as a dialogue system. The language modeling head H_{LM} is used to produce a reply based on the given input sequence. The input is passed through the attention block layers and the hidden-states of the last layer h_n are passed into a fully-connected network f_{FC1}.

$$H_{LM} = f_{FC1}(h_n)$$

The fully-connected layer calculates the prediction scores for each vocabulary token, therefore the output layer size must be equal to the dictionary size. The cross entropy loss CE_{LM} combines the log-softmax function and the negative log likelihood loss and is used as optimization criteria:

$$CE_{LM}(target) = -\log \frac{\exp H_{LM}[target]}{\sum_j \exp H_{LM}[j]}$$

Taking the facts in the memory into account a second head is used to enforce that the network is using these facts. This utility objective introduces a further feedback to generate memory-grounded responses. For example, if the memory contains the fact that the user likes fish and chips, this information should be used to generate a personalized reply on the query *"I'm hungry and don't know what to eat."*. To achieve this the multiple choice

[1] https://github.com/huggingface/tokenizers

head needs to select the correct reply from the different *candidates* in the training data. Regarding the example above a candidate list could look like this: *"I'm not hungry."*, *"I don't know your favorite dish."*, *"You told me that you like fish and chips, how about that?"*. Based on the memory content the last candidate is the best-suited reply. The other distractor sentences are not intended replies since they may not consider the current memory state or are out of the context. The multiple choice samples are generated to match exactly this condition that only one candidate is the correct one. This approach is described in detail in Section 4.2.1. The multiple choice head H_{MC} is composed out of a dropout layer $f_{Dropout}$ and a fully-connected layer f_{FC}. The last hidden state h_n is used as an input for the classification head:

$$H_{MC} = f_{FC2}(f_{Dropout}(h_n))$$

The multiple choice head is optimized as well using a cross entropy loss. The loss takes into account if the model selects the correct candidate:

$$CE_{MC}(target) = -\log \frac{\exp H_{MC}[target]}{\sum_j \exp H_{MC}[j]}$$

The overall loss L is the weighted sum of the multiple-choice and the language modeling loss:

$$L = \lambda_1 CE_{MC} + \lambda_2 CE_{LM}$$

An AdamW optimizer with fixed weight decay is used in addition to a linear decreasing learning rate while training.

4.2.3 Inference

For the inference the model is slightly changed. Since the dialogue system is only intended to generate a response, the multiple choice head is discarded. The weights of the GPT-2 base model and the language modeling head are initialized with the fine-tuned and correlating weights. For the response generation only the current user input, the memory and the history are fed into the model.

Beam-search is strongly sensitive to the output length [32] and in generation tasks the beam-search distribution is different to human-written texts [8]. Because of that a sampling method is used for decoding. In recent work the top-k and Nucleus sampling [9] is introduced and the authors have demonstrated good results. The implementation of *Huggingface* uses these sampling methods and is applied to the dialogue model. The logit output of the language modeling layer H_{LM} is scaled with factor t and passed into the f_{filter} function together with the two hyperparameters top_p and top_k.

$$N = softmax(f_{filter}(\frac{H_{LM}}{t}, top_p, top_k))$$

The top_k hyperparameter changes the behaviour of f_{filter} in the following way. The intention is that only the top_k most probable tokens should be considered for sampling. To achieve this behaviour the logit values of the other tokens are set to minus infinity.

Similarly, the top_p hyperparameter defines a threshold for a cumulative filtering of the sorted logit values. The function f_{filter} keeps only the tokens where the cumulative logit values are equal or greater than the threshold top_p. In practice the tokens with the lowest logit values are filtered by setting the values of them to minus infinity. The softmax function is applied to the result of f_{filter}. The consequence is that tokens with a minus infinity value are assigned to a very small probability. Finally, the next token is obtained by sampling it out of the distribution N.

To generate the entire sequence, the token is appended to the already generated tokens. This sequence is iteratively extended and passed in the network. The decoding process will be stopped once the *<EOS>* special token is sampled. To avoid infinity loops while generating the sequence the process is also stopped if a specific sequence length is reached.

4.2.4 Input and Special Tokens

Task-dependent special tokens are introduced and added to the tokenizer. The use of the extra added tokens is discussed in the following.

The input of the network is a string representation of the memory, the history and the reply if the network gets trained. The different parts are separated through special tokens to support the network to differentiate the different parts of the input. The content of the memory is represented by the concatenated facts and is in every sample between the <BOS> and the first <human> special token located. The memory of listing 4.3 and 4.4 contains one and two facts in the memory respectively. Alternating the input of the user and the system, response is concatenated to the memory. The input of the user is between the <human> and the <system> special token. The already given responses of the dialogue system are between the <system> and the <human> tokens. The string between the <system> and the <EOS> token is the actual target reply of the training sample. The sample in listing 4.3 contains only the user input and the target response. However, in listing 4.4 the response *"i don't know your favorite dish"* should be generated based on the first turn and the user input *"i'm hungry"*.

Listing 4.3: Representation of a *one turn* dialogue training sample before tokenization.

```
<BOS> name: lukas <human> hi <system> hi lukas <EOS>
```

Listing 4.4: Representation of a *two turn* dialogue training sample before tokenization.

```
<BOS> name: lukas, my computer is in the office <human> hi <system> hi lukas
   <human> i'm hungry <system> i don't know your favorite dish <EOS>
```

At inference time the input does not contain the target. Once the user enters a query, it gets encoded and appended to the history. The newly created sequence is used to predict the response. The listing 4.5 shows an input where the user just greeted the dialogue system. The system is intended to greet the user with his name.

Listing 4.5: Representation of an input at inference time
sample before tokenization.

```
<BOS> name: lukas <human> hi <system>
```

The introduced way to model the history brings some limitations with it. Currently only alternating dialogues can be modelled. Some logic would be needed to terminate the user input to indicate that the model should generate a response. Furthermore, the training data should also contain such samples. Additionally, the dialogue system can not initiate a dialogue. The system takes a user input and generates a proper reply. As well here, one would have to augment the dataset and implement some logic to decide when to generate an output without an explicit trigger by the user.

