Contextual Slot Filling for Task-Oriented Conversational Agents

Erik Stammes
10559736

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisors
Prof. Khalil Sima'an
Dr. Miguel Angel Ríos Gaona

Institute for Logic, Language and Computation
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 29th, 2018

Abstract

Slot filling aims to extract the relevant values of certain types of slots from a given utterance in a dialogue. Previous research has either used probabilistic models, requiring little data and having an explicit feature extraction step, or neural networks, using vector representations of words and achieving state-of-the-art results. In this thesis, this problem is solved by using a neural model with two specific properties: (1) a memory component, giving the model unrestricted access to previous utterances, effectively adding context to the current input utterance. (2) an explicit feature extraction step that uses the dialogue acts as features. The resulting model and two baseline methods are evaluated on a multi-turn dialogue dataset, outperforming both by using the benefits of both probabilistic models and neural networks. The model requires little data and uses an explicit feature extraction step, a memory component and word embeddings. It performs significantly better than the baselines on smaller dataset sizes, allowing the model to be used in practice when building task-oriented conversational agents.

Acknowledgements

First, I would like to thank Miguel Angel Ríos Gaona for his guidance throughout this process. He was available when needed and provided me with excellent guidance and feedback during our meetings and e-mail sessions. From the beginning he pointed me in the right direction and provided me with valuable resources.

I would also like to thank Frank Smit for asking me to write a thesis on this topic. Even though he has the busiest schedule of the people I know, he was available when needed and helped steer the project in the direction that we could both benefit from it. A honourable mention to the other people at OBI4wan too, thanks for the help.

Last, I would like to express my gratitude to Khalil Sima'an for taking on this thesis project which fell outside of the regular thesis schedule.

Contents

1 Introduction

Task-oriented conversational agents are chatbots that are designed for a particular task, for example to help a customer order a service, and set up to have short conversations to reach a goal. Current state-of-the-art agents can handle increasingly advanced tasks, such as finding, creating and managing reservations for flights. They are built end-to-end, also generating responses, using deep learning and therefore require a lot data (Serban et al. 2016). A system that requires little data is a prerequisite for building conversational agents efficiently, but end-to-end systems are prone to generating unexpected responses when there is little data available. Systems that do have this property are limited in the range of words they can use as input and therefore misinterpret utterances that are dependent on the context.

There are a few different steps in building task-oriented conversational agents. First the input needs to be processed, in case of a spoken conversational agent this would require a speech recognition component, when the input is text a spell check would suffice. In this thesis, the work is restricted to written dialogues in the form of chats between a user and a conversational agent. The dialogue is then to be further processed by a natural language understanding component, classifying the domain, intent and slots of the utterance. The slots are defined in a frame for each different task the agent supports, specifying attributes that are needed to accomplish the task. Using the frame with filled slots, a dialogue engine can then query a database to retrieve some information or ask the user for more information. The response is then generated using all previously gathered information and it aims to complete the required task as soon as possible.

This thesis focuses on slot filling, the task of filling the predefined slots in a frame based on a dialogue. For example, the following user utterance:

```
hi - buy 6 tickets for gimme danger at cinelux plaza theatre
```

Should result in the following frame with some filled slots:

```
NUM_TICKETS:    6
MOVIE:          gimme danger
THEATRE_NAME:   cinelux plaza theatre
TIME:           ?
DATE:           ?
```

(Example from Shah et al. (2018))

The slot filling can be reduced to a sequence tagging problem, tagging the appropriate words in each sentence with their slot names. Commonly, the words are tagged using the IOB scheme, tagging each word as either the beginning (B) or inside (I) of each slot label or outside (O) of any slot label. The following user utterance is tagged using the IOB scheme.

```
O   O   O    O B-category O          O  B-location I-location O
can you find a vietnamese restaurant in los        altos      ?
```

Ideally, a conversational agent system requires little data to train but can still take context from previous utterances into account when filling slots. Therefore, the following research question is investigated:

How can the slot filling problem for task-oriented agents be solved using little data?

