Karlsruhe Institute of Technology

# Multimodal goal-oriented dialog using Encoder-Decoder-Networks

Bachelor's Thesis
of


## Leonard Bärmann

KIT Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Interactive Systems Lab (ISL)


Reviewers: Prof. Dr. Waibel
Prof. Dr. Asfour

Advisors: Dr. Jan Niehues
M. Sc. Stefan Constantin

Duration: May 15, 2018 – September 14, 2018

**Erklärung:**

Ich versichere hiermit, dass ich die Arbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis beachtet habe.

Karlsruhe, den 14. September 2018

Leonard Bärmann

**Abstract:**

Based on previous work exploring the use of neural networks to craft goal-oriented dialog systems (Bordes and Weston, 2016; Constantin et al., 2018), this thesis proposes multimodal encoder-decoder networks combining textual and visual information to create an end-to-end trainable goal-oriented dialog model. Different architectures and approaches for multimodality are evaluated and compared at the example of the ARMAR-III kitchen robot. A dataset composed of kitchen-scene images with corresponding example dialogs is introduced, focusing on ambiguous commands which can only be resolved by considering the image content. Evaluation results show a notable improvement of the best multimodal model in comparison to the text-only baseline, although there is still room for refinement since the image feature extraction turned out to be a bottleneck. Nevertheless, the number of wrong ambiguity detections is approximately halved, showing that the proposed models actually learn to distinguish ambiguous from unambiguous situations using visual information provided by the second modality. Additionally, interesting connections concerning the type of visual input and the place where to merge both modalities are revealed. Finally, contributing to further research on multimodal sequence-to-sequence tasks, all implemented models are open-sourced as an extension to the OpenNMT framework by Klein et al. (2017).

**Kurzzusammenfassung:**

Aufbauend auf vorherigen Untersuchungen zum Einsatz neuronaler Netze für zielorientierte Dialogsysteme (Bordes and Weston, 2016; Constantin et al., 2018) stellt diese Arbeit multimodale, Text- und Bildinformationen kombinierende Encoder-Decoder-Netzwerke vor und nutzt diese für ein Ende-zu-Ende-trainierbares, zielorientiertes Dialogsystem. Dabei werden verschiedene Architekturen und Ansätze am Beispiel des ARMAR-III Küchenroboters ausgewertet und verglichen. Ein aus Fotos von Küchenszenen und zugehörigen Dialogen bestehender Datensatz wird vorgestellt, wobei der Fokus auf mehrdeutigen Kommandos liegt, welche nur mithilfe der Bildinformationen aufgelöst werden können. Die Versuchsergebnisse zeigen eine deutliche Verbesserung des besten multimodalen Modells gegenüber dem nur auf den Dialogdaten arbeitetenden Baseline-Modell, auch wenn aufgrund der verhältnismäßig schlechten Bilderkennung noch Luft nach oben besteht. Dass die Anzahl der falschen Ambiguitätserkennungen um etwa die Hälfte reduziert wird, zeigt die Fähigkeit der vorgestellten Modelle, den zweiten, visuellen Sinn für die Ambiguitätsauflösung zu verwenden. Zusätzlich wird eine interessante Verbindung zwischen der Art der visuellen Eingabe und dem Ort der Kombination von beiden Sinnen aufgedeckt. Die implementierten Modelle werden als Erweiterung des OpenNMT-Frameworks von Klein et al. (2017) veröffentlicht, um die weitere Forschung an multimodalen sequence-to-sequence Netzwerken zu unterstützen.

# Contents

# 1. Introduction

## 1.1. Motivation

The key to success of technological products is its interface – users expect their devices to be easy to use and want to focus on the task rather than the technology. Starting with the first home computers requiring huge manuals to study all commands to type, technology evolved to become more intuitive and usable for non-experts first by the introduction of the mouse and, later, of touch screens. The next, already ongoing evolution aims at using the probably most intuitive interface all humans possess – their voice. While first primitive voice assistants were already created by academic actors in the 1970s, the consumer break-through of voice assistants started with the introduction of products like Siri, Google Assistant or Amazon Echo.

Despite speech recognition by itself, which cares about converting spoken audio signals to written text, a major challenge for these systems is natural language processing and reasoning. Everybody knows the "Sorry, I don't understand that question" responses frequently occurring when not using exactly the predefined assistant commands. Neural networks, already extremely popular in the image recognition society, offer a chance to escape that limitation when huge amounts of sample data are available.

When comparing nowadays "smart" systems to assistants proposed in science fiction, another major difference is their ability to work with information about their environment and use this for reasoning and dialog composition. For example, one might think of a future home assistant reminding the user of taking an umbrella when the forecast predicts rain – but the system should only place a reminder when he is actually about to leave the house without one. Therefore, a visual sense needs to be connected to the dialog component. When thinking of robot systems, the need for combining multiple senses and information sources for dialog composition is even more evident. To make a first step towards this direction, this work will enhance a kitchen robot's dialog system with a visual sense.

## 1.2. Goal of this work

A goal-oriented dialog system is a system controlled by the user using natural language, designed to carry out a domain-specific task and reach a specific goal. Previous work in this field (Bordes and Weston, 2016; Constantin et al., 2018) solely used the text spoken by the user as input. Based on that, this work aims at combining spoken and visual information as input to a single end-to-end trainable multimodal neural network. That way, the quality of the responses given by the agent should be improved, especially by autonomously detecting and resolving ambiguities. The concepts shall be tested and evaluated at the example of the ARMAR-III kitchen robot (Asfour et al., 2007).

In order to reach the stated goals, first of all the model must be able to distinguish ambiguous from unambiguous commands – for instance, if there is only one cup on the table, the robot should instantly execute the command "please give me the cup". If there are two cups of different color, however, the ambiguity needs to be detected and a further inquiry has to be asked ("Do you want to have the green or the red cup?"). If necessary, the agent should ask multiple questions to resolve all ambiguities and eventually generate a unique API call which can be used to execute the users command.

To conclude, this thesis aims at exploring whether and to which extent the usage of a multimodal neural network improves performance and usability of a goal-oriented dialog system. In particular, different train algorithms, implementation techniques and network architectures shall be compared regarding their suitability for the stated task.

## 1.3. Structure

The present introduction is followed by chapter two dealing with the basics this work is based on. Beginning with a single perceptron cell, the complexity level is raised step by step visiting Multi-Layer Perceptrons, Recurrent Neural Networks and finally Encoder-Decoder-Networks. Thereby, the math notation used later on is introduced. Afterwards, related work from different fields of research is presented and discussed. Knowing the current state of art, the fourth chapter proposes different architectures for multimodal encoder-decoder networks and their usage for a goal-oriented dialog system. Evaluating these models is the next step in chapter five. Initially, the created dataset is introduced and analyzed, whereupon the results of the multimodal dialog model experiments are presented. Finally, the last chapter draws a conclusion and highlights some points for further research.

Additional information about the implementation of the dataset generator as well as the multimodal models can be found in appendix A. Subsequently, some supplementary figures and examples concerning the created dataset are included. Details about model training, experimental setup and results make up the end of the appendices.

# 2. Basics

The following chapter describes some of the basic techniques on the field of artificial neural networks used in this work.

## 2.1. Perceptron

A perceptron is the smallest building block of an artificial neural network and has similarities to a single biological neuron as well in structure as in function. The following explanations build on (Bishop, 2006, section 4.1.7).

**Function** A perceptron defines a function $y : \mathbb{R}^n \to \mathbb{R}$ of $n$ input values given as a vector $\tilde{x} \in \mathbb{R}^n$ into the set of real numbers. The function $y$ is determined by weights $\tilde{w} \in \mathbb{R}^n$, a bias $w_0 \in \mathbb{R}$ and an activation function $f : \mathbb{R} \to \mathbb{R}$:

$$y(\tilde{x}) = f(\tilde{w}^T \tilde{x} + w_0) \tag{2.1}$$

By defining $x = \begin{pmatrix} 1 \\ \tilde{x} \end{pmatrix}, w = \begin{pmatrix} w_0 \\ \tilde{w} \end{pmatrix}$, this can be simplified to:

$$y(x) = f(w^T x) \tag{2.2}$$

Usually, $f$ is a nonlinear function like $step$:

$$step(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases} \tag{2.3}$$

Figure 2.1 visualizes the perceptron function. The output of $y(x)$ can be interpreted as a classification of $x$ in a two class problem of $C_1, C_2$: If $y(x) > 0$, then $x \in C_1$, if $y(x) < 0$ equivalently $x \in C_2$. Thus, the equation $y(x) = 0$ defines the $(n-1)$-dimensional hyperplane separating the classes $C_1$ and $C_2$ in $\mathbb{R}^n$. Thereby, $\tilde{w}$ represents the normal vector of this hyperplane, $-w_0$ is the offset from the origin in direction of $\tilde{w}$.

**Training** The goal of running a train algorithm on a perceptron is to automatically determine the parameters $\tilde{w}, w_0$ to optimize the prediction of the classes $C_1, C_2$ on a given train dataset. Therefore, target values $t \in \{-1, 1\}$ are defined by setting $t = 1$ for all $x \in C_1$ and $t = -1$ for all $x \in C_2$. Consequently, a sample $x$ is correctly classified by the perceptron (that means, $w^T x > 0$ iff $x \in C_1$), iff $w^T x \cdot t > 0$. The perceptron algorithm now tries to minimize the error function

$$E(w) = -\sum_{x \in M} w^T x \cdot t \tag{2.4}$$

where $M$ is the set of misclassified samples. The negation sign is required because $w^T x \cdot t < 0$ for the (misclassified) $x \in M$.
The train algorithm finally consists of the following steps:

1. Forward-pass the training data through the perceptron (i.e. calculate $y(x)$ for each sample $x$) and determine the set of misclassified samples $M$.

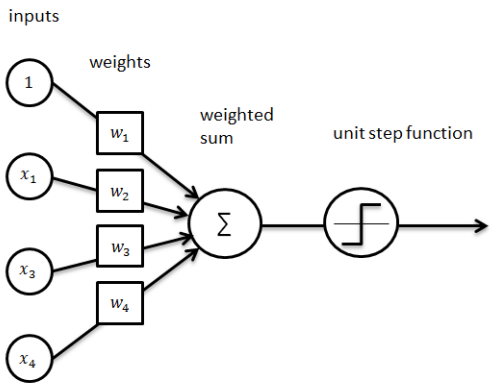



Figure 2.1.: Illustration of a single perceptron according to equation (2.2). Graphic taken from Taspinar (2016)

2. For each $x \in M$, adjust the weight vector $w$ in the direction of the gradient of the error function (so that the error decreases):
$$w \leftarrow w - \nabla E(w) = w + x \cdot t \tag{2.5}$$

3. Repeat until all samples are classified correctly, i.e. $E(w) = 0$.

**Limitations** The perceptron algorithm only works for linear-separable problems, if a problem is not, the perceptron algorithm will run infinitely. Additionally, until the algorithm stops, it is impossible to say if the problem is non-linear or the convergence is just slow.

**Multiple parallel perceptrons** When multiple classes shall be detected on a dataset at once, multiple parallel perceptrons may be used. This leads to the replacement of the weight vector $w \in \mathbb{R}^n$ with a weight matrix $W \in \mathbb{R}^{m \times n}$, where the $i$-th row of $W$ is the transposed weight vector of the $i$-th perceptron, with $m$ being the size of the perceptron layer. As a result, the output of this "layer" of perceptrons is a vector rather than a single value. The perceptron function modifies to

$$y : \mathbb{R}^n \to \mathbb{R}^m, y(x) = f(Wx) \tag{2.6}$$

where $f$ is widened to $\mathbb{R}^m$ by pointwise application. For training a layer of perceptrons, different loss functions can be used. With $d \in \{0,1\}^m$ being a vector of target values and $y = y(x)$ the output of the layer, a straightforward approach is to use the mean squared error function:

$$E = \frac{1}{2} \sum_{i=1}^{m} (y_i - d_i)^2 \tag{2.7}$$

The so called "backpropagation" algorithm now calculates the derivative $\frac{\partial E}{\partial w_{ij}}$ for all entries of the weight matrix $W$. These are used to update the perceptron's parameters at each epoch.

## 2.2. Multi-Layer Perceptron (MLP)

To solve the perceptron limitations mentioned above, it is possible to connect multiple layers of perceptrons to build up a more powerful model. Thereby, the output of one layer is used as the input of the next one (see figure 2.2). Each layer $i$ has its own weight matrix $W^{(i)}$. The function $y$ of an MLP is therefore

defined recursively by:

$$\begin{aligned} y_1(x) &= f_1(W^{(1)}x) \\ y_i(x) &= f_i(W^{(i)}y_{i-1}(x)) \\ y(x) &= y_N(x) \end{aligned} \tag{2.8}$$

with $N$ being the number of layers, $y_i$ the output of layer $i$ and $f_i$ its activation function. A common choice for the activation function is the sigmoid function $\sigma$:

$$\sigma : \mathbb{R} \to [0,1], \sigma(x) = \frac{1}{1+e^{-x}} \tag{2.9}$$

Other non-linearities include tanh or the rectified linear unit function $relu(x) = \max(0,x)$. The last layer is called the output layer and its yielded values are interpreted as classification results. Intermediate or "hidden" layers produce output called "hidden state", which tends to represent lower level features depending on the layer depth.