The final network input $S = S_0, \ldots, S_n$ is the sum of the word embedding ϕ_{word}, the positional encoding ϕ_{pos} and the segment embedding ϕ_{seg}.

$$S_i = \phi_{word}(token_i) + \phi_{pos}(i) + \phi_{seg}(segment_i)$$

Figure 4.2 visualizes a representation of an input like S. The topmost bar represents the ϕ_{seg} embedding which is added to every token. In the middle the representation of the word prices ϕ_{word} are shown. The bottom-most bar represents the positional encoding ϕ_{pos}.

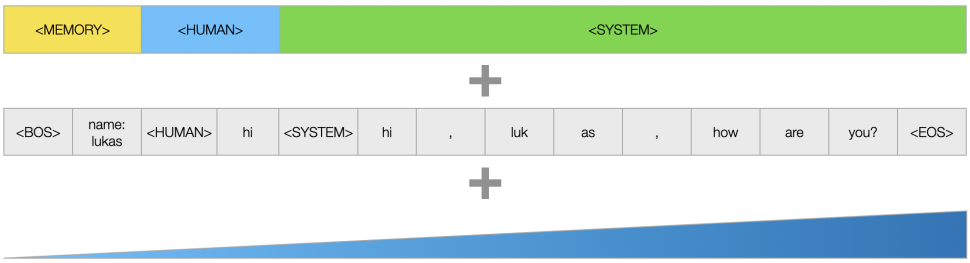

Figure 4.2: Visual representation of the three embeddings which form the network input S

Both the word embedding ϕ_{word} and the positional encoding ϕ_{pos} are applied like proposed in the GPT-2 paper using Byte-Pair encoding and the sin/cos functions. The segment embedding ϕ_{seg} is introduced to support the network to differentiate the input parts (memory, user input, system output). Depending on the part, the related special token (<memory>, <human> or <system>) is encoded and the embedding is added to the other embedding. This information injects to every token the information to which segment the token belongs to. Additionally the special tokens itself are marking the beginning and the end of the sequence like in Figure 4.2 visualized.

4.2.5 Memory

The content of the memory is always bound to a specific person. It means that all facts in the memory refer to the person which is currently chatting with the dialogue system. The

memory content is represented in a symbolic way and looks, for example, like the following example: *"name: lukas, computer is in the living room, i like apples"*. Because of the textual representation arbitrary information can be associated with the user and potentially used for the response generation. There is as well the option to inject information from external sources.

To be able to read from or to write to the memory, additional special tokens are used. Furthermore, a particular output format of the network is expected. The listing 4.6 shows that to the actual user response a special token and optional parameters are prepended and separated by the <SEP> special token.

Listing 4.6: Output scheme where an action special token and parameters are added to the actual response.

```
1 <ACTION SPECIAL TOKEN> PARAMETERS <SEP> RESPONSE TO THE USER
```

For example, if the user introduces himself like *"hello, my name is peter"*, the expected output of the network looks like this: *"<new_identity>peter<SEP>hi, peter"*. The special token *<new_identity>* is an indicator to the dialogue system whether the following sequence up to the *<SEP>* token should be added to the memory. The sequence after the <SEP> token is finally handed back to the user as the reply. If the user enters a query which involves a lookup in the memory, the network also prints a special token including a parameter. In this case the parameter is only used to support the training. Experiments have shown that the network uses the parameter as a hint to find the relevant fact in the memory. Besides that, the generated actions may also be used by other components, for example, to call APIs.

The Table 4.1 lists all used special tokens which represent an action, the access method on the memory and the parameters.

special token	read or write	parameter
<greet>	read	name
<ask_name>	-	-
<new_identity>	write	name
<merge_identity>	write	name
<ensure_identity>	-	-
<use_fact>	read	fact
<new_fact>	write	fact
<help>	-	-

Table 4.1: Summary of the special tokens which are added to the dictionary. The special tokens might indicate an access to the memory (action) with a specific parameter.

4.3 Personalization

The personalization component basically implements the functionality to manipulate the memory based on the camera input. This section gives a general overview and introduces the developed architecture.

4.3.1 Overview

The intent is to use the entire system in the kitchen robot ARMAR. Therefore, it is necessary to detect the person which is at the moment interacting with the robot. To be able to extend the system to achieve this goal, the approach is to select the best matching memory based on the camera input. It allows consequently for the system to recognise different people and use the associated memory to adapt the dialogue.

The entire personalization component consists out of three main modules like already visualized in Figure 4.1:

- **Multi-task CNN (MTCNN)**: Automatic face detection and image pre-processing.
- **ResNet**: Network to compute face embedding.
- **k-nearest neighbors classifier**: Selection of best memory profile.

In order to query the user profile, create and update the profiles and to be able to add facts to the memory, the above mentioned modules are executed sequentially. A new camera snapshot is taken in a cyclical interval and saved to a specific directory. The new image is passed every turn into the pipeline to get the correct memory corresponding to the latest image.

4.3.2 Architecture

In this section the pipelines steps are explained in detail. Finally, the operations are introduced which are realized using the processing pipeline.

4.3.2.1 MTCNN

The multi-task CNN [33] is the first step in the processing pipeline. The input of the network is the camera image which might have variable dimensions. The model detects the faces on the image and returns the cropped face jointly with a detection probability. An image pyramid is built and passed into the network. That means that the image is resized to different sizes to be invariant to different face sizes on the image. It is an easy approach compared to modifying the network architecture to achieve scale invariance. The network itself consists out of P-Net, R-Net and an O-Net.