The proposed solution to this problem is a neural network-based machine learning approach that stores previous utterances and dialogue acts in a special memory component, effectively using context and important speech features to model the dialogue.

The outline of this thesis is as follows: In the following section background information, including related work and relevant algorithms, on the slot filling problem for conversational agents are highlighted. In section 3 the methods, dataset and evaluation metrics that are used in this thesis are introduced. Section 4 shows experiments that were performed and discusses the results. Last, section 5 concludes this thesis with some final remarks and a discussion of future work.

2 Background

This section motivates the approach taken by exploring related work and highlighting the algorithms that are used in this thesis.

2.1 Related work

The task of filling slots in frames was first introduced by Bobrow et al. (1977) in the influential GUS system for travel planning. The GUS-style frame is still being used in state-of-the-art research and software packages to track information in dialogues. Bobrow et al. used hand-written rules and regular expressions to fill slots, while some later approaches used full grammars with thousands of rules (Ward and Issar 1994). These rule-based approaches have the advantage of high precision but they are slow to create and can suffer from low recall. Modern slot filling systems almost always use supervised machine learning methods, which require a training dataset to be available. An important dataset in the development of the slot filling problem is the ATIS dataset (Hemphill, Godfrey and Doddington 1990), which consists of single queries in the air travel domain. It was the first dataset to define common evaluation metrics and a common test set. Machine learning methods that were used on the slot filling problem were generative models such as hidden Markov models (Pieraccini et al. 1992) and maximum entropy Markov models (McCallum, Freitag and Pereira 2000), discriminative classification methods (Kuhn and De Mori 1995) and Conditional Random Fields (Raymond and Riccardi 2007). These methods often use an explicit feature extraction step, for example extracting n-grams and character features. Another approach uses a data oriented parsing method to parse the utterance in the context of the dialogue, not filling slots but parsing the sentence and outputting an update expression which specifies which slots should be updated with what values (Sima'an 2004). Recently, neural network approaches have been proposed to tackle sequence tagging problems, such as convolutional (Collobert et al. 2011), recurrent (Mesnil et al. 2013), long short-term memory (Huang, Xu and Yu 2015) and dynamic memory (Kumar et al. 2016) networks, these approaches often do not make use of an explicit feature extraction step. Common to all previously mentioned approaches, vector representations of words can be used to compare the meanings of input words and

	king	*queen*	*woman*	*princess*
royality	0.99	0.99	0.02	0.98
masculinity	0.99	0.05	0.01	0.02
$feminity$	0.05	0.93	0.99	0.94
age	0.70	0.60	0.50	0.10
⋮	⋮	⋮	⋮	⋮

Figure 1: Simplified word embedding with explicit dimensions as rows.[1]

sentences instead of their bag-of-words or one-hot representation.

2.2 Word Representations in Vector Space

Word representations in vector space, or word embeddings, can be used to measure similarity between words or even to do vector arithmetic. The theory is based on the distributional hypothesis in linguistics which states that words that are used and occur in the same contexts tend to purport similar meanings (Harris 1954). Figure 1 shows a simplified word embedding with explicit dimensions as rows, in real world applications the dimensions are always implicit. By representing words as vectors from this matrix, words that are similar in meaning are close to each other in terms of cosine distance. Now vector arithmetic can be used on words, resulting in expressions like $\text{vec}(king) - \text{vec}(male) + \text{vec}(female) \approx \text{vec}(queen)$, where $\text{vec}(w)$ is the vector representation of word w.

Word representations can be learned through both count-vector-based distributional semantic approaches and context-predicting approaches. Mikolov et al. (2013) have introduced two methods to learn word embeddings using the latter approach, Continuous Bag-of-Words (CBOW) and Skip-gram (SG), both denoted by word2vec. CBOW uses sentences from large corpora of words as input, where it tries to predict the current word based on the four words before and four words after the current word using a neural network with one hidden layer and an output layer. The training objective is to maximise the conditional probability of observing the actual output word given the input context, which uses two weight matrices. One between the input and hidden layer, and one between the hidden and output layer, which can be converted into a word embeddings matrix with a softmax function. The SG model also uses sentences from large corpora of words as input, but it tries to predict the context based on the current word. The neural network is similar to the CBOW approach and has the training objective of minimising the summed prediction error across all context words.