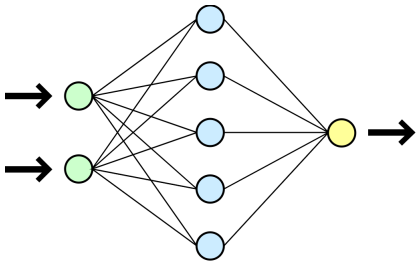


Figure 2.2.: Illustration of a two-layer MLP. The green circles represent the input values, blue the hidden layer and yellow the output layer. Each connection line is associated with a weight and corresponds to an entry in the weight matrix of that layer. Figure taken from Mysid and Dave (2006)

Because of their multiple layers, MLPs are able to model non-linear classification problems, such as XOR. Many different variants of the standard perceptron layer have come up over time and most of them can be combined and stacked up. Standard perceptron layers are most often used as transition or final layers, as seen in section 2.4.

**Datasets** (Hastie et al., 2001, p. 222). When training a neural network for a given task, a dataset containing input-output samples is required. The same dataset should not be used for evaluating the model to keep the network from simply remembering solely the training data. Therefore, an additional test dataset, containing samples not seen in the train set, is used to determine the generalization capability of the trained model. Furthermore, it is useful to evaluate the current error during training to determine if training can be stopped. Because the data used for that calculation strongly affects the train procedure (although it is not actually used for training), a third dataset called "validation" is introduced. To summarize, there are commonly three sets of data involved:

- *Train*: The train dataset is used to optimize the model with the backpropagation algorithm.
- *Validation*: The validation dataset is used for model selection. Constantly evaluated during training (e.g. after each epoch), it determines when to stop training (or more generally: which hyperparameters to choose).
- *Test*: The final assessment of the model's generalization capability is done using this dataset.

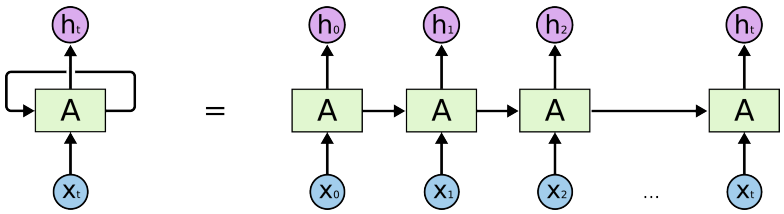



Figure 2.3.: A single RNN cell on the left, its time-unrolled version on the right. Graphic taken from Olah (2015)

## 2.3. Recurrent Neural Networks (RNN)

The goal of RNNs is to process variable-length sequences, whereas a standard MLP can only receive a fixed amount of input values. This is especially applied to input with a temporal order, e.g. in speech recognition or natural language processing (NLP). Based on Chung et al. (2014) and Olah (2015), the following section first explains the fundamental idea of RNNs and then proceeds to the commonly used LSTM architecture.

**Concept** To process variable length sequences over time, the core idea is to add recurrent connections to the neuron cells. That way, at each time step $t$, the neuron uses the input[1] $x_t$ together with the hidden state of the previous time step $h_{t-1}$ to produce the next output (= hidden state) $h_t$. The first time steps $h_{t-1} = h_0$ can be initialized arbitrarily, e.g. set to zero. As a result, the RNN's function $y$ can be expressed recursively as

$$\begin{aligned} y_t(x) &= f(Wx_t + Uh_{t-1}) \\ y(x) &= (y_1(x), \ldots, y_T(x)) \end{aligned} \tag{2.10}$$

with $U$ being a trainable weight matrix for the recurrent connection and $T$ the length of the input sequence, i.e. the total number of time steps the RNN works on. Note that the output of the RNN is also a sequence, containing the output of all time steps. Similar to MLPs, recurrent networks can be stacked by evaluating each single layer as explained above while redirecting the output $y$ of one layer to the input $x$ of the next one.

**Backpropagation** To train a recurrent neural network, a variant of backpropagation called "backpropagation through time" is applied. The core idea is to unfold the recurrent connections over time (as shown in figure 2.3), producing a (deep) non-recurrent network on which the default train algorithm can be applied. Equation (2.10) already suggests this procedure. By viewing the output at time step $t$ as a function of the input $x_t$ and the previous hidden state $h_{t-1}$, the final output $y_T$ and consequently the error function $E$ can be derived to the contributions $\frac{\partial E}{\partial w_{i,j}}$ and $\frac{\partial E}{\partial u_{i,j}}$ for adjusting the parameter matrices $W$ and $U$, respectively.

**LSTM** While basic RNNs allow to work with sequential data, they often suffer from the vanishing gradient problem when working with long input sequences (Hochreiter, 1991). The main issue is that, due to the hidden state being overwritten at each time step, the gradient vanishes (multiplies near to zero) when applying too many time steps, and therefore, the net is unable to be trained to "remembering" relevant facts over a longer period of time. Long short-term memory (LSTM) cells introduced by Hochreiter and Schmidhuber (1997) avoid this problem by adding multiplicative weights, so called "gates", to the hidden state $h_t$ and introducing an additional cell state $c_t$ (also referred to as "memory"). At time step $t$, first, a new cell state candidate $\tilde{c}_t$ is calculated using

$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1}) \tag{2.11}$$

[1] where $x_t$ can be the user input or the output of a previous layer, if RNN layers are stacked

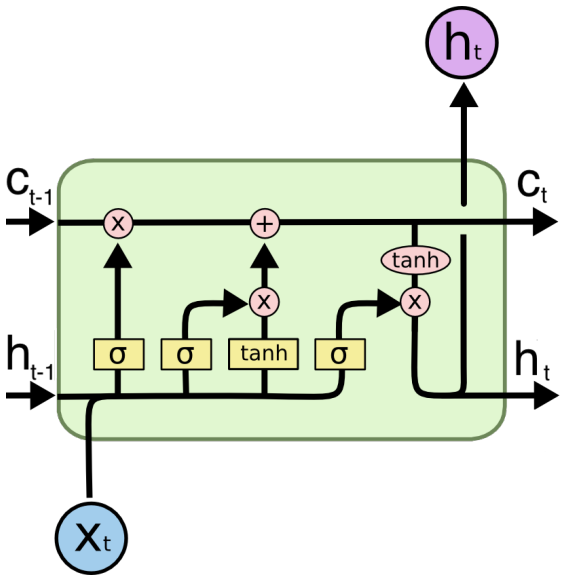



Figure 2.4.: Illustration of the calculations done inside a single LSTM cell. $\sigma$ and tanh boxes represent a neural network layer with the given activation function, red circles pointwise operations. Merging vector paths are concatenations. Graphic modified from Olah (2015)

Updating the cell state is done by "forgetting" the old cell state data with factors $f_t$ and "inserting" the proposed data with factors $i_t$ as

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) \tag{2.12}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \tag{2.13}$$
$$c_t = f_t \odot c_t + i_t \odot \tilde{c}_t \tag{2.14}$$

with $\odot$ representing pointwise multiplication. Eventually, the hidden state to output is determined by the cell state and an "output gate" multiplier $o_t$ specifying the amount of memory information to let through:

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \tag{2.15}$$
$$h_t = o_t * tanh(c_t) \tag{2.16}$$

All matrices $W_*$ and $U_*$ are trainable parameters. Figure 2.4 summarizes the calculations done inside an LSTM cell.

Using the controlled insertion and deletion of memory state independently of the output value, LSTM cells are able to remember relevant information over long time distances. Therefore, most major successes with RNNs were actually achieved using LSTMs or some of its many variants (Chung et al., 2014).

## 2.4. Encoder-Decoder Network

Encoder-Decoder Networks are a network architecture first introduced by Sutskever et al. (2014) designed to handle sequence-to-sequence tasks. The general idea is to use an RNN encoder network transforming the input sequence $\mathbf{x} = (x_1, \ldots, x_N)$ to a fixed-length representation (context) vector $c$. Another RNN called "decoder" then uses this context vector to generate outputs $\mathbf{y} = (y_1, \ldots, y_{M-1}, y_M)$, with the final output $y_M$ being a special token to represent the sequence end. Note that the input sequence length $N$ and the output sequence length $M$ may differ.

**Function** More formally, with $rnn_{enc}$ being the encoder RNN time step function and $h_t$ the output hidden states of that RNN, the context vector $c$ is defined as

$$\begin{aligned} h_0 &= 0 \\ h_t &= rnn_{enc}(x_t, h_{t-1}) \\ c &= h_N \end{aligned} \tag{2.17}$$

with $t$ running from 1 to $N$. Decoding with the $rnn_{dec}$ function uses the context vector to produce its outputs $s_t$:

$$\begin{aligned} s_0 &= 0 \\ s_t &= rnn_{dec}(c, s_{t-1}) \end{aligned} \tag{2.18}$$

Eventually, an MLP layer $g$ called "generator" maps the decoder hidden states to the actual output token probabilities:

$$\mathbb{P}(y_i|t) = g(s_t)[i] \tag{2.19}$$

$$y_t = \underset{y_i \in V_{out}}{\operatorname{argmax}} \left(\mathbb{P}(y_i|t)\right) \tag{2.20}$$

where $V_{out} = \{y_1, \ldots, y_O\}$ is the output vocabulary set and $g$ outputs a vector of dimension $O$. $g$ usually uses the $softmax$ function as activation function to ensure all values sum up to 1, which is required by the probability interpretation.

**Training and Inference** During training, instead of using the previous output of the decoder as $s_{t-1}$, the target output can be used. When testing the model, the probability $\mathbb{P}(y_t|y_{t-1}, \ldots, y_1, \mathbf{x})$ needs to be maximized. The easiest strategy is to use a greedy algorithm and simply use the $y_t$ giving the highest probability at each time step as shown by equation (2.20). A more sophisticated implementation will use an algorithm like beam search to search the path $y_1, \ldots, y_t$ with the highest probability while avoiding an exponential explosion of computation time.

**Attention** Making encoder-decoder models competitive with state-of-the-art phrase based machine translation systems was a major achievement of Bahdanau et al. (2014). They extended the framework by adding a so-called "attention" mechanism. Informally, it enables the decoder to choose on which part of the input to focus while generating the next output word (Olah and Carter, 2016). Instead of using solely the final encoder hidden state $h_N$ as the context vector $c$, the whole sequence of hidden states $h_1, \ldots, h_N$ is kept. The context vector is then computed at each decoder time step, i.e. $c$ is replaced by a time-dependent $c_t$ in (2.18). Based on the encoder output sequence, $c_t$ computes to

$$c_t = \sum_{i=1}^{N} \alpha_{i,t} h_i \tag{2.21}$$

where the $\alpha_{i,t}$ can be interpreted as a measure of how much the output at time $t$ aligns with the input at time $i$. These alignment factors are computed based on an attention model $a$, which is usually implemented by a simple MLP layer, using the last decoder state to score the relevance of all encoder states. A softmax is applied to normalize the attention weights to sum up to 1:

$$\begin{aligned} e_{i,t} &= a(s_{t-1}, h_i) \\ \alpha_{i,t} &= \frac{\exp(e_{i,t})}{\sum_{k=1}^{N} \exp(e_{k,t})} \end{aligned}$$

The attention $a$ is differentiable and therefore can be trained along with the other parts of the model. This mechanism led to major improvements especially for lengthy input sequences, since "by letting the decoder have an attention mechanism, [...] the encoder [is relieved] from the burden of having to encode

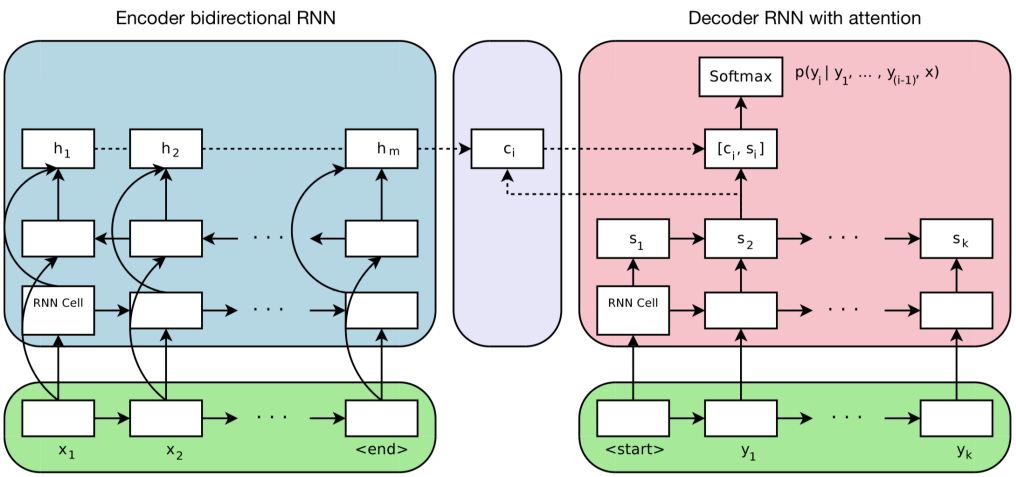



Figure 2.5.: Schematic depiction of the RNN encoder-decoder framework with attention mechanism and a bidirectional encoder. Graphic modified from Britz et al. (2017)

all information in the source sentence into a fixed-length vector" (Bahdanau et al., 2014).

**Bidirectional encoder** A common technique to improve performance of encoder-decoder models is to use a bidirectional encoder, introduced by Schuster and Paliwal (1997). Thereby, two separate encoders are used: The "forward encoder" reads the input sequence in its defined order $x_1, \ldots, x_N$ and produces hidden states $\overrightarrow{h_1}, \ldots, \overrightarrow{h_N}$. Similarly, a "backward encoder" is added receiving the input in reverse order $x_N, \ldots, x_1$ and producing hidden states $\overleftarrow{h_N}, \ldots, \overleftarrow{h_1}$. The final encoder output is simply set to the concatenation of both the forward and backward hidden states:

$$h_t = \begin{pmatrix} \overrightarrow{h_t} \\ \overleftarrow{h_t} \end{pmatrix} \tag{2.22}$$

Figure 2.5 shows the complete encoder-decoder architecture including a bidirectional encoder. Bidirectional encoders have shown to achieve better results, explained by the introduction of additional short-time dependencies in the reversed input sequence.