The P-Net is a fully convolutional network and serves as a proposal network. The network detects all faces on the image which are bigger than 20x20 pixels. Face candidate bounding boxes are generated and roughly calibrated.

The R-Net is used to filter the majority of the generated boxes. The network also performs a calibration and a bounding box regression.

The O-Net network again refines and outputs the left over candidates. Furthermore, overlapping boxes are merged. The network generates additional facial landmarks to describe the face more detailed.

For the further processing only the candidate with the highest probability is kept. The final step is to crop and resize the face to a 160x160 pixel output. The Figure 4.3 shows a sample image input and output of the network. The input was not aligned before the processing step.

Figure 4.3: On the left a sample input image of the multi-stage convolutional neural network is shown. On the right an automatically aligned, scaled and cropped image produced by the neural network is visualized.

4.3.2.2 ResNet

The output image of the MTCNN is used by the InceptionResnetV1 [25] to compute a face embedding. The network is trained originally on the VGGface2 [1] dataset to classify persons. In order to get an abstract representation of the face, the last linear layer for classification is discarded. Consequently the network generates a 512 dimensional vector representing the face. Experiments have shown that the distance is small between two embeddings of similar looking faces. In the experiment multiple photos of a person were embedded and visualized in a 3-dimensional space. In general, clusters are formed by faces of the same person. Figure 4.4 is a three-dimensional representation of the 512 dimensional face embeddings. The dimensionality is reduced using a principal component analysis. The four face embeddings in the front are from the same person. They are well separated from the two other clusters of two additional person embeddings.

4.3.2.3 k-NN

To retrieve the correct memory profile according to the face, the spacial proximity of the same identities is a precondition. Since this is given, a k-nearest neighbors algorithm

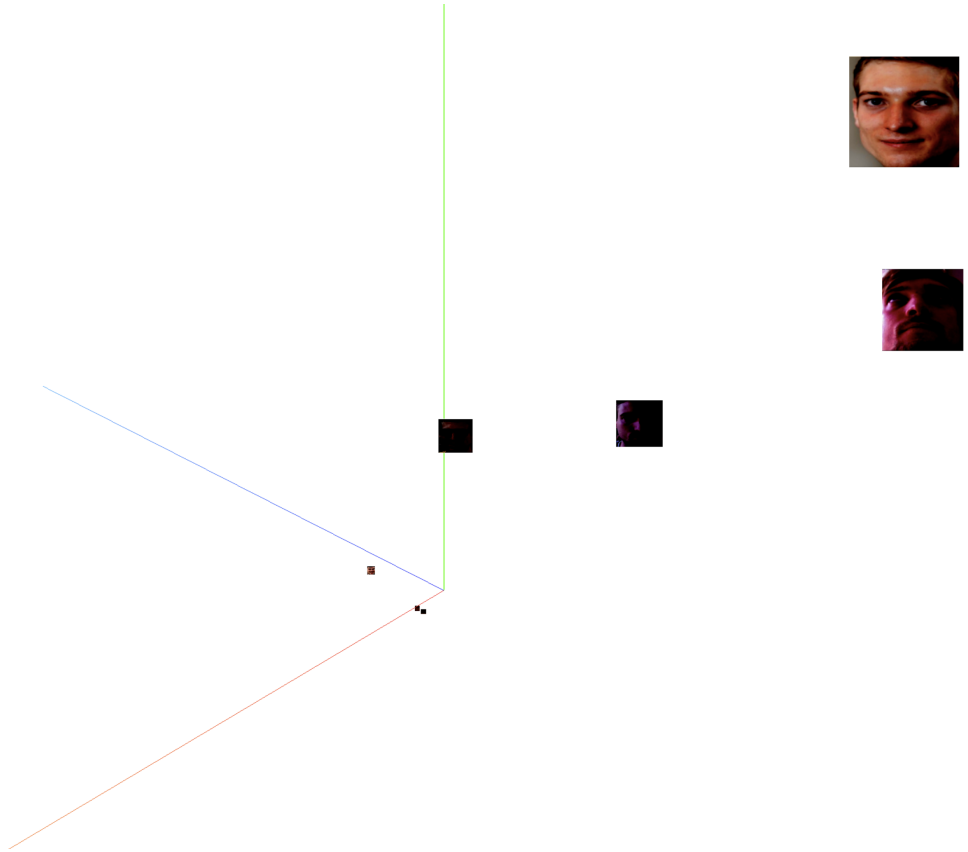

Figure 4.4: Three dimensional representation of the face embeddings. The size and the position of the images correlate with the spacial position in the coordinate system. The four faces in the front are from the same person and cluster well. In the background there are two more clusters of other persons.

can be used to retrieve the memory profile. This approach is chosen since the algorithm does not need exhaustive training and still gets better over time with more samples. In order to decide which memory should be selected the three closest data points are used to determine the profile. The average distance to the next samples is interpreted as the confidence level.

4.3.2.4 Operations

The above introduced pipeline is used to provide the following operations in order to interact with the memory:

- Query user profile
- Create and merge user profiles
- Add facts to user profile

In general, the special token emitted by the dialogue networks indicate which operation should be executed.

Query user profile The query user profile selects the best matching profile based on the image input. The processing pipeline is used to obtain the reference to the actual memory. The confidence level of the k-NN classifier is attached to the memory profile. Thus, the dialogue component is able to adapt the dialogue.

Create and merge user profiles New profiles are added by inserting the face embedding into the k-NN data structure. Then the name is associated with the new data point. This routine is triggered every time when the dialogue network outputs the special token *<new_identity>*. In this case the first parameter is used for the name. To avoid the creation of ghost profiles, the dialogue network ensures with whom it is interacting through asking if the assumed name is correct. If the user agrees, the *<merge_identity>* special token is triggered to link the additional face embedding to the already existing profile. Otherwise, if the user corrects the dialogue system, the <new_identity> special token is generated to introduce an empty profile.