Updating weights for every word in a training set of billions of words is a very expensive process, so this is mitigated by using two optimisation techniques: hierarchical softmax and negative sampling. Hierarchical softmax is an efficient way of computing softmax by using a binary tree to represent all words in the vocabulary. Each word is represented as leaf in the tree, which has a unique path from the root, which can be used to estimate the probability of the

[1] Example from `https://blog.acolyer.org/2016/04/21/the-amazing-power-of-word-vectors/`

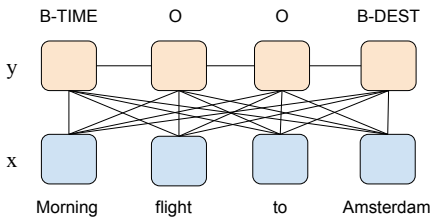

Figure 2: Linear chain Conditional Random Fields

leaf word. This reduces the computational complexity of computing the softmax from $\mathcal{O}(V)$ to $\mathcal{O}(\log_2(V)$, with V being the vocabulary size. Negative sampling is a method to update only a small portion of the weight matrices for each training sample, which decreases the computational complexity. The portion that gets updated is selected using a unigram distribution over the word counts, where common words are more likely to be chosen. Combined, these optimisation technique drastically reduced the computational complexity of learning high dimensional word vectors on a large amount of data compared to previous methods.

Pennington et al. (2014) use a count-vector based approach called GloVe, stating that word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves. Since vector spaces are linear structures, the most natural way to encode the information present in a ratio between words in vector space is with vector differences. The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. To do this, word co-occurrences are counted in a large matrix, which is then reduced in size by dimensionality reduction. The resulting word embeddings significantly outperform word2vec on word analogy tasks and are slightly better in word similarity and Named Entity Recognition (NER) tasks.

As NER is the most similar task to slot filling, the difference in performance between the two methods will have to be examined, which is reported in section 4.1.

2.3 Conditional Random Fields

Conditional random fields (CRFs) are undirected graphical models that model the conditional probability of a sequence of output labels (states) $\mathbf{Y} = \{y_1, y_2, \ldots, y_t\}$ given a sequence of observations $\mathbf{X} = \{x_1, x_2, \ldots, x_t\}$ (Lafferty, McCallum and Pereira 2001). CRFs are discriminative, whereas the previously commonly used Hidden Markov Models (HMMs) and Maximum Entropy Markov Models (MEMMs) are generative. Generative models explicitly attempt to model the joint probability distribution $p(\mathbf{Y}, \mathbf{X})$ over inputs and outputs, while discriminative models model the conditional distribution $p(\mathbf{Y}|\mathbf{X})$ directly. The main advantage of this difference is that dependencies that involve only variables in $\mathbf{X}$ play no role in conditional models, resulting in a much simpler structure compared to the joint model.

Graphical models are probabilistic models that express the conditional de-

pendence between random variables using vertices and edges. CRFs are undirected graphical models, as opposed to the directed graphs in HMMs and MEMMs. By using undirected graphical models CRFs are not biased towards states with fewer successor states. Figure 2 shows the graphical structure of a CRF model with an example input and output sequence.

CRFs take context into account when making predictions by using feature functions, defined as

$$f(y_t, y_{t-1}, \mathbf{X}) = 1_{\{z=z'\}}$$

where t is the current index in the observations, y_t is the label at index t, $\mathbf{X}$ is the vector that contains all components of the global observations and $1_{\{z=z'\}}$ denotes an indicator function of z which takes the value 1 when $z = z'$ and 0 otherwise. Each feature function f_k is assigned a weight $\mathbf{W}_k$ which the algorithm is going to learn.

$$p(\mathbf{Y}|\mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \mathbf{W}_k f_k\left(y_t, y_{t-1}, \mathbf{X}\right)\right\} \tag{1}$$

where $Z(\mathbf{X})$ is an instance-specific normalisation function

$$Z(\mathbf{X}) = \sum_{\mathbf{Y}} \prod_{t=1}^{T} \exp\left\{\sum_{k=1}^{K} \mathbf{W}_k f_k\left(y_t, y_{t-1}, \mathbf{X}\right)\right\}$$

This is known as *linear chain* CRF, because the feature functions only have access to current and previous labels, rather than arbitrary labels throughout the sentence. Because of this property exact inference is possible using the forward-backward algorithm.