# 3. Related work

Since this work aims at combining visual and spoken information, there is a broad range of work this thesis builds upon. An exemplary list grouped by research field is given in the following section:

**Image processing** The field of image processing with its wide range of applications such as object recognition or face detection is probably the most advanced discipline of deep learning. The first development of convolutional neural networks by Lecun et al. (1998) built upon time delay neural networks, which were initially introduced for speech recognition by Waibel et al. (1990). Nowadays, state-of-the art CNNs are tuned both for performance and feature generality. For example, He et al. (2016) introduced an architecture named “ResNet”, containing up to 152 layers with residual connections. Feature extraction, i.e. using the outputs of the perpetual layer of a pretrained CNN for another problem the original net was trained against, is a common technique shown to be effective (Yosinski et al., 2014) and therefore used throughout the literature (Antol et al., 2015; Malinowski et al., 2015; Huang et al., 2016; Caglayan et al., 2016).

**Dialog systems** Traditional dialog systems often contain a large amount of handcrafted rules and therefore require a lot of manual work. Data-driven approaches try to eliminate this issue by learning the conversation rules directly from dialog data. Statistical methods such as partially observable Markov decision processes (POMDP) (Young et al., 2013) show good results, but still require a lot of craftsmanship for feature space representations. Vinyals and Le (2015) were one of the first to use a deep learning approach training an encoder-decoder based sequence-to-sequence model on a dialog task. In their study, they examined an open conversation about movie knowledge as well as a (kind of) goal-oriented IT-helpdesk dialog. Beyond that, Bordes and Weston (2016) explored another approach using so called "memory networks" on a goal-oriented restaurant reservation dialog task. This work was proceeded by Constantin et al. (2018), who further evaluated different model types and created an ARMAR-III data set similar to the one used in this work, but without ambiguous command situations.

**Visual question answering (VQA)** Combining visual and spoken input into a single neural network is an approach already explored in the field of VQA. Antol et al. (2015) created a model able to answer various questions about real-world as well as abstract scene images. Using a model similar to the HSM approach explained in section 4.3.1, they processed the question input text using an encoder LSTM and the corresponding image using a pretrained CNN. Thereafter, they combined both internal representation vectors by pointwise multiplication, supplied that into a dense layer and finally applied a softmax to generate the response. In contrast to the model used in this work, they, however, did not use a LSTM decoder but rather chose an answer from a fixed set of utterances.

The architecture proposed by Malinowski et al. (2015) does not involve an encoder-decoder network either. Using a single LSTM net, they first fed all input words into it and afterwards let the same net generate the response words, until an end token is yielded. At each time step, they additionally provided the image features as given by a pretrained CNN with a supplementary trained final linear layer to the LSTM. This approach, which could be called “Simultaneously look and listen”, is not further examined in this thesis.

**Multimodal Neural Machine Translation (MNMT)** Encoder-decoder networks were initially proposed for neural machine translation (NMT) tasks. Consequently, a recent research interest in the NMT community is to improve translation by adding images supplementing the input sentence. The idea is to

disambiguate and clarify the input using the image. This task is referred to as multimodal neural machine translation (MNMT).

Different architectures for combining textual and visual inputs into an encoder-decoder network were examined: Libovický et al. (2016) merged the visual features with the encoder hidden states before using this merged vector to initialize the decoder. While this is very similar to the HSM model proposed in this work, Huang et al. (2016) introduced an approach of the same kind as the FVTL model explained in section 4.3.2. They processed an image using a pretrained CNN and used the extracted features as the first time step's input to the encoder. Furthermore, they examined the addition of regional visual features (in contrast to global features calculated on the entire image) to add focus on specific objects depicted on the image, with the intention of the source language sentence probably mentioning these objects. For this purpose, the region proposal network of Ren et al. (2017) was used to extract object bounding boxes with which the image was clipped. These clipped images were again processed by the CNN and the resulting features were fed into the encoder network, each at its own time step, followed by the features calculated on the entire image and lastly, the source sentence input. Even going further, in a third architecture, multiple encoders sharing their parameter tensors were ran simultaneously, each with its own clipped version of the image (including one running on the entire image). While the decoder received a fused hidden state of all the encoder threads as initial input, it could still attend to each word in one of the threads separately. That way, the best results were achieved.

A "Simultaneously look and listen" approach, where the image features are given to the decoder at each time step, was examined by Caglayan et al. (2016). They proposed a novel, multimodal attention mechanism, where attention is applied to the image feature vector and the encoder hidden state separately. After computing the attended context vectors $c_{img}$ and $c_{text}$ at each time step, both are fused to serve as the decoder input. Calixto et al. (2017) further improved this idea and partly outperformed the model of Huang et al. (2016) described above.

# 4. Models and methods

## 4.1. Image classifier

As a first subtask, a simple multi-class classifier was trained solely on the images of the dataset. Its output is the set of objects and drinks recognized on the given input image. Because of the relatively small dataset size, training a deep convolutional neural net from scratch would have led to poor results. Therefore, existing models pre-trained on ImageNet data (Deng et al., 2009) were utilized. All but their last layer were fixed, whereas the final linear layer leading to the output of ImageNet classes was replaced by a trainable linear layer with its output size set to the number $N$ of all objects in the dataset. This technique known as "feature extraction" or "transfer learning" has shown to improve CNN performance (Yosinski et al., 2014). Different pre-trained model architectures provided by an open-source library [1] were tested to see which net extracts the best features for the given task. Before interpreting the output as object detection results, the *sigmoid* function was applied. Eventually, with the net yielding the result vector $o \in [0,1]^N$, object $x \in [0,N]$ was said to be detected iff $o[x] > 0.5$.

Since this is a multi-class classification problem, the multi label soft margin loss function [2] was used. Different standard optimization algorithms like SDG, Adadelta and Adam were tried. Additionally, experiments with various learning rate (LR) strategies were conducted. Next to fixed and regularly decreasing LR, reduction of LR on validation loss stagnation was also examined. See section 5.2 for the results.

## 4.2. Baseline: Raw dialog net

As a baseline for the other dialog net experiments, the *raw* model was trained solely on the dialogs of the training data. Thereby, the OpenNMT framework (Klein et al., 2017) was utilized without modification to train an encoder-decoder network on the dialog task (represented as a sequence to sequence task). Individual dialog question-answer pairs were decoupled by appending the complete dialog history to the input of the current question, i.e. a dialog of the form $q_1 \rightarrow r_1 \rightarrow \cdots \rightarrow q_n \rightarrow r_n$ with user questions $q_i$ and agent answers $r_i$ was transformed to $n$ independent $(s_i, t_i)$ pairs with the source sentence $s_1 = q_1$, $s_i = q_1 r_1 \ldots q_{i-1} r_{i-1} q_i, (i > 1)$ and the target $t_i = r_i$ (the same technique was applied to the multimodal models, too). A bidirectional encoder was used since they are known to achieve better results than unidirectional ones (Britz et al., 2017). Both encoder and decoder were configured to have two layers with 256 neurons each.

As a reminder and for consistency with the following chapter, figure 4.1 shows the architecture implemented by OpenNMT used as the baseline. First, the input is given to a bidirectional encoder producing hidden states $h_t$. The decoder uses an attention mechanism to compute its context vector $c_t$ based on the sequence of encoder outputs. Eventually, the generator uses the current decoder output to define a probability distribution over the output vocabulary.

[1] https://github.com/Cadene/pretrained-models.pytorch
[2] https://pytorch.org/docs/stable/nn.html#torch.nn.MultiLabelSoftMarginLoss

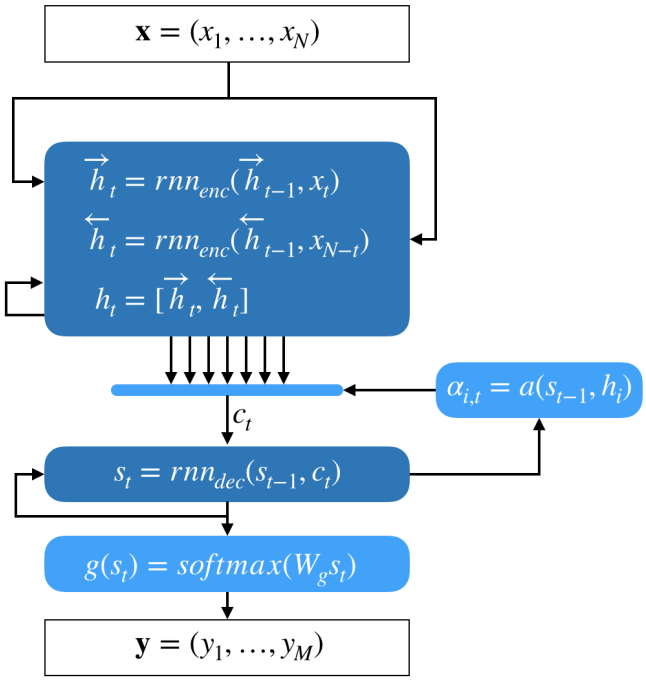



Figure 4.1.: The traditional encoder-decoder architecture with attention mechanism, used as the baseline (4.2). The dark blue boxes represent recurrent LSTM networks (i.e. the encoder and decoder), the light blue ones simple MLP layers.

## 4.3. Multimodal dialog models

The following sections explain the different encoder-decoder architectures for combining visual and textual features examined in this work. Thereby, each training sample consists of a source sentence $\mathbf{x} = (x_1, \ldots, x_N)$ (the dialog history including the users current question), a target sentence $\mathbf{y} = (y_1, \ldots, y_M)$ (the agents response) and a second modality input, i.e. an image representation tensor $\mathbf{i}$. All proposed models share the usage of a second encoder to transform the second modality input $\mathbf{i}$ to a representation vector $f \in \mathbb{R}^S$, with $S$ being the hidden state dimension for the secondary input. As the second modality is an image, using a pre-trained convolutional neural network to extract a feature vector from the image is a reasonable approach. Additionally, the feature vector is passed to a dimensionality reduction layer $d$ applying a trainable linear layer and the sigmoid function.

$$f = d(cnn(\mathbf{i}))$$
$$d(x) = \sigma(W_d x)$$

The following models present different strategies to combine both input modalities $\mathbf{x}$ and $f$ to enable the generation of the target sequence $\mathbf{y}$.

### 4.3.1. Hidden State Merge (HSM)

The HSM model aims at combining the encoded representations of both the primary and secondary input before passing them to the decoder. A bidirectional textual encoder produces a hidden state sequence $(h_1, \ldots, h_N)$ as given by (2.17) and (2.22). The second encoder yields the image representation vector $f$ as explained above. In between the encoder and decoder, these hidden states are now merged by concatenating $f$ with each encoder state and using a linear merge layer $m$ to reduce the dimension according to the needs of the decoder:

$$\tilde{h}_t = W_m \begin{pmatrix} h_t \\ f \end{pmatrix}$$

This merged sequence $(\tilde{h}_1, \ldots, \tilde{h}_N)$ is passed to the decoder to serve as the basis for calculating the attended context vectors $c_t$ according to (2.21), with the decoder eventually generating the output sequence $\mathbf{y}$.

The intention of this model design is to let the model first look and reason about the input sequence and the image separately. After "understanding" each of them on its own, the model combines these representations (i.e. hidden states) into a common space, containing both the image and text information. Using that combined representation, the output sequence is generated.

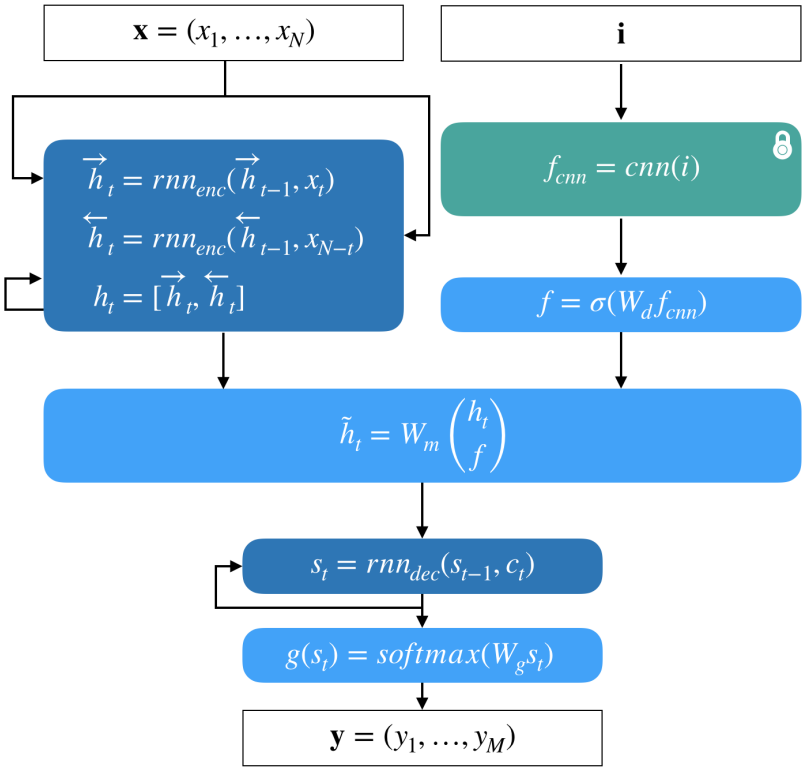



Figure 4.2.: Graphic illustrating the HSM model. The main idea is to combine the internal representations of both the textual and the visual encoder using a merge layer producing $\tilde{h}_t$ values given to the decoder. Since the attention mechanism was not touched, it is not displayed here (as in the following figures).