Add facts to user profile The dialogue network is conditioned to always output the *<new_fact>* special token if it is likely that the user input is relevant for the later conversation. For instance, if the user tells what food he likes or where the objects are placed, the input utterance is saved. To extend this behaviour to further scenarios, the training data has to be extended. Before the network assigns the fact to a user profile, it ensures that the confidence is high that it is the correct profile. Since merged profiles only keep the reference to the initial created profile, the reference has to be resolved while adding new facts to the memory.

5 Experimental Setup

This chapter gives a brief overview of the used frameworks. Furthermore, the generated training and evaluation dataset is introduced.

5.1 Frameworks

In the last years TensorFlow [12] and PyTorch [15] have distilled out as the major frameworks for building neural networks. PyTorch is an open source machine learning framework introduced by Facebook. Since PyTorch supports dynamic computational graphs, it is often used in the language research community. The developed models in this thesis are based on PyTorch.

The HuggingFace[1] project provides several state of the art models for natural language understanding and natural language generation. The models have an abstract interface which works with TensorFlow as well as with PyTorch. Therefore, the weights of the models are interchangeable and usable for the both frameworks.

The tool Weights&Biases[2] is used to document and visualize the training runs. There is only a little modification in the code needed in order to get started. The tool automatically collects metadata like the argument parameters, GPU utilization and the TensorBoard logs. The gathered data is available online after the training run. The main benefit is that the training runs are documented well in the different states of the development process. It makes it easier to follow the changes of the implementation of the model and thus the performance of the model.

Scikit-learn [16] is a machine learning library with a big amount of various models. The focus is not put on deep learning approaches like neural networks but more on traditional ones. The framework is used since it provides implementations for the k-nearest neighbour algorithm.

5.2 Dataset

The dialogue network is trained on the generated dataset which is presented here. Since the objective of the network is to generate a personalized greeting and to gather simple facts, the dataset needs to represent samples of such dialogues to archive this behaviour.

[1] https://huggingface.co/transformers/
[2] https://www.wandb.com

Scenario	# Samples
Greeting	20 000
Query name	10 000
Query memory	5 000
Ask for help	5 000
Insert food preferences	5 000
Query food preferences	12 500
Insert location of object	7 500
Query location of object	12 500

Table 5.1: Dataset broken down into scenarios including the number of each samples.

Confidence / Greeting	with name	without name
low	<new_identity>	<ask_name>
mid	<new_identity>	<ensure_identity>
high	<merge_identity>	<greet>

Table 5.2: Triggered special tokens based on the user input and the confidence of the person identification system. The special token also corresponds to a natural language response.

The training dataset contains in total 92 500 data points and covers the scenarios which are shown in Table 5.1.

The *Greeting* scenario covers the different ways how a user can greet the dialogue system. Based on the confidence of the person identification, the dialogue flow is different. If the personalization component is sure about a person, the dialogue system greets straight away the person with the stored name. Otherwise, a question whether the assumption is right or if the user can tell his name is more appropriate. Table 5.2 shows the different cases how the system is intended to behave based on the confidence of the person identification and on the user input. If the user is not recognized (conficence: low) and a greeting without a name is entered (*"hello"*), two turns are needed. The first turn models the <ask_name> special token and a corresponding natural language reply (*"hi, I haven't met you so far. what's your name?"*) to ask for the name. In the second turn a user input with his name is expected jointly with the reply of the dialogue system.

The *Query name* scenario covers cases when the user is asking for his own name. The dialogue system answers with the name stored in memory. *Query memory* needs the network to read from the memory, however, all the facts should be returned. *Ask for help* includes samples where the user tries to figure out what capabilities the dialogue system has. The response is generated with no reference to the memory content. *Insert food preferences* and *Query food preferences* are containing samples to store and retrieve food preferences of the user. The user is also able to teach the system where household objects are placed. The samples are covered by *Insert location of object* and *Query location of object*.

The validation dataset contains 2714 samples and is generated with different variables. Person names, the places of the objects, the objects itself and the favorite dishes are different compared to the training set. The distribution of the scenarios in the validation dataset is the same as in the training dataset.

The memory is constantly growing during inference time since facts are added but not removed. In order to guarantee that the model generates proper responses with a memory that contains many facts, additional random entries are added. Between one and ten facts are randomly added to the memory containing the fact which is needed for the training. The entire memory is additionally shuffled to avoid same ordering in the memory.

6 Evaluation

The chosen hyperparameters of the dialogue network are introduced in this chapter. Next, the evaluation metrics are defined and the training results of the network are presented.

6.1 Hyperparameters

The best results presented in the following are achieved using the set of hyperparameters which are listed here. The hyperparameters are chosen based on the evaluation of the trained model.

- **Number of epochs:** The model is trained 1 epoch on the above described dataset.
- **Optimizer:** The AdamW optimizer [11] is used for training.
- **Learning rate:** The learning rate is set to 0.0000625 and linear decreased down to 0. The learning rate of AdamW is initialized at every epoch.
- **Gradient accumulation:** The gradient of 4 steps is accumulated before the weights are updated.
- **Gradient clipping:** The gradient is clipped to 1 if it is larger than 1.
- **Language modeling loss:** The weight of the language modeling loss is set to 1.5.
- **Multiple choice loss:** The weight of the multiple choice loss is set to 1.
- **Number of distractors:** The multiple choice head selects out of 10 distractor options the correct response.
- **History:** A history of the last two utterances are used while training.

The transformer model related hyperparameters like, for instance, the 10% dropout are not changed[1].

[1]Hyperparameters of the transformer can be found here: `https://huggingface.co/transformers/model_doc/gpt2.html`

6.2 Evaluation Methodology

The fully-automated evaluation of dialogue systems is still an open research topic [3]. Especially challenging to evaluate are conversational agents since they open domain conversations and aren't designed to fulfill a specific task. Because of that it is not obvious what characteristics of the conversation should be measured [3].