### 4.3.2. First View Then Listen (FVTL)

The "First view then listen" model behaves as its name suggests: Before receiving the input text sequence, the model is allowed to take a look at the image. This is done by using the image representation $f$ as the initial state of the bidirectional encoder (in both directions). An additional linear conversion layer $t$ is used to transform $f$ to the encoder RNN dimension. More formally, this means (2.17) modifies to:

$$h_0 = W_t f$$
$$h_t = rnn_{enc}(h_{t-1}, x_t)$$

The rest of the model architecture is kept unchanged.

The idea behind this model is to supply the image information as soon as possible. The used LSTM cells are able to capture relevant (i.e. ambiguity resolving) information over many time steps. Therefore, the hope is that the model converges to use this information during encoding (and through the $h_t$ also during decoding) to produce output resolving all ambiguities.

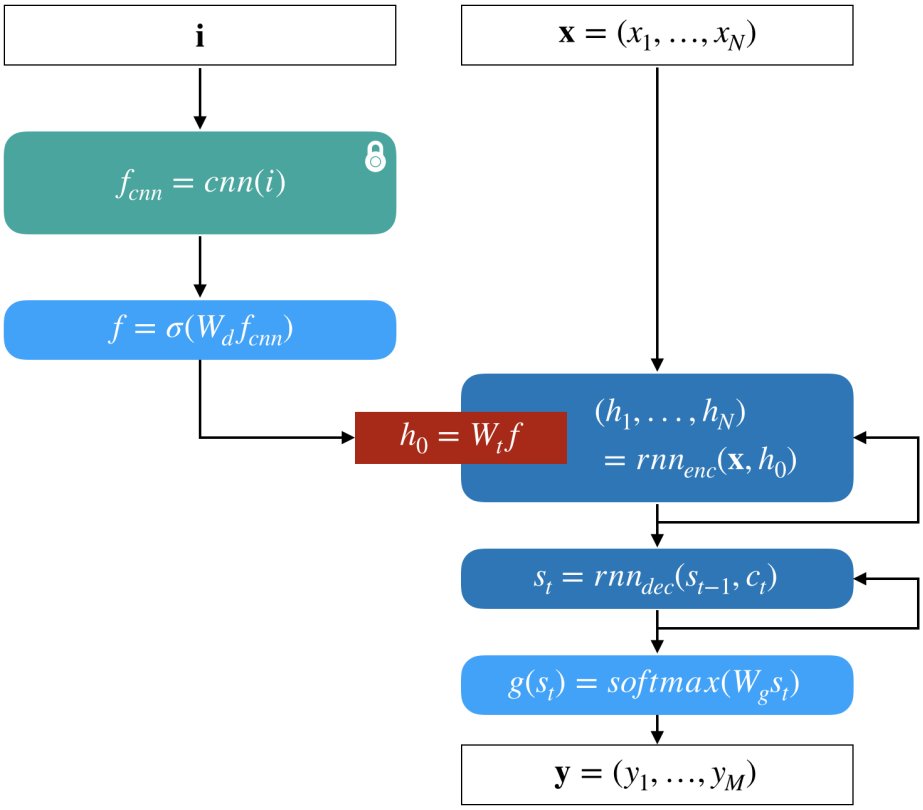



Figure 4.3.: Illustration of the FVTL model. It uses the (encoded) image representation as the initial RNN state of the encoder. To match dimensions, an additional linear transform layer $t$ is introduced. For simplicity's sake, the details of the bidirectional encoder and the attention mechanism are not shown.

### 4.3.3. Generator Merge (GM)

The GM model merges the textual and visual information at the very last step of the encoder-decoder framework: the generator. With the decoder producing the output $s_t$ at time $t$, the generator usually transforms this vector to a probability distribution over the output vocabulary set $V_{out} = \{y_1, \ldots, y_O\}$ according to (2.19). The generator is now modified to allow an extended input consisting of $s_t$ as well as the second encoders output $f$. This leads to:

$$g(s_t, f) = softmax\left(W_g \begin{pmatrix} s_t \\ f \end{pmatrix}\right)$$
$$\mathbb{P}(y_i | y_{t-1}, \ldots, y_1, , \mathbf{x}, \mathbf{i}) = g(s_t, f)[i]$$

This model design is chosen with the hope of the usual encoder-decoder architecture detecting uncertain situations (where the image must be used) and marking these in the $s_t$ in some way. The generator then should recognize these ambiguity signs and use the image representation to fill in the right word. That way, the ambiguity shall be resolved.

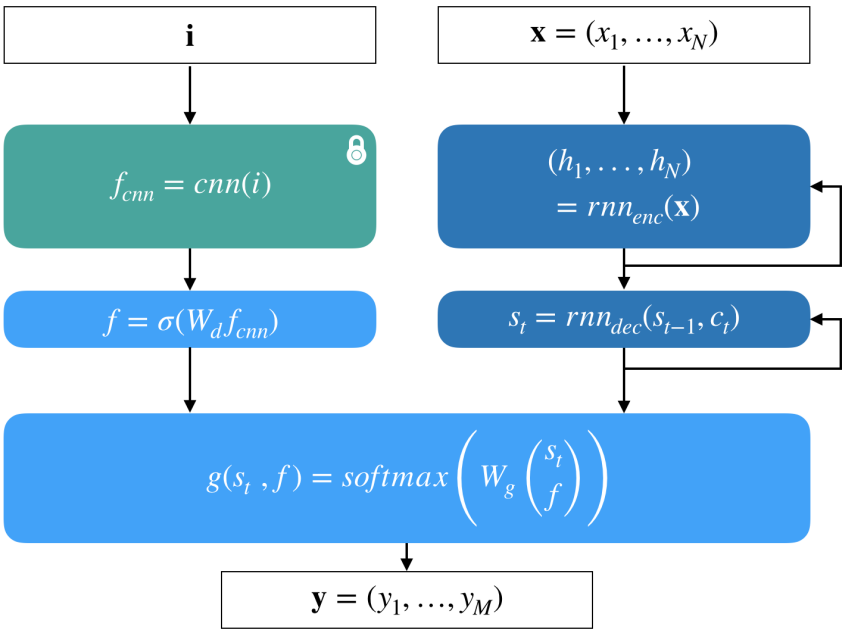



Figure 4.4.: The GM model, merging the second modality input at the last step before interpreting the decoders output as probability distribution over the target vocabulary.

### 4.3.4. Combination

The FVTL and GM architectures were combined to the FVTL+GM model. It uses the image representation $f$ as initial encoder input as described in section 4.3.2 and additionally supplies $f$ as supplementary input to the generator as explained in section 4.3.3. This approach could also be called "First view, then listen, finally view again". The hope is that by supplying the image in the initial as well as in the final layer will improve ambiguity resolving: When first looking at the image, some information may be captured and used during question encoding and answer decoding. Any uncertainties remaining should eventually be solved by the GM part of the model.

# 5. Evaluation

## 5.1. Dataset

**Dataset structure** The dataset used to train a multimodal dialog system needs to consist of images with corresponding dialog texts. Since manually taking images is by far more time-consuming than generating dialogs, each image was used for multiple dialogs. In the ARMAR-III example used throughout this thesis, an image defines a *scene* consisting of a set of objects and a set of drinks depicted on the image. The set of scenes to capture was created based on object count, combination of vessels and drinks and ambiguity situations (e.g. same object of different colors). Each scene was photographed multiple times from various perspectives by different cameras to increase abstraction capability of the model. Eventually, a grammar-based dialog generator was implemented to create many dialogs for each scene. See appendix A.2 for details on the implementation of this generator.

**Robot dialogs** When composing the robot dialog grammar, different possible commands leading to ambiguous situations were taken into focus. As a basic example, in the *grab* dialog, the user asks for a specific object ("Please give me the cup"). Depending on the given scene, the robot either has to respond instantly by generating an API call and confirming the request to the user or ask a further inquiry. In the latter case, the properties resolving ambiguity need to be extracted from the scene. After the user responds with his selection, the robot can finally generate an unambiguous API call and a response using this information.
Further dialogs were defined to increase complexity in ambiguity resolving:

- In the *grab unspecified* dialog, the user just asks for a "thing". The robot first has to inquire for the specific object type and afterwards, if necessary, for an unambiguous property.
- The *move* command contains an object to move and a place to move the object to. Further inquiry only has to be done for the object property, if necessary.
- The *move next to* command contains an object to move and another object to move the first one next to. This leads to four cases: Nothing has to be asked, only the object to move or only the object to place it next to has to be further specified, or both need further inquiry.

| | |
|---|---|
| User | *Please give me something to drink* |
| Robot | Do you want to drink beer or milk |
| User | *I want the beer please* |
| Robot | Do you want the beer served in the cup or mug |
| User | *The cup please* |
| Robot | API_CALL pour beer cup |
| Robot | Ok I am going to pour the beer into the cup |

Figure 5.1.: Example image and corresponding, exemplary *pour* dialog from the test dataset.

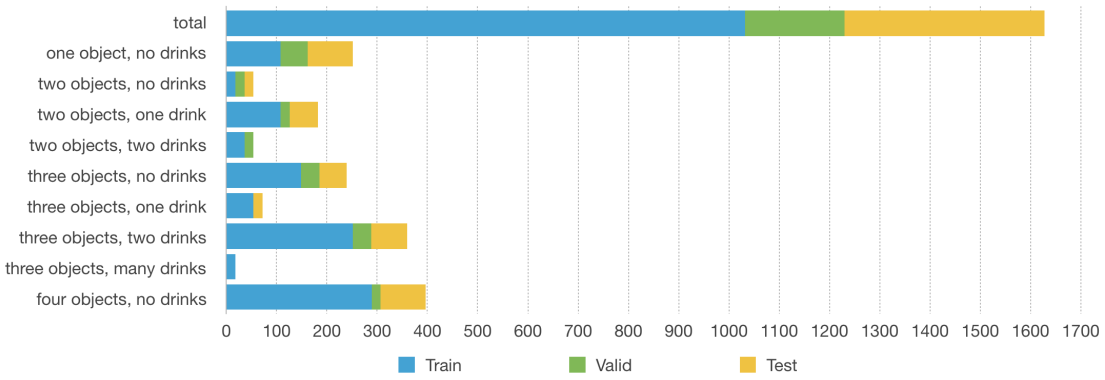



Figure 5.2.: Image count by complexity in each dataset split

- Involving drinks, the *pour* dialog has the highest complexity regarding dialog length. First, the user asks for "something to drink" and the robot has to present the drinks available. After the user chose his drink, the robot needs to ask which vessel to use (if there are multiple). Eventually, if the vessel is still ambiguous by only mentioning its type (e.g. if there are two cups of different color), the robot has to do a further inquiry for an ambiguity-resolving property.

See figure 5.1 or appendix B.1 for some example dialogs for different scenes.

The dialog generator was written with a *full* and *simple* mode. In *full* mode, a tremendous amount of non-qualitative permutations using different wordings, unnecessary words etc. is generated. Contrary, the *simple* mode only yields dialogs qualitatively distinguishable, i.e. dialogs with different user commands. For simplicity and training efficiency, the *simple* mode was used throughout this thesis. Nevertheless, enabling the dialog system to work with various variations in wording would be as simple as retraining in *full* mode.

**Dataset analysis** The image dataset was created based on randomly generated scene plans involving a predefined set of available objects. The train, validation and test datasets were also split at random: From the initially created dataset, scenes were chosen with a probability of 15 % to become part of the validation set, while the rest stayed in the train set, yielding train and valid dataset sizes of 57 and 11 scenes, respectively. Later on, a test scene plan of approximately the validation dataset size was generated randomly (excluding the scenes already part of the other datasets), making up 14 scenes. Figure 5.2 shows the train/valid/test data split. Special interesting scenes were added to the test dataset:

1. A scene with a never seen object (red plate) occurring neither in the train nor in the validation data. Nevertheless, the model should be able to recognize the object since it saw other objects of the same type (plates) and also knows the color (red).

2. Two scenes with another new object (yellow mug) in combination with various known objects.

3. Five additional *one object* scenes, although the scenes are already part of the train or validation set. This is necessary because the set of objects is rather small, and the *one object* setting allows no permutation. Despite the scenes already being part of the other datasets, the actual images differ in object position, light and perspective.

Because of this additions, the complete test dataset comprises 22 scenes and therefore is twice as big as the validation dataset with its 11 scenes.