The developed dialogue system combines attributes of a task-oriented and conversational agent. The domain is restricted to a kitchen scenario but the conversation follows no concrete structure. Nevertheless, the intention is to gather information of the user which is connected to the introduced task dependent on special tokens. Since there is no public dataset similar to the generated dataset including memories, the neural network cannot be tested on other data. It is also not possible to evaluate the model against a public baseline since the objective is not a common one.

To still obtain an impression on how well the model works, this work is evaluated on three different levels.

- **Human evaluation:** Through human evaluation the natural language response is evaluated.
- **Automatic evaluation:** The automatic evaluation measures if the correct actions are triggered.
- **Attention visualization:** The internals of the model are visualized to understand if the memory is used for the response generation.

The focus of the evaluation is put on the dialogue component since the overall system performance heavily depends on it. In the following chapters the results are presented.

6.3 Results

In this section the performance of the trained dialogue network is presented. The evaluation of the model is divided into human and automatic evaluation followed by the visualization of the attention mechanism.

6.3.1 Human Evaluation

Many of the automatic evaluation approaches such as word overlap-based metrics e.g. BLEU [14] do not correlate with human judgments [10]. Because of that in this work the language capabilities of the model are graded manually. The evaluation was realized in a lab-like environment where the system is prepared before the user interacts with it. The sessions were done remotely using screen-sharing while having an audio call. The evaluation is divided into two parts, the system got reset after each part. Firstly, the user got a list of tasks which he needed to perform. The intention is to guide the user that he gets familiar with the system. In the second part the user was able to freely chat with the

Metric	ø	CS	non CS
naturalness	3.2	3.6	2.8
logic	2.4	2.8	2.0
correctness	2.6	3.0	2.2
consistency	2.5	3.0	2.0
variety	3.2	3.4	3.0

Table 6.1: Human evaluation results after one chat session. Average rating of all testers, only computer science testers and the group with no computer science background.

system. The reason of the evaluation, an overview of the capabilities of the model and the evaluation metrics were presented to the user before the actual test was started. After the testing the users were asked to grade the system based on predefined metrics.

The model was evaluated by 10 persons where half of the participants have no computer science background. The testers are intended to give scores from 1-4 for naturalness, logic, correctness, consistency, and variety. The score of 4 is interpreted as a well performing system. The average rating of every metric is presented in Table 6.1. The ratings are also differentiated based on the computer science background of the testers.

The evaluation has shown that in general the ratings are lower if the tester has no direct relation to computer science. These testers are also referring to already existing speech assistants like Amazon Alexa.

6.3.2 Automatic Evaluation

This part of the evaluation assumes that the triggered action correlates with a meaningful natural language output. If the correct action is triggered based on a certain input, the response will be counted as a correct one. In contrast to this evaluation method plain goal-oriented dialogue systems are often evaluated based on a task-success rate or on a dialogue efficiency [10].

The evaluation focuses on three different aspects. Firstly, the general capabilities are evaluated to cover the defined scenarios. Secondly, the focus is put on reading from the memory. Finally, the system is evaluated on how the number of facts in the memory affect the performance.

6.3.2.1 General Performance

The general performance is evaluated by 280 samples with 25 positive and 10 negative samples covering each of the 8 scenarios. The positive samples contain varying formulations to trigger the intended response. The negative input samples use formulations which should not generate a response for the related scenario. An example for a negative sample in the greeting scenario is, for instance, the input *"i like flowers"*. These samples are used to

Scenario	Accuracy	F1
Greeting and introduction	80 %	86 %
Query name	97 %	98 %
Insert food preferences	94 %	96%
Query food preferences	86 %	89 %
Insert location of object	89 %	93 %
Query location of object	89 %	91 %
Query content of memory	71 %	75 %
Ask for help	83 %	86 %

Table 6.2: Accuracy and F1 score broken down to the scenarios.

explicitly test the behaviour with not intended input. The memory contains between zero and two random facts and the fact related to the scenario if needed. Based on this setup the accuracy $ACC = \frac{\sum TP + \sum TN}{N}$ and the $F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$ score[2] is in Table 6.2 reported. The error analysis of the wrong predictions resulted in two insights. The network relies on punctuation characters to determine what response should be generated. Questions without a question mark were frequently misunderstood by the model. Furthermore, the network generates more correct responses if the input of the network is an entire sentence. A single word reply like *"Peter"* to the question of the network *"hi, I haven't met you so far. What's your name?"* generates an incorrect action and reply.

6.3.2.2 Memory Read Performance

The memory read performance is evaluated using 40 memory samples with 3 - 6 facts. The dialogue system is initialized with the memory. A valid read query (an utterance which triggers a memory read) is executed on every fact. If the network gives a wrong answer, the error type is reported. Three different types are distinguished. If the action or the natural language response does not match with the expected response, it is counted as a *wrong action* error. In case that the memory contains the needed fact but the system still replies that the fact is not provided, it is counted as the *fact not in memory* error. If the dialogue system replies only a partially correct or a wrong fact, it is counted as a *wrong fact* error.

Table 6.3 visualizes the results of the evaluation. One can notice that the error categories of different tasks do not follow the same distribution. In case of the *query name* task, the error is made most frequently because of a wrong reply. An assumption is that in these cases the formulation was too different and not well-covered by the data set. Additionally, the network does not need to search for the name in the memory because it is always at the first position. User-given facts like the food preferences and the location of the objects are appended to the memory. Depending on which order the user tells this information, it is stored in the memory. The assumption is that the lookup in the unordered memory is

[2] $Precision = \frac{\sum TP}{\sum TP + \sum FP}$, $Recall = \frac{\sum TP}{\sum TP + \sum FN}$ with TP=true positive, FN=false negative, TN=true negative, FP=false positive

Task	wrong action	fact not in memory	wrong fact
Query name	95 %	0 %	5 %
Query food preferences	5 %	70 %	25 %
Query location of object	5 %	80 %	15 %
Query content of memory	10 %	5 %	85 %

Table 6.3: Memory query error types.