Vessels (cups and mugs) appear more frequent than other objects (plates) since they lead to more interesting dialogs when combined with drinks (especially the *pour* dialog). None of the colors clearly dominates the other ones, except for orange, which was the color of three objects. Most objects were

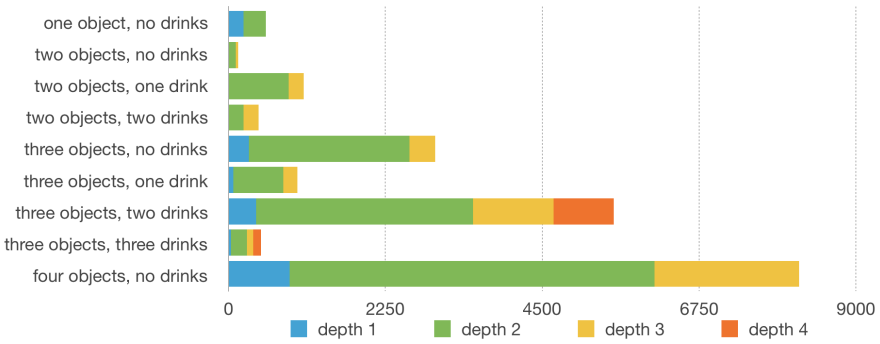



Figure 5.3.: Number of dialogs of different depths in train dataset, grouped by complexity level

stated to be "big", even though this is only a relative property: Only two of the nine objects (excluding drinks) were titled "small". Especially, there is an orange small cup and an orange big cup, which inquires whether the model can distinguish the "size" property. See appendix B.2 for detailed dataset statistics.

The scenes were classified using a simple complexity measure counting the number of objects and the number of drinks. This measure maps well to the dialog depth, defined as the number of command-response-pairs part of the dialog (see figure 5.3). Since certain complexity levels lead to much more permutations regarding the set of all possible scenes using the available objects, these classes form a higher percentage of the dataset (e.g. *three objects, two drinks* and *four objects*).

## 5.2. Image classifier results

Although the image classifier model described in section 4.1 is not the main part of this thesis, evaluating its performance is important since it provides an upper bound for the multimodal models which can only construct correct dialogs if the image content is recognized appropriately. The first experiment conducted compared the performance of different CNN model architectures trained using the simple stochastic gradient descent algorithm with a learning rate of 0.8 and scheduled learning rate reduction. Additionally, a comparison of nets trained only on the images taken by the robot's camera and nets trained on all images was done. Later on, more training algorithms like Adam and Adamax with different learning rates were examined.

For result evaluation, various measures were used: The loss after the last epoch on the train and valid dataset were utilized as first indicators of performance. From the test dataset, images depicting a set of objects $S$ were passed to the model and the resulting classification $C$ was calculated. An image was said to be classified "partly correct", if $S \cap C \neq \emptyset$ and "fully correct", if $S = C$. Note that the latter ones are also counted as "partly correct". Furthermore, the percentage of correct items $c$ and wrong items $w$ was calculated as

$$c = \frac{|S \cap C|}{|C|} \qquad w = \frac{|S \setminus C|}{|C|}$$

for each image, and eventually averaged over all samples. In combination, these metrics form a good view of the overall performance of the image classifier model, especially with $c$ and $w$ mapping to the precision and recall approach.

Across all evaluated train algorithms, results showed the "Squeeze-and-Excitation" CNN architecture ("SENet") of Hu et al. (2018) to outperform the other ones [1] at all metrics. Therefore, this model was

[1] tested architectures included VGGNet (Simonyan and Zisserman, 2014), AlexNet (Krizhevsky, 2014), Inception-v3 (Szegedy et al., 2015), Inception-ResNet (Szegedy et al., 2016), Caffeeresnet (He et al., 2016) and ResNeXt (Xie et al., 2017), each with different parameter configurations.

| Net | train loss | valid loss | partly correct | full correct | ∅*w* | ∅*c* |
|---|---|---|---|---|---|---|
| senet154 | 0.5930 | 0.6436 | **98.4 %** | 30.8 % | 37.0 % | 73.2 % |
| se_resnet50 | 0.6130 | 0.6620 | 88.3 % | 22.4 % | 36.7 % | 55.7 % |
| se_resnet101 | 0.6090 | 0.6587 | 92.9 % | 18.8 % | **33.4 %** | 61.0 % |
| se_resnet152 | 0.5868 | 0.6379 | 97.4 % | 32.1 % | 37.7 % | 74.5 % |
| se_resnext50_32x4d | **0.5852** | **0.6378** | 97.1 % | 34.4 % | 38.3 % | **77.9 %** |
| se_resnext101_32x4d | 0.5924 | 0.6410 | 89.0 % | **36.0 %** | 41.2 % | 71.3 % |
| cafferesnet101 | 0.6133 | 0.6463 | 89.0 % | 23.7 % | 38.3 % | 60.8 % |

Table 5.1.: Results of image classifier experiments with different CNN architectures trained using Adamax algorithm with a learning rate of 0.005. All except the first two columns refer to results on the test dataset.

used as image encoder for the multimodal networks. As expected, training on the complete dataset led to better generalization results compared to training solely on the images taken by the robot's camera, supporting the common data science approach of "the more data, the better". The best results in correct item percentage as well as train and valid loss were achieved with the Adamax training algorithm and the *se_resnext50_32x4d* architecture (see table 5.1). Since the validation dataset is used for model selection, this CNN architecture was chosen for the multimodal dialog tasks. Nevertheless, this model had a relatively poor wrong item percentage of 38.3 %. On the other hand, the model achieving the lowest wrong item percentage also suffered from low correct classification performance.

All in all, it can be stated that the available pretrained image recognition networks have difficulties recognizing the features needed for the kitchen objects classification task. No model shows outstanding performance in both precision and recall. With the best fully correct classification percentage of under 40 %, the image feature extraction network can be expected to form a bottleneck for the multimodal dialog task.

## 5.3. Multimodal model results

The different multimodal encoder-decoder architectures described in section 4.3 were trained using the multimodal robot dialog dataset. The baseline dialog net was trained with the same parameters on the text-only dataset. Each model was trained and evaluated independently five times in a row, choosing the model with the highest accuracy on the validation dataset for test evaluation. All the presented results refer to the median of these five runs.

Using the number of correct dialogs as the key metric to assess model performance is intuitive. A dialog is said to be correct if and only if all responses are identical to the target utterance. As this is a quite rigorous criteria, the number of correct lines (i.e. correct responses) was counted independently. To analyze further details and identify failure reasons, the ambiguity detection capability of the model was measured using two metrics: On the one hand, the number $a$ of API calls even though the situation is actually ambiguous, i.e. the target utterance contains a question, on the other hand, the complementary number $q$ of questions although the situation is unambiguous (i.e. the target utterance is an API call) is recorded. Both numbers are necessary, similarly to precision and recall (e.g. if a net always generates an API call, $q$ would be zero although the net is far from making good decisions). As the test dataset contains special scenes with unknown objects (see section 5.1), all metrics were evaluated once using the complete dataset and once excluding scenes with unseen items.

Concerning the type of input the multimodal models received as second modality, four different approaches were examined.

1. First, only the pretrained CNN was applied to the input image and the resulting features were fed to a trainable linear layer before passing them to the merging part of the different multimodal model architectures. This method, now called *features*, is exactly the one proposed in section 4.3, where $f = \sigma(W_d \cdot m)$ with the second modality input vector $m = m_f = cnn(\mathbf{i})$.

2. The second approach is contrary to the idea of end-to-end training, but was examined to partly avoid the image classifier bottleneck resulting from the relatively small set of training images. Thereby, the output of the *cnn* function was replaced with the output of the network pretrained on the image classifier task, resulting in the multimodal model receiving the object detection probabilities rather than the image features. This method can be expressed as $m = m_p = img\_classifier(m_f)$ and will be called *probabilities* from now on.

3. Moving further away from end-to-end training, the *classes* approach makes hard decisions based on the object detection probability vectors, i.e. it binarizes each entry $i$ to 1 if $o[i] > 0.5$ and 0 otherwise. This leads to $m = m_c = binarize(m_p)$.

4. Finally, the *ground-truth* approach uses the target ground-truth object classes $t$ instead of the image classifier output classes, i.e. $m = t$. While this is obviously a non-realistic method unsuitable for real-world applications, it is utilized here to completely eliminate the upper bound on performance caused by the insufficient CNN object recognition.

Figure 5.4 shows an overview of the performance of the tested multimodal model architectures across all approaches. Obviously, the non-realistic *ground-truth* approach outperforms the other ones since it does not suffer from the image processing bottleneck.

Binarization of object detection probabilities in the *classes* strategy seems to have a conflicting effect: While the FVTL and GM model remain approximately constant in comparison to *probabilities*, the performances of the FVTL+GM and HSM model slightly decrease. This could be explained by the split effect of binarization reinforcing detection certainty on the one hand but completely erasing too low probabilities on the other hand, therefore resulting in a neutral overall effect. Considering this explanation, the slight decrease of FVTL+GM and HSM is not significant and appears to be coincidentally.

Despite the image features not being adapted to the task in any way, the GM and FVTL+GM models achieve the best results across the three realistic methods using the *features* approach. This applies both for the complete test dataset as well as when excluding unknown object scenes. While this might seem surprising on the first look, it could be explained by using abstract image features being preferable

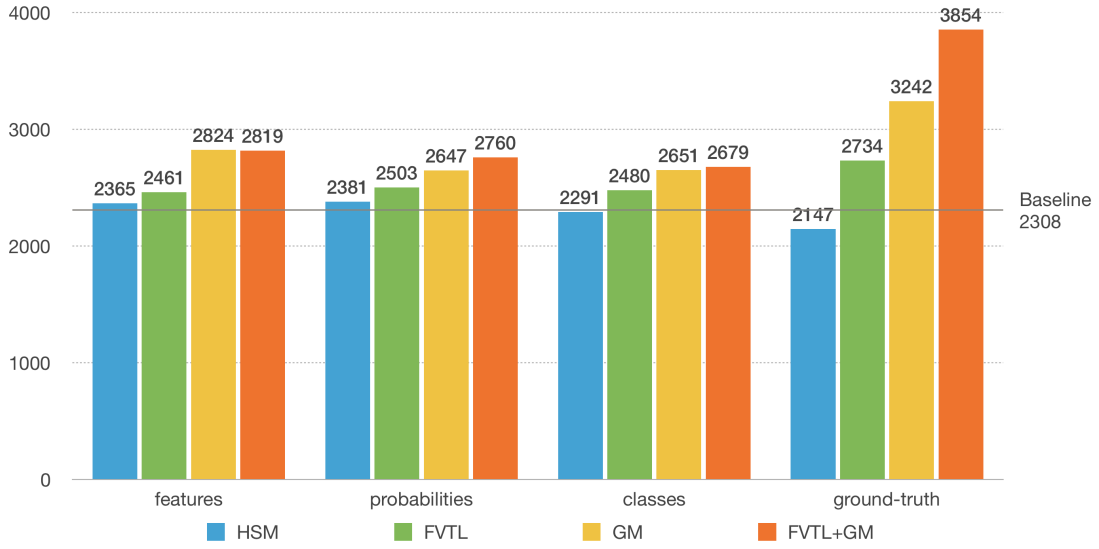



Figure 5.4.: Overview of the multimodal model performance on the complete test dataset (including unknown object scenes). The total number of dialogs is 6536. "Baseline" refers to the text-only dialog network.

over using explicit probabilities when combined with abstract sentence features in form of the decoder output. In contrast, the FVTL model performs slightly better using the explicit object detection probabilities or classes rather than abstract image features. Correspondingly, it merges the visual modality with explicit vocabulary sentence representations rather than abstract hidden states. While the rise of FVTL in the *probabilities* approach compared to *features* is not substantial, this hypothesis is especially supported by the HSM model's performance decreasing using the *ground-truth* strategy. Merging the explicit object classes with the abstract encoder hidden state seems to confuse the model, even if the classes are completely correct due to the unrealistic *ground-truth* approach. This suggests that combining different modality inputs on a similar abstraction level is preferable over mixing abstract and explicit representations.

Comparing the different architectures, the experiments show HSM to be inferior across all approaches. Merely outperforming the baseline, the idea of merging the visual modality with the encoder hidden state seems to be unsuitable for effectively using the image information. As HSM is probably not able to handle explicit object representations, future work should explore if improving the suitability of image features extracted from the CNN makes this architecture competitive with the other models. In contrast to that, supplying second modality input at the start or the end of encoder-decoder networks using the FVTL or GM architecture, respectively, yields results considerably beating the text-only model. As stated above, depending on the form of sentence representation when merging the second modality, the abstract *features* or concrete *classes* strategy is preferable. Reasonably, the combined FVTL+GM model wins at all approaches (except *features*, where it shares performance with GM). Particularly, taking a look at the *ground-truth* performance, a remarkable rise of 67 % in comparison to the baseline performance can be observed. Relative to the total number of dialogs, this is an improvement of 23.7 %. When switching to the more realistic *features* approach, the image bottleneck pulls down the performance betterment compared to the text-only model to 22.1 %, nevertheless making up an improvement of 7.8 % relative to all dialogs. Excluding the most complex "four objects" task, the increases become more notable: While the baseline achieves a score of 32.8 % of correct dialogs regarding this reduced dataset, the FVTL+GM model improves to 46.4 % using *features* and 66.1 % using *ground-truth*, leading to relative improvements of 41.2 % and 101.1 %, respectively. All these numbers verify that the combined model actually considers information provided by the second modality when generating dialog responses.