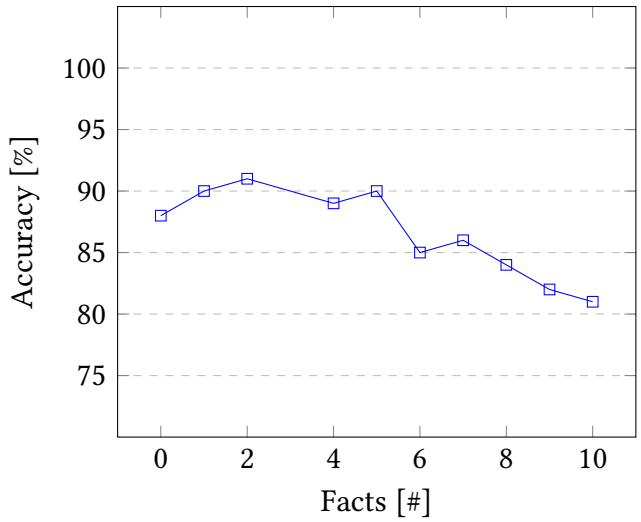

Table 6.4: Accuracy of given responses dependent on the number of facts in the memory.

more difficult. An additional difficultly is that the added fact is exactly the input utterance of the user. In some cases the network is not able to draw the connection between the question and the fact in the memory input because of no uniform format. In case of *query content of memory* mostly some parts of the memory were missing. The problem might be that outputting the content of the memory is not a natural way of speaking which is not modeled well by the architecture.

6.3.2.3 Memory Fill Level Performance

This evaluation task focuses on how the network performs with different amount of facts in the memory. The network is initialized with 22 different memory sets. For every number of facts (0 - 10 facts) there are respectively 2 test sets including different facts. A single fact is queried using three different question formulations. The true positive rate $TPR = \frac{\sum TP}{\sum TP + \sum FN}$ is reported in graph 6.4 where TP are the true positive and FN the false negative samples.

6.3.3 Attention Visualization

In this section the attention weights of the transformer network are visualized to understand which parts of the input the network puts focus on. A visualization tool [28] is used

to generate the figures. In Figure 6.1 and 6.2 the weights of a single attention head of the last layer are visualized. The grey highlighted token in the left columns of the visualizations is attended while the token in the right columns are being attended. The intensity of the highlighted color corresponds to the magnitude of the attention weight.

Figure 6.1 visualizes the attention weights of the same input for three tokens to be generated. Attention is especially put on the memory content and on the input of the user (*Hello*) to generate the *<greet>* special token. In the middle and right visualisation the parameter of the action is generated.

In Figure 6.2 the weights of three different inputs are visualized. In the first sample (left) the user queries a fact which is not in the memory. Attention is put on the memory and on the input to generate the *<use_fact>* special token. However, the second sample (center) contains the queried fact. Attention is jointly put on the relevant part of the memory and of the user input. In the last sample (right) the memory contains a not needed fact and the user asks a memory independent question. The attention weights are visualizing well that the network only focuses on the user input to generate the *<help>* special token.

Based on the attention weight visualization the assumption is that the network has learned to put attention on relevant parts of the memory dependent on the input of the user.

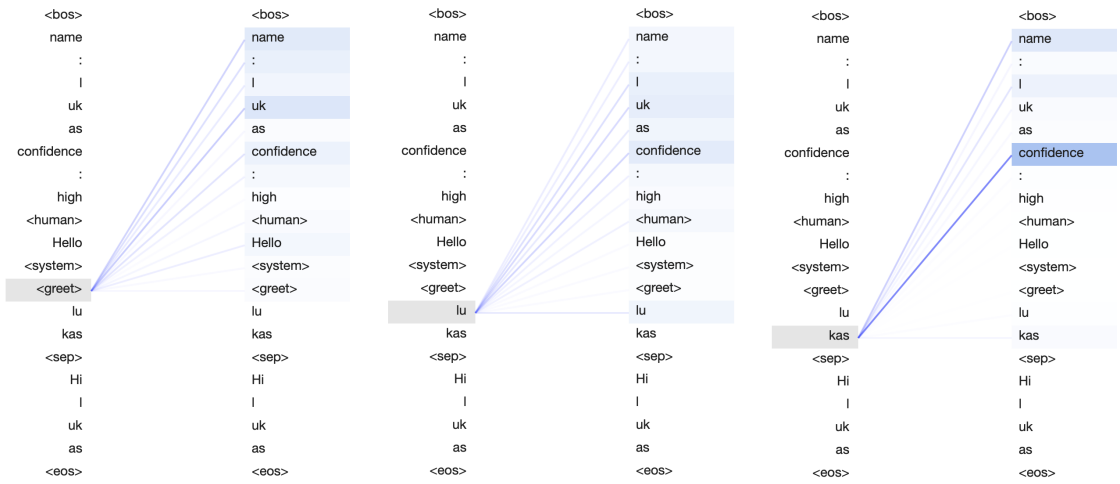

Figure 6.1: Attention visualization of the same input but of different tokens.

6.4 End-to-end example

In the following a sample dialogue is presented where the conversation flow gets adapted based on the current camera input. One person has two subsequent dialogues with the system. However, the appearance of the person changes from Figure 6.3 a) to Figure 6.3 b) after the first dialogue. The dialogue shows that firstly the identity is unknown to the system and therefore it asks for a name. While interacting the fact also gets assigned to the identity. Since the camera input has changed in the second dialogue, the system validates

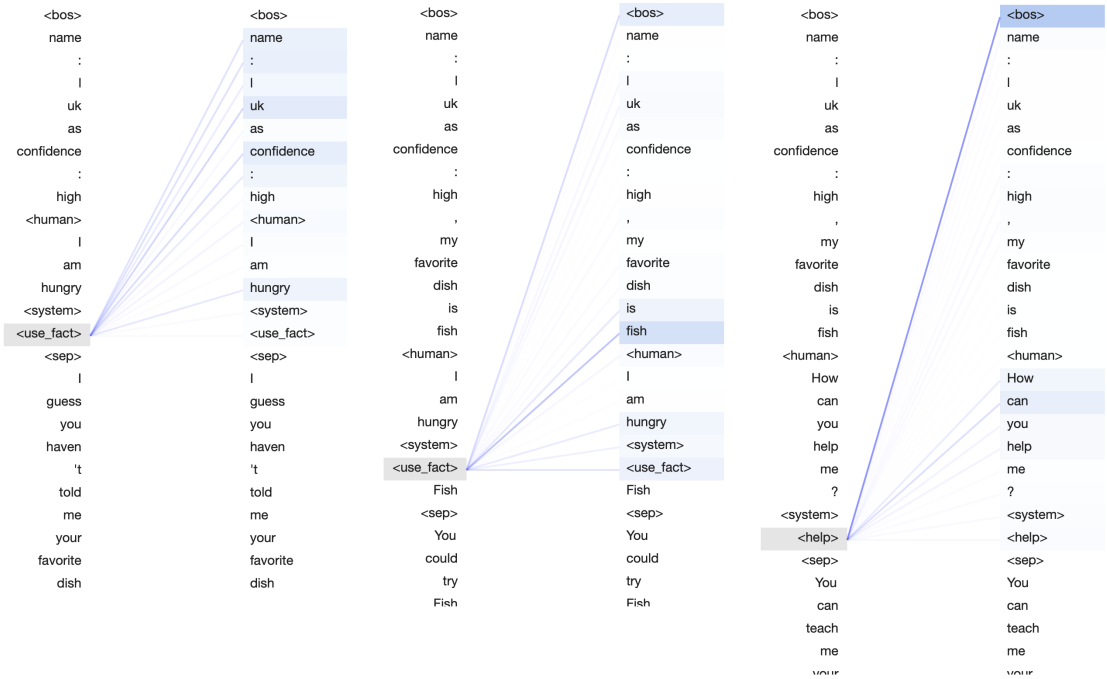

Figure 6.2: Attention visualization of three different inputs. Left: user input which requires a memory lookup but fact is not present in the memory. Middle: user input which requires a memory lookup with fact present in the memory. Right: user input which which does not require a memory lookup.

the assumption that it is still the same identity. Since the user agrees, both input images refer finally to the same memory. That can be verified since the system knows where the computer is located.

HUMAN: hi.

SYSTEM: Hey, I haven't met you so far, what's your name?

HUMAN: my name is Lukas.

SYSTEM: ok. I will not forget it.

HUMAN: i put my computer now in the living room.

SYSTEM: I'll keep it in mind

INFO: *Camera input changed*

HUMAN: hi i am back.

SYSTEM: Hey, Lukas. Is it you?

HUMAN: yes it's me

SYSTEM: Ok, I was not sure. Good afternoon, Lukas

HUMAN: i can't find my computer

(a) (b)

Figure 6.3: Normalized camera input images of the same identity.

SYSTEM: You told me that your computer is in the living room.

The examples below show that the dialogue system is limited to the domain covered through the training data. Since there are no chit-chat samples represented, the system tries to map the input to a memory query response.

HUMAN: how are you?

SYSTEM: I'm not sure where you left your objects.

HUMAN: how is the weather today?

SYSTEM: I don't now where you are storing your weather."

7 Conclusion

This chapter summarizes the work and introduces further ideas how to improve the presented approach.

7.1 Summary

In this thesis a personalized dialogue system was developed by using a camera input for the person identification and by injecting a user profile as a bias into the dialogue system. It represents an increment into the direction of more natural conversations between humans and computer systems. Especially in a field of application of kitchen robot systems it is important to use multiple modalities in order to understand the user right and to be able to have personalized interaction.

Besides that, the presented approach shows that a pre-trained high-capacity network is able to learn to read from a memory with a relative small training set. Instead of learning a general probability distribution to generate universal responses, the personal information is taken from the memory to adapt the dialogue. At the same time the approach also shows how important the training data is. Even the network works reasonably well, there are cases where the system does not behave right. The reason is that these cases are not covered by the training data due to the fact that it is a synthetically generated data set. In case that the human input differs too much from the data, the intent might not be understood right. Additionally, the system is limited to a specific domain. However, by enriching the training data set it is possible to extend the capabilities to further domains. It leads to the assumption that combining two datasets works best. A mixture of a synthetically generated data along with a dataset which contains samples of human dialogues might be a good choice. Generating such a data set implies that the data needs to be annotated with the memory content within the dialogue.

The proposed approach emits special tokens to store facts into the memory. Depending on the action type the network stores the raw user input. The evaluation shows that the accuracy degrades with the amount of facts in the memory. To avoid the issue of storing duplicated information or providing the capability to update information, a further special token might be introduced.

To conclude, a working prototype of a dialogue system capable of adapting to different profiles was built. The system's behaviour to generate a personalized response is influenced by a memory profile that is selected by a computer vision component. When deploying

such a system in a real world scenario, the dialogue system might be more engaging for the user due to the more natural and human conversation.

7.2 Future Work

There are many further directions in order to extend the capabilities of the system or to improve the performance of the model.

Dataset improvements:
To accomplish the full potential of the developed architecture, a large scale dataset covering multiple domains is beneficial. As already pointed out, the dataset should contain various formulations collected from natural dialogues. Two fine-tuning stages might be one approach to avoid the need of annotating the data. Firstly, the model is trained on the annotated and syntetic dataset. Secondly, the model can be trained on the collected dataset. The expectation is that the model still generalizes to learn to take the memory into account.