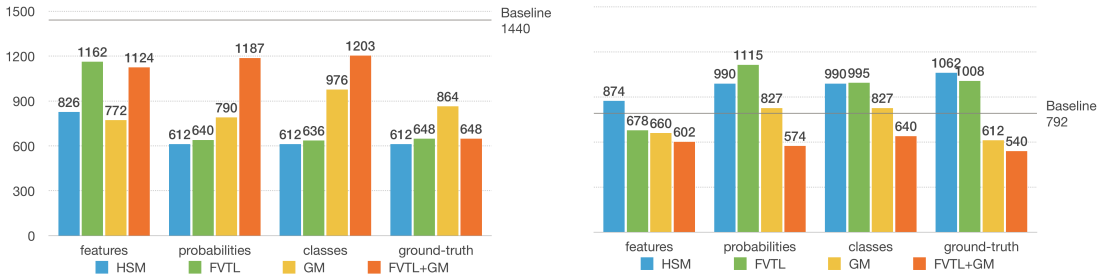



Figure 5.5.: Ambiguity detection statistics for the different multimodal models. Left: $a$ (API calls although ambiguous), right: $q$ (questions although unambiguous)

Taking a look at the ambiguity detection statistics in figure 5.5 provides further insights: The baseline model seems to focus on generating API calls instead of asking questions, shown by the low value of $q$ in contrast to the high value of $a$. Compared to that, all multimodal models lower their values of $a$ while some increase and some decrease their $q$. Finding a good balance between $a$ and $q$ while reducing the total number of wrong ambiguity detections $w = a + q$ should be the goal. The HSM model seems to avoid generating API calls and focus on asking questions, mapping to its low correct dialog performance across all approaches. Contrary, the GM model has the lowest number of $w$ for the *features* strategy, supporting it outperforming the other architectures there. Obviously, when using the *ground-truth* object classes, the best performing FVTL+GM model also shows the best ambiguity detection statistics, reducing $a$ by 55.0 % and $q$ by 31.8 % in comparison to the baseline, resulting in the total number of wrong ambiguity detections $w$ being approximately halved. Keeping with the more realistic *features* approach, the GM model still decreases $w$ by 35.8 %. These remarkable declines of wrong ambiguity detection results prove that the information provided by the second modality is actually used to detect and resolve ambiguity.

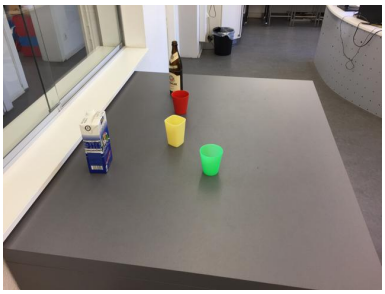
The results of the *features* approach outperform or are approximately equal to the *probabilities* models at all architectures. This is surprising, since the features are not adapted to the task in any way, while the probabilities are pretrained especially for recognizing the kitchen objects. As a consequence, the question was raised if using features is generally preferable over using probabilities or if the performance decrease (especially for the GM model) solely results from the poor performance on the image classifier task. To analyze this issue and make both approaches comparable, since it is impossible to avoid adapting the probabilities to the task, the features were adapted too. Fearing overfitting when training the complete CNN, another strategy was used: The image classifier network was modified to use two linear layers applied to the (unadapted) CNN features, each followed by a sigmoid function. The first layer reduced dimension from the CNN feature dimension (2048) to an arbitrary intermediate feature size (100), while the second layer mapped to the output probabilities (now called "two-layer probabilities" to distinguish from the original *probabilities* approach). Despite this modification not improving the image classifier task performance, its sole purpose was to evaluate the proposed question. Therefore, the dialog net was trained once using the output probabilities of this network as its second modality input and once using the intermediate features. Using this setup, the experiments led to both approaches approximately

| Architecture | intermediate features | two-layer probabilities | *features* | *probabilities* |
|---|---|---|---|---|
| HSM | 2277 (34.8 %) | 2129 (32.6 %) | 2365 (36.2 %) | 2381 (36.4 %) |
| FVTL | 2721 (41.6 %) | 2594 (39.7 %) | 2461 (37.7 %) | 2503 (38.3 %) |
| GM | 2716 (41.6 %) | 2675 (40.9 %) | **2824 (43.2 %)** | 2647 (40.5 %) |
| FVTL+GM | **2788 (42.7 %)** | **2864 (43.8 %)** | 2819 (43.1 %) | **2760 (42.2 %)** |
| Ø | 2626 (40.2 %) | 2566 (39.3 %) | 2.617 (40.0 %) | 2.572 (39.3 %) |

Table 5.2.: Results of the dialog net experiments with a two layer image classifier network, compared with the original results. As a reminder, the baseline achieved 2308 correct out of 6536 dialogs (35.3 %).

balancing out, with the intermediate features only excelling the two-layer probabilities by 0.9 % relative to all dialogs averaged over the four architectures and the combined FVTL+GM model keeping its top position (see table 5.2). Thereby, referring to the FVTL+GM model, the intermediate features drop by 0.5 % in comparison to the original *features* approach, while the two-layer probabilities performance raises by 1.6% in comparison to *probabilities*. All in all, no notable difference comparing task-adapted, intermediate features to the original, generic features can be observed. This indicates that the difference between *features* and *probabilities* is probably caused by the poor image classifier performance and none of the approaches is clearly favorable over the other based on the present results. Nevertheless, the *features* strategy is definitely the one to choose when increasing dataset variety, particularly when it is about handling unknown situations.

Test dataset scenes with unknown objects as described in section 5.1 were analyzed separately. Because these objects do not appear in the train or validation set, the *probabilities*, *classes* and *ground-truth* approaches have no chance to perform well. Solely the *features* models may have learned to connect color and object words to visual features. Nonetheless, the results show this is not the case (see figure 5.6 for a sample failure, table C.1 for the numbers): Except for some *grab unspecified* dialogs for which the unknown object can be inferred by only looking at the user utterance, no other dialogs involving unseen items are performed correctly. All model architectures appear to have the same poor performance of 18 correct out of 630 dialogs about unknown objects, being equal to the baseline results. Surprisingly, a non-*features* experiment (FVTL+GM *classes*) shows a slightly higher score on this task, but this turned out to be coincident as well. To conclude, the statistics state that the models cannot abstract from color or object words to a higher level concept. Despite knowing a yellow plate and a red mug, a yellow mug remains unfamiliar. This is probably the fault of too few training images, scenes especially designed to enable color and object abstraction would be necessary in particular. For example, to learn the meaning of "yellow", the train dataset would have to contain various objects of that color for the model to perceive the similarity between them. Exploring that abstraction capability is an interesting task up to future work.



| | |
|---|---|
| User | *Please move the mug to the table* |
| Robot | Do you want me to move the green or the red mug |
| Target | Do you want me to move the green, red or yellow mug |
| User | *The yellow one please* |
| Robot | API_CALL move mug yellow table |
| Robot | Ok I am going to move the yellow mug to the table |

Figure 5.6.: Failure example for a dialog about an unknown object. The unknown yellow mug is simply ignored when inquiring for the color. Output as generated by the FVTL *features* model.

# 6. Conclusion and future work

## 6.1. Conclusion

In this work, different architectures for integrating visual and textual inputs into an encoder-decoder network were proposed. In particular, the place where to combine both inputs in this framework was varied from the first step within the initial encoder input to the last step within the generator. Additionally, a combination of these architectures was examined.

The proposed models were evaluated on a sequence-to-sequence goal-oriented dialog task in a closed-world kitchen robot setting. Therefore, a multimodal dataset composed of images and corresponding dialogs was collected. By requiring many domain-specific pictures taken by hand together with hand-crafted dialog grammars to generate sample dialogs for each image scene, dataset acquisition required great effort. In return, the amount of work necessary to craft the actual dialog system diminished to the training time, which maps to the general neural network approach of trading program for dataset complexity.

Using the collected dataset, it could be shown that using a multimodal neural network actually improves the performance and usability of the examined goal-oriented dialog systems. More precisely, remarkable improvements of the best multimodal model in comparison to the text-only baseline were observed, showing that the models actually train to use the second modality information. Different approaches concerning the type of visual input for the dialog systems were compared, with the results indicating a similarity of inputs for both modalities is favorable. Especially, performance suffers from mixing abstract encoder hidden states with explicit object detection classes, even if these are completely correct due to the *ground-truth* approach.

Ambiguity detection and resolution as an abstract skill was examined in particular. The experiments showed that the image input is used by the multimodal models to considerably decrease both indicators for wrong ambiguity detection and reduce the total number of them by over 45 %. Although using pre-trained, state-of-the-art convolutional neural networks for image processing, object recognition appeared to be a bottleneck because of the relatively small amount of train images. To prove ambiguity detection capability despite this limitation, experiments with the processed image data replaced by the ground-truth object classes were conducted. This way, the best performing architecture improved by 67 % in comparison to the baseline, showing that the model actually learned to identify and resolve ambiguous situations using the visual information provided by the second modality.

## 6.2. Future work

This work presented four different multimodal model architectures, including a combined approach. More architectures could be developed and examined based on mutlimodal encoder-decoder models coming from MNMT. Putting focus on the attention mechanism like Calixto et al. (2017) did by attending to text and image separately is a promising approach, especially when the image dataset size and diversity increases. Considering fine-tuning of the pretrained CNN, i.e. not fixing all of its weights after a few training epochs, could enhance object recognition and therefore circumvent the image processing bottleneck spotted.

While this thesis showed that multiple modalities can be used to improve performance on a goal-oriented dialog task, future work exploring multimodal dialog systems should aim at widening the setting in comparison to the closed-world kitchen robot example used here. As a first step, the kitchen robot could be upgraded to a more general-purpose robot with more diversity in command structure and action

types. More ambiguous situations should be added, going beyond color and size towards more complex or indirect ambiguities. Furthermore, expanding the architecture to support parallel video and text streams would lead to a much bigger scale of real-world applications. For instance, a goal-oriented personal assistant dialog system could be trained with a visual sense to react to spoken phrases considering the emotional state of the user, detected using CNNs like the one of Jain et al. (2018).

To conclude, despite currently being in a very early stage, combining visual and textual features into a multimodal dialog system provides tremendous opportunities to improve user experience for goal-oriented tasks and build up the next generation of human computer interaction.

# Appendices

# A. Implementation

## A.1. Image acquisition

**Scene** To collect images of possible dialog scenes in a structured way, first, the concept of a "scene" had to be defined. In the ARMAR-III example used throughout this thesis, a scene is a set of objects and drinks placed on the kitchen table in front of the robot. Objects are composed of a name (e.g. "cup"), a color ("red") and a size ("big"). The set of object properties is designed to be easily extensible to allow posterior addition of other attributes such as material. Additionally, each object has a flag stating if it can be used to pour drinks in. The set of drinks is simply a set of strings representing their names (e.g. "water"). A simple script `read_available_objs.py` was implemented to parse a text file of a predefined syntax into a `Scene` object containing all available objects.

**Image plans** Since manually labeling all the captured images in hindsight would require a tremendous amount of time, plans for taking the images were created in advance. An image acquisition plan is simply a list of scenes to capture. To ease the process of capturing the images, an easy-to-read text file format was defined for representing image plans. Thereby, each line represents one scene to capture. To avoid listing all objects of a scene each time, only the difference to the previous scene is written down. Listing A.1 shows an example of this.

```
+ grey big cup (d) + blue big mug (d) + milk
+ cola + green small plate
+ beer
- milk
- blue big mug (d) + orange big mug (d)
- cola + milk
```

Listing A.1: Example of an image plan file. The + and - direct to add or remove the given item, while `(d)` indicates the given object can be used to pour drinks in.

Before each image acquisition session, plans were generated by a custom script. Initially, this script generates a list of interesting scenes based on object count, combination of vessels and drinks and ambiguity situations (e.g. same object of different colors). See section 5.1 for more details on "interesting" scenes. Afterwards, a greedy algorithm is used to sort the scenes so that the number of objects to change between each scene is minimized [1]. Eventually, an image diff-plan as explained above is created based on the sorted list of scenes.

During image sessions, these image plans were mostly obeyed. Nevertheless, since the diff-plan format allows for easy manipulations, this opportunity was sometimes used to spontaneously take some challenging images the generated plan did not mention (e.g. with three drinks at once).

## A.2. Dialog generation

**Grammar** To generate sufficiently complex dialogs, a small formal grammar framework was written.

Its base is a context-free grammar library consisting of the base type `Grammar` and its subtypes `Terminal`, `Or` and `Concat`. The framework heavily uses operator overloading to improve grammar

[1] This algorithm does not find optimal solutions, however this was not required either.

definition readability, i.e. instead of `Concat(expr1, expr2)` or `Or(expr1, expr2)` one can simply write `expr1 + expr2` or `expr1 | expr2`, respectively. Generation of all words defined by a context-free grammar is a simple task solved recursively.

Since a context-free grammar is not sufficient for generating multi-step dialogs, the framework was extended to allow back-references and context-dependent term choice. Therefore, the `Grammar`-subtype `Variable` with its specializations `IndependentVariable` and `BoundVariable` were introduced. An `IndependentVariable` behaves similar to an `Or`, except that each occurrence of the variable yields the same string, whereas an `Or` yields all permutations of its options if occurring multiple times. Bound variables can change their set of possible options depending on other variables. With this extension, the following steps are applied to generate all words of the grammar:

1. Sort the occurring variables topologically according to their dependencies.
2. For each variable, choose the next of its possible options (in nested loops, so that all permutations are hit once). Note that the set of possible options may change for bound variables depending on the value chosen for their dependencies, hence the topological sorting.
3. With all variables fixed to one value, the resulting grammar behaves like a context-free one. All words in this grammar are generated recursively.

The following example shows a simple usage of the described grammar framework:

```
T, V = Terminal, Variable
places = T("table") | "ground" | "fridge"
objects = T("cup") | "mug" | "plate"

place = V("place", places)  # First argument is the variable name
obj = V("object", objects)

move_cmd = "Please move the " + obj + " to the " + place
move_answer = "Ok, I am going to move the " + obj + " to the " + place
move_dialog = move_cmd + "\n" + move_answer

for w in generate_all_word(move_dialog):
    print(w)
```

Listing A.2: Example of grammar framework usage. Note that the occurrences of the variables will yield the same values, respectively. This example will generate dialogs of the form “Please move the cup to the table \n Ok, I am going to move the cup to the table” with all nine possible combinations of places and objects.