Memory representation:
Currently the entire utterance of the users is stored in the memory. An improvement could be to use a structured memory representation. For instance, to indicate the favorite food an array-like data structure might be better: *favorite food: apples, sea food.* The claim is that the network can query the already pre-proceed data easier.

Model improvements:
On the architectural side the model might be extended with a memory network. The intention is that the network is capable of learning what is relevant to keep in the memory and which entries can be overwritten or deleted. Currently facts are just added to the memory but not updated or removed.

Robot integration:
In order to use the developed system in the ARMAR kitchen robot and to control it with voice, an automatic speech recognition system (ASR) would be needed. Usually such systems return the textual representation of the speech with the special characters filtered out. The developed system needs to be retrained to be resistant to the output of an ASR system. At the moment special characters are taken into account for the response generation.

Bibliography

[1] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. "VGGFace2: A dataset for recognising faces across pose and age". In: *International Conference on Automatic Face and Gesture Recognition*. 2018.

[2] Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. "A survey on dialogue systems: Recent advances and new frontiers". In: *Acm Sigkdd Explorations Newsletter* 19.2 (2017), pp. 25–35.

[3] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. "Survey on evaluation methods for dialogue systems". In: *arXiv preprint arXiv:1905.04071* (2019).

[4] *Encoder Decoder Image*. https://smerity.com/articles/2016/google_nmt_arch.html. Accessed: 2020-04-24.

[5] *Fully Connected Network Image*. https://www.researchgate.net/figure/Typical-architecture-of-fully-connected-deep-neural-network_fig1_326570271. Accessed: 2020-04-24.

[6] Marjan Ghazvininejad, Chris Brockett, Ming-Wei Chang, Bill Dolan, Jianfeng Gao, Wen-tau Yih, and Michel Galley. "A knowledge-grounded neural conversation model". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[8] Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. "The Curious Case of Neural Text Degeneration". In: *CoRR* abs/1904.09751 (2019). arXiv: 1904.09751. URL: http://arxiv.org/abs/1904.09751.

[9] Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. "The curious case of neural text degeneration". In: *arXiv preprint arXiv:1904.09751* (2019).

[10] Chia-Wei Liu, Ryan Lowe, Iulian Serban, Mike Noseworthy, Laurent Charlin, and Joelle Pineau. "How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2122–2132. DOI: 10.18653/v1/D16-1230. URL: https://www.aclweb.org/anthology/D16-1230.

[11] Ilya Loshchilov and Frank Hutter. "Fixing Weight Decay Regularization in Adam". In: *CoRR* abs/1711.05101 (2017). arXiv: 1711.05101. URL: http://arxiv.org/abs/1711.05101.

[12] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[13] *Overfitting Image*. `https://en.wikipedia.org/wiki/Overfitting`. Accessed: 2020-04-24.

[14] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. "BLEU: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. "Scikit-learn: Machine learning in Python". In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[17] Qiao Qian, Minlie Huang, Haizhou Zhao, Jingfang Xu, and Xiaoyan Zhu. "Assigning personality/identity to a chatting machine for coherent conversation generation". In: *arXiv preprint arXiv:1706.02861* (2017).

[18] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners". In: (2019).

[19] *RNN Image*. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2020-04-24.

[20] *RNN Image*. `https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129`. Accessed: 2020-04-24.

[21] Stephen Roller, Emily Dinan, Naman Goyal, Da Ju, Mary Williamson, Yinhan Liu, Jing Xu, Myle Ott, Kurt Shuster, Eric M Smith, et al. "Recipes for building an open-domain chatbot". In: *arXiv preprint arXiv:2004.13637* (2020).

[22] Frank Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Tech. rep. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[23] Johan AK Suykens and Joos Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[24] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261 (2016). arXiv: 1602.07261. URL: http://arxiv.org/abs/1602.07261.

[25] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[26] Ryota Tanaka, Akihide Ozeki, Shugo Kato, and Akinobu Lee. "An Ensemble Dialogue System for Facts-Based Sentence Generation". In: *arXiv preprint arXiv:1902.01529* (2019).

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[28] Jesse Vig. "A Multiscale Visualization of Attention in the Transformer Model". In: *arXiv preprint arXiv:1906.05714* (2019). URL: https://arxiv.org/abs/1906.05714.

[29] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. "Phoneme recognition using time-delay neural networks". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.3 (1989), pp. 328–339.

[30] Alex Waibel. *Consonant Recognition by Modular Construction of Large Phonemic Time-Delay Neural Networks*. http://isl.anthropomatik.kit.edu/cmu-kit/downloads/Constant_Recognition_by_Modular_Constrution.pdf.

[31] Thomas Wolf, Victor Sanh, Julien Chaumond, and Clement Delangue. *TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents*. 2019. arXiv: 1901.08149 [cs.CL].

[32] Yilin Yang, Liang Huang, and Mingbo Ma. "Breaking the Beam Search Curse: A Study of (Re-)Scoring Methods and Stopping Criteria for Neural Machine Translation". In: *CoRR* abs/1808.09582 (2018). arXiv: 1808.09582. URL: http://arxiv.org/abs/1808.09582.

[33] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. "Joint face detection and alignment using multitask cascaded convolutional networks". In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503.

[34] Yizhe Zhang, Siqi Sun, Michel Galley, Yen-Chun Chen, Chris Brockett, Xiang Gao, Jianfeng Gao, Jingjing Liu, and Bill Dolan. "DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation". In: *arXiv preprint arXiv:1911.00536* (2019).

[35] Tiancheng Zhao and Maxine Eskenazi. "Towards End-to-End Learning for Dialog State Tracking and Management using Deep Reinforcement Learning". In: *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. Los Angeles: Association for Computational Linguistics, Sept. 2016, pp. 1–10. DOI: 10.18653/v1/W16-3601. URL: https://www.aclweb.org/anthology/W16-3601.