## A.3. Multimodal models

The multimodal encoder-decoder models were implemented as an extension of the OpenNMT framework by Klein et al. (2017). Initially, the `preprocess.py` script of OpenNMT was modified to allow two simultaneous input modalities. For space and computation efficiency as well as improved flexibility, the component responsible for converting images to tensors was made configurable to enable the definition of a custom image to tensor callback function. This mechanism was used to place the image features, detected classes or ground-truth classes in the dataset when implementing the different approaches. Therefore, the dataset files yielded by the preprocess step contained triples of source sentence, target sentence and second modality tensor.

The next step was to modify `train.py` to support multimodal models. This was done by additionally passing the second modality tensor to the model if present in the dataset. An abstract class

`MultiModalModel` was defined with an interface equivalent to `NMTModel`. The different model architectures described in section 4.3 were implemented as subclasses of `MultiModalModel`, with the FVTL+GM class being a subclass of both FVTL and GM. Finally, `translate.py` was adjusted to support multimodal models and datasets.

All changes made to OpenNMT were implemented in a generic way, neither specific to goal-oriented dialogs nor to images as second modality. In fact, the image to tensor callback stated above could be used to convert any file type to tensors, enabling the use of any other second modality like audio. Furthermore, `MultiModalModel` and its subclasses can be used for any sequence to sequence task, especially the initial purpose of OpenNMT, which is machine translation. To prove this generality, the framework as used in this thesis was trained on the multimodal machine translation dataset Multi30K by Elliott et al. (2016) and tested on the ambiguous COCO test dataset created for WMT17 (Elliott et al., 2017), reaching a BLUE score of 20.8 in comparison to the WMT provided baseline of 18.7.

All changes made to the OpenNMT framework will be open-sourced to make a contribution to research on multimodal sequence-to-sequence tasks. The following command line options were added to `preprocess.py` and `translate.py`:

**Image loading**

`-img_to_tensor_fn` A custom image file to tensor function which will be used to load images. Its type must be (str -> Tensor). The function should be given by its full name ("`module.submodule.function`"). Make sure the corresponding module is in `PYTHONPATH`! If this is not given, a default implementation (PIL + torchvision) will be used to read image files to tensors. (default: None)

**Multimodal data**

`-second_data_type` Type of the second source input. If this is not present, multimodal input is not used. If you supply this, you also have to supply `second_train_src`, `second_valid_src` and `src_dir` (for the images). Options are [img]. (default: None)

`-second_train_src` Path to the second modality training source data (listing file names inside of `src_dir`)

`-second_valid_src` Path to the second modality validation source data (listing file names inside of `src_dir`)

The following options were added to `train.py`:

**Multimodal model**

`-multimodal_type` The multimodal model implementation to use. If not specified, no multimodal model is used at all. Options:

- hsm = hidden state merge
- fvtl = first view, then listen
- gm = generator merge
- fvtl+gm = first view, then listen, finally view

`-second_dim_in` The input dimension of the second encoder.

`-second_dim` The output dimension of the second encoder.

# B. Dataset

## B.1. Examples

The following section presents four example images with corresponding exemplary dialogs from train and test datasets. At the bottom of each scene, the total number of dialogs without and with all possible word and sentence variants is shown. User utterances are shown in *italic*, robot utterances in normal letters.

| | |
|---|---|
| Dataset | Train |
| Camera | Canon EOS 7D |
| Scene | yellow big plate |

*Please give me the plate*
API_CALL grab plate
Ok I am going to give you the plate

*Please give me that thing over there*
What type of object are you talking about
*The plate please*
API_CALL grab plate
Ok I am going to give you the plate

*Please move the plate to the table*
API_CALL move plate table
Ok I am going to move the plate to the table

Total dialog count (simple / full): 5 / 1736

| | |
|---|---|
| Dataset | Test |
| Camera | Canon EOS 7D |
| Scene | orange small cup, orange big cup, milk |

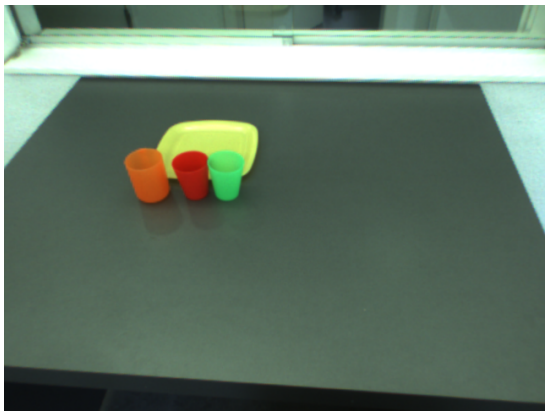
*Please give me the cup*
Do you want to have the big or the small cup
*The small one please*
API_CALL grab cup small
Ok I am going to give you the small cup

*Please give me that thing over there*
What type of object are you talking about
*The cup please*
Do you want to have the big or the small cup
*The small one please*
API_CALL grab cup small
Ok I am going to give you the small cup

*Please give me that thing over there*
What type of object are you talking about
*The small cup please*
API_CALL grab cup small
Ok I am going to give you the small cup

Total dialog count (simple / full): 10 / 4752

| | |
|---|---|
| Dataset | Train |
| Camera | ARMAR-III left eye |
| Scene | orange big cup, red big mug, green big mug, yellow big plate |



*Please give me the cup*
API_CALL grab cup
Ok I am going to give you the cup

*Please move the plate next to the cup*
API_CALL move_next_to plate cup
Ok I am going to move the plate next to the cup

*Please move the mug next to the plate*
Do you want me to move the green or the red mug
*The red one please*
API_CALL move_next_to mug red plate
Ok I am going to move the red mug next to the plate

*Please move the cup next to the mug*
Do you want the cup next to the the green or the red mug
*The green one please*
API_CALL move_next_to cup mug green
Ok I am going to move the cup next to the green mug

Total dialog count (simple / full): 28 / 8544

| | |
|---|---|
| Dataset | Test |
| Camera | iPhone 6 |
| Scene | orange small cup, red big mug, yellow big mug, milk, beer |

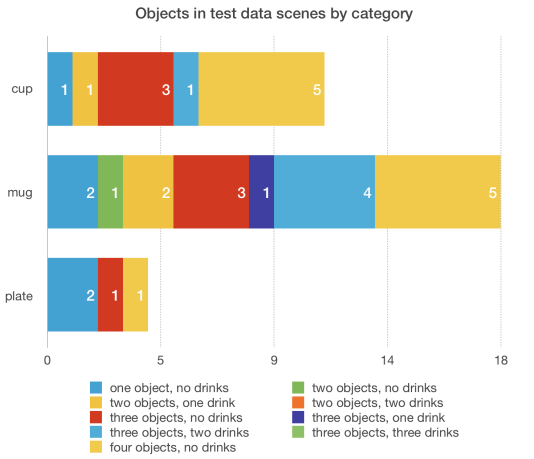
*Please give me something to drink*
Do you want to drink beer or milk
*I want the beer please*
Do you want the beer served in the cup or mug
*The cup please*
API_CALL pour beer cup
Ok I am going to pour the beer into the cup

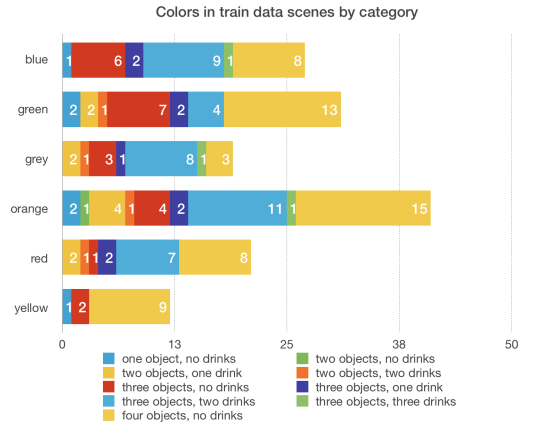
*Please give me something to drink*
Do you want to drink beer or milk
*I want the beer please*
Do you want the beer served in the cup or mug
*The mug please*
Do you want me to use the red or the yellow mug
*The red one please*
API_CALL pour beer mug red
Ok I am going to pour the beer into the red mug

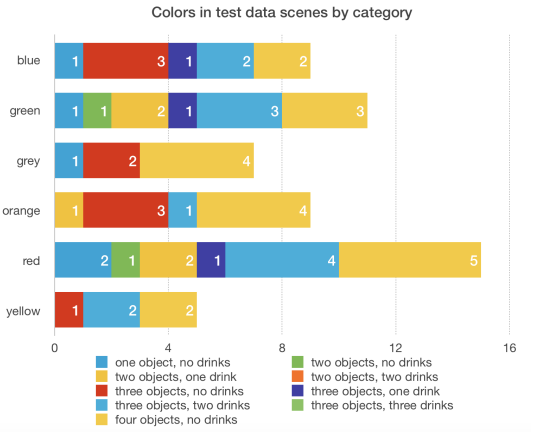
Total dialog count (simple / full): 23 / 966408

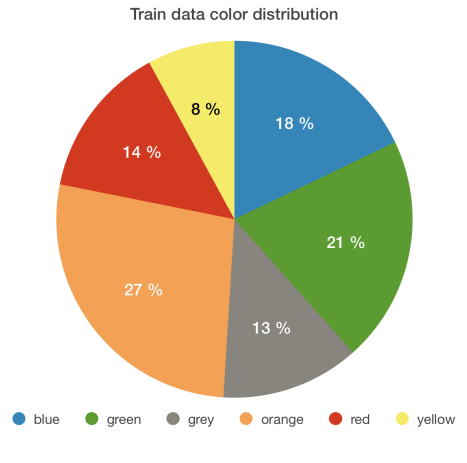
## B.2. Statistics

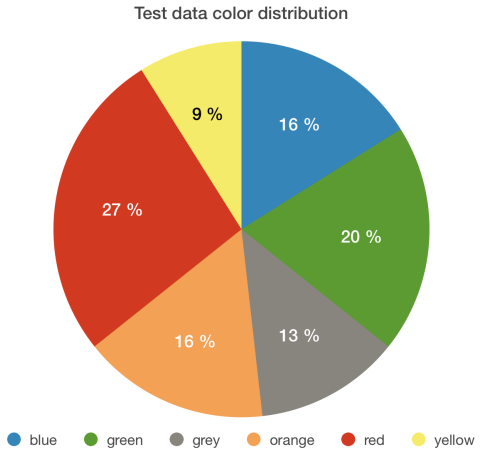
Supplementary figures for dataset analysis. See section 5.1.

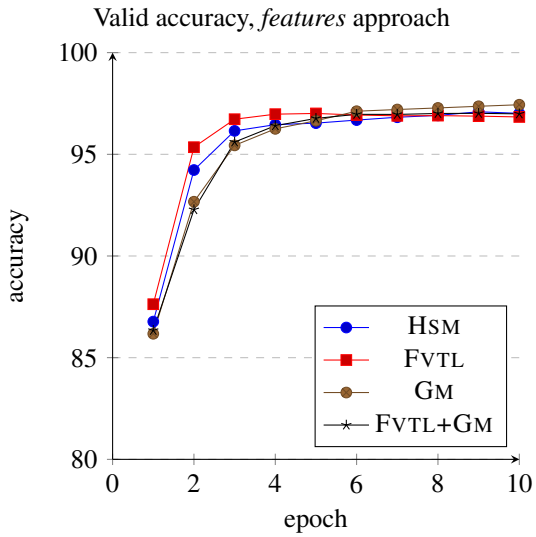



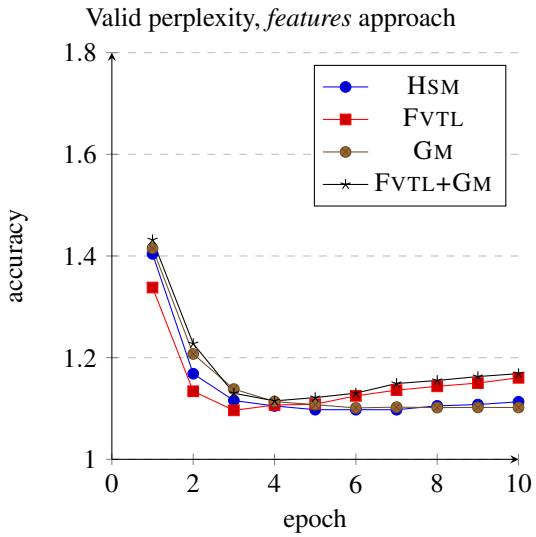



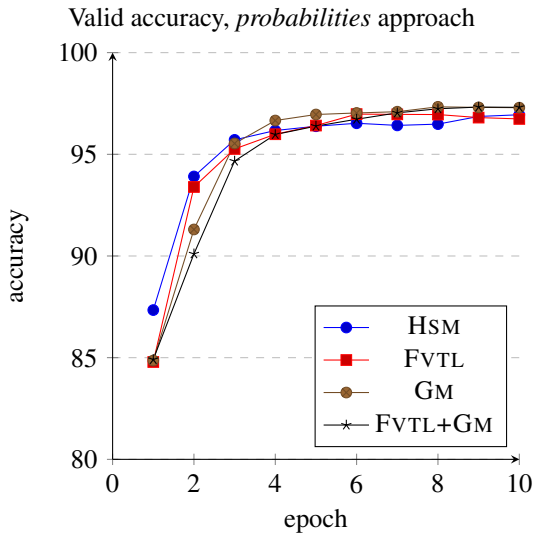
# C. Results

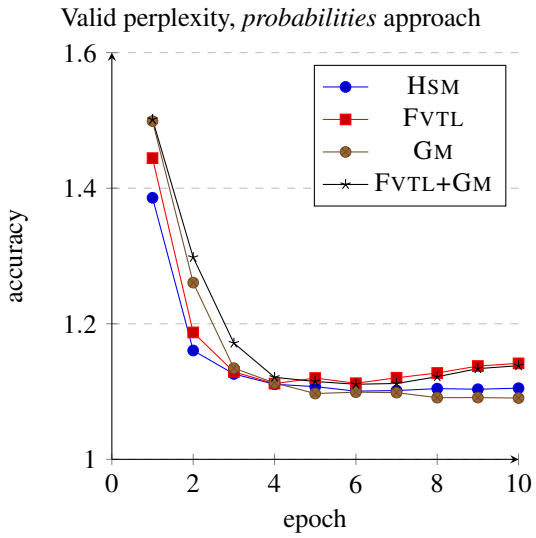
## C.1. Train curves

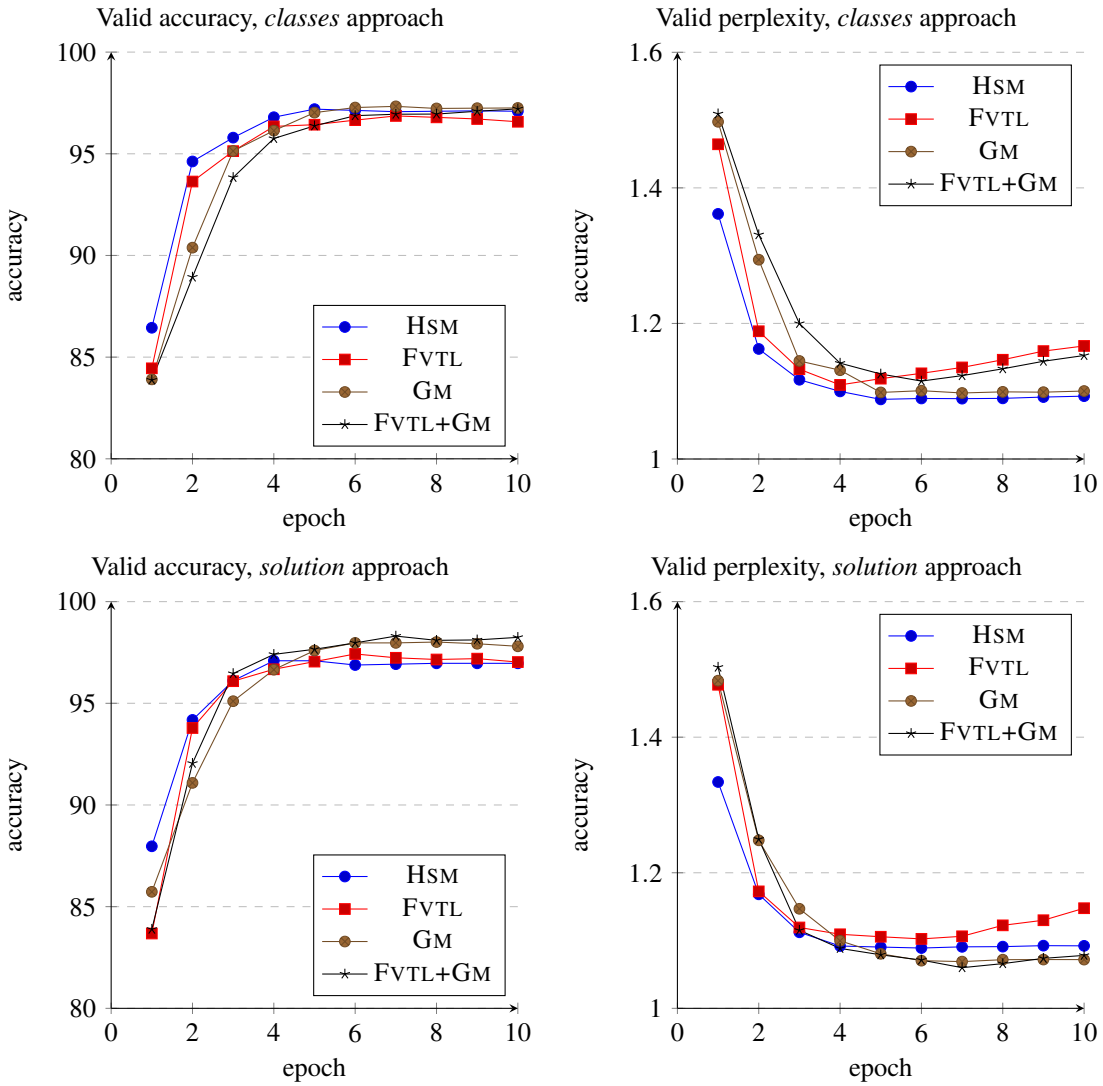
The following graphs show the course of validation accuracy and perplexity during training of the multimodal model architectures using the different approaches. Each model was trained five times, the graphs show an arbitrary one of the train courses. For test evaluation, the result of the epoch with the highest validation accuracy was chosen.






## C.2. Evaluation details

Table C.1 shows the detailed evaluation results for the multimodal model experiments. All nets were trained using the following parameters:

| | |
|---|---|
| Encoder | bidirectional |
| Optimizer | Adagrad |
| Learning rate | 0.1 |
| Learning rate decay | 0.75 |
| Adagrad accumulator init | 0.1 |
| Epochs | 10 |
| RNN size | 256 |
| RNN type | LSTM |
| RNN layers | 2 |
| Use input feed | true |
| Attention | MLP |
| CNN (does not apply to the raw model) | se_resnext50_32x4d |
| Second modality dimension ($S$ from section 4.3) | 256 |

| Architecture | Approach | $l$ | % | $d$ | % | $d_{excl}$ | % | $d_{unknown}$ | % | $a$ | $q$ | $w$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Total | | 14264 | | 6536 | | 5420 | | 630 | | | | |
| Raw | | 8880 | 62.3 % | 2308 | 35.3 % | 1859 | 34.3 % | 18 | 2.9 % | 1440 | 792 | 2232 |
| HSM | *features* | 9006 | 63.1 % | 2365 | 36.2 % | 2061 | 38.0 % | 18 | 2.9 % | 826 | 874 | 1700 |
| | *probabilities* | 8862 | 62.1 % | 2381 | 36.4 % | 2039 | 37.6 % | 18 | 2.9 % | **612** | 990 | **1602** |
| | *classes* | 8880 | 62.3 % | 2291 | 35.1 % | 2039 | 37.6 % | 18 | 2.9 % | **612** | 990 | **1602** |
| | *ground-truth* | 8718 | 61.1 % | 2147 | 32.8 % | 1895 | 35.0 % | 18 | 2.9 % | **612** | 1062 | 1674 |
| FVTL | *features* | 9210 | 64.6 % | 2461 | 37.7 % | 2058 | 38.0 % | 18 | 2.9 % | 1162 | 678 | 1840 |
| | *probabilities* | 8826 | 61.9 % | 2503 | 38.3 % | 2223 | 41.0 % | 18 | 2.9 % | 640 | 1115 | 1755 |
| | *classes* | 9145 | 64.1 % | 2480 | 37.9 % | 2226 | 41.1 % | 18 | 2.9 % | 636 | 995 | 1631 |
| | *ground-truth* | 9126 | 64.0 % | 2734 | 41.8 % | 2357 | 43.5 % | 18 | 2.9 % | 648 | 1008 | 1656 |
| GM | *features* | **9652** | **67.7 %** | **2824** | **43.2 %** | **2461** | **45.4 %** | 18 | 2.9 % | **772** | 660 | **1432** |
| | *probabilities* | **9443** | **66.2 %** | 2647 | 40.5 % | 2297 | 42.4 % | 18 | 2.9 % | 790 | 827 | 1617 |
| | *classes* | **9300** | **65.2 %** | 2651 | 40.6 % | **2277** | **42.0 %** | 18 | 2.9 % | 976 | 827 | 1803 |
| | *ground-truth* | 10206 | 71.6 % | 3242 | 49.6 % | 2749 | 50.7 % | 18 | 2.9 % | 864 | 612 | 1476 |
| FVTL+GM | *features* | 9437 | 66.2 % | 2819 | 43.1 % | 2412 | 44.5 % | 18 | 2.9 % | 1124 | **602** | 1726 |
| | *probabilities* | 9344 | 65.5 % | **2760** | **42.2 %** | **2343** | **43.2 %** | 18 | 2.9 % | 1187 | **574** | 1761 |
| | *classes* | 9204 | 64.5 % | **2679** | **41.0 %** | 2251 | 41.5 % | **19** | **3.0 %** | 1203 | **640** | 1843 |
| | *ground-truth* | **11024** | **77.3 %** | **3854** | **59.0 %** | **3351** | **61.8 %** | 18 | 2.9 % | 648 | **540** | **1188** |

Table C.1.: Overview of the multimodal model experiment results (on the test dataset). $l$: correct lines, $d$: correct dialogs, $d_{excl}$: correct dialogs excluding scenes with unknown objects, $d_{unknown}$: correct dialogs about unknown objects, $a$: API-calls even though ambiguous, $q$: questions even though unambiguous, $w$: wrong number of ambiguity detections ($= a + q$). The best of each approach is highlighted.

# Bibliography

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). VQA: visual question answering. *CoRR*, abs/1505.00468. 11

Asfour, T., Regenstein, K., Azad, P., Schroder, J., Bierbaum, A., Vahrenkamp, N., and Dillmann, R. (2007). Armar-iii: An integrated humanoid platform for sensory-motor control. pages 169 – 175. 2

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473. 9, 10

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, Berlin, Heidelberg. 4

Bordes, A. and Weston, J. (2016). Learning end-to-end goal-oriented dialog. *CoRR*, abs/1605.07683. 5, 6, 2, 11

Britz, D., Goldie, A., Luong, M.-T., and Le, Q. (2017). Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906. 10, 13

Caglayan, O., Barrault, L., and Bougares, F. (2016). Multimodal attention for neural machine translation. *CoRR*, abs/1609.03976. 11, 12

Calixto, I., Liu, Q., and Campbell, N. (2017). Doubly-attentive decoder for multi-modal neural machine translation. *CoRR*, abs/1702.01287. 12, 26

Chung, J., Çaglar Gülçehre, Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555. 7, 8

Constantin, S., Niehues, J., and Waibel, A. (2018). An end-to-end goal-oriented dialog system with a generative natural language response generation. *CoRR*, abs/1803.02279. 5, 6, 2, 11

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*. 13

Elliott, D., Frank, S., Barrault, L., Bougares, F., and Specia, L. (2017). Findings of the Second Shared Task on Multimodal Machine Translation and Multilingual Image Description. In *Proceedings of the Second Conference on Machine Translation*, Copenhagen, Denmark. Association for Computational Linguistics. 32

Elliott, D., Frank, S., Sima'an, K., and Specia, L. (2016). Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics. 32

Hastie, T., Friedman, J., and Tibshirani, R. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, USA. 6

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume abs/1512.03385. IEEE. 11, 20

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. diploma thesis. Master's thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München. 7

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780. 7

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. 20

Huang, P.-Y., Liu, F., Shiang, S.-R., Oh, J., and Dyer, C. (2016). Attention-based multimodal neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*. Association for Computational Linguistics. 11, 12

Jain, N., Kumar, S., Kumar, A., Shamsolmoali, P., and Zareapoor, M. (2018). Hybrid deep neural networks for face emotion recognition. *Pattern Recognition Letters*. 27

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. (2017). OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*. 5, 6, 13, 31

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997. 20

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324. Institute of Electrical and Electronics Engineers (IEEE). 11

Libovický, J., Helcl, J., Tlustý, M., Bojar, O., and Pecina, P. (2016). CUNI system for WMT16 automatic post-editing and multimodal translation tasks. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, volume abs/1606.07481. Association for Computational Linguistics. 12

Malinowski, M., Rohrbach, M., and Fritz, M. (2015). Ask your neurons: A neural-based approach to answering questions about images. *CoRR*, abs/1505.01121. 11

Mysid and Dave (2006). Neural network. *https://commons.wikimedia.org/w/index.php?curid=1412126*. 6

Olah, C. (2015). Understanding lstm networks. *http://colah.github.io/posts/2015-08-Understanding-LSTMs*. 7, 8

Olah, C. and Carter, S. (2016). Attention and augmented recurrent neural networks. *Distill, http://distill.pub/2016/augmented-rnns*. 9

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster r-CNN: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149. 12

Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681. 10

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556. 20

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215. 8

Szegedy, C., Ioffe, S., and Vanhoucke, V. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261. 20

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567. 20

Taspinar, A. (2016). The perceptron. *http://ataspinar.com/2016/12/22/the-perceptron*. 5

Vinyals, O. and Le, Q. V. (2015). A neural conversational model. *CoRR*, abs/1506.05869. 11

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1990). Phoneme recognition using time-delay neural networks. In Waibel, A. and Lee, K.-F., editors, *Readings in Speech Recognition*, pages 393–404. Elsevier, San Francisco. 11

Xie, S., Girshick, R., Dollar, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 20

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3320–3328, Cambridge, MA, USA. MIT Press. 11, 13

Young, S., Gasic, M., Thomson, B., and Williams, J. D. (2013). POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179. 11