To use the forward-backward algorithm, equation 1 can be rewritten as

$$p(\mathbf{Y}|\mathbf{X}) = \frac{1}{Z(\mathbf{X})} \prod_{t=1}^{T} \Psi_t\left(y_t, y_{t-1}, \mathbf{X}\right)$$

where $\Psi_t\left(y_t, y_{t-1}, \mathbf{X}\right)$, the transition weight, is defined as

$$\Psi_t\left(y_t, y_{t-1}, \mathbf{X}\right) = \exp\left\{\sum_{k} \mathbf{W}_k f_k\left(y_t, y_{t-1}, \mathbf{X}\right)\right\}$$

Now, using a set of forward variables $\alpha_t(j) = p\left(x_{\langle 1\ldots t\rangle}, y_t = j\right)$ and backward variables $\beta_t(i) = p\left(x_{\langle t+1\ldots T\rangle}|y_t = i\right)$, the following functions can be used to calculate $Z(\mathbf{X})$ recursively.

$$\alpha_t(j) = \sum_{i \in S} \Psi_t\left(j, i, x_t\right) \alpha_{t-1}(i)$$

$$\beta_t(i) = \sum_{j \in S} \Psi_{t+1}\left(j, i, x_{t+1}\right) \beta_{t+1}(j)$$

To obtain the marginal distributions $p(y_{t-1}, y_t|\mathbf{X})$ the results of both recursion functions can be combined and normalised. Next, to calculate the most

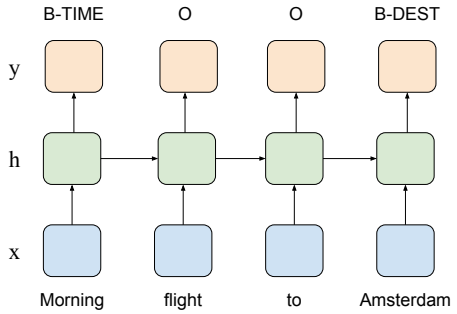

Figure 3: A Recurrent Neural Network (RNN)

probable label using the marginal distribution all probabilities of each label sequence must be calculated, which increases exponentially with the number of labels and the sequence length. Luckily, the Viterbi algorithm can be used to conveniently find the most probable label using another recursion function:

$$\delta_t(j) = \max_{i \in S} \Psi_t\left(j, i, x_t\right) \delta_{t-1}(i)$$

Linear-chain CRFs have been applied to many sequence labelling tasks even though they only capture local structure. Skip-chain CRFs can have connections between items that are more distant from each other in the input sequence, alleviating the problem, but coming with computational costs and incompatibility with exact inference.

2.4 Long Short-Term Memory Networks

Long-Short Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) that have cells with input, output and forget gates that allow the network to remember values over arbitrary time intervals (Hochreiter and Schmidhuber 1997). First, figure 3 shows the structure of an RNN with an input layer x, hidden layer h and output layer y (Elman 1990). In the context of slot filling x represents input features and y represents a probability distribution over output tags, both at time t. The connection between states in the hidden layer h gives the network short-term memory using recurrent layer weight parameters. These values are computed as follows:

$$h(t) = \sigma(\mathbf{U}x(t) + \mathbf{W}h(t-1))$$

$$y(t) = \text{softmax}(\mathbf{V}h(t))$$

where $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{V}$ are weight matrices to be learned at training time, and $\sigma(z)$ and $\text{softmax}(z)$ are the sigmoid and softmax activation functions respectively.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\text{softmax}\left(z_m\right) = \frac{e^{z_m}}{\sum_k e^{z_k}}$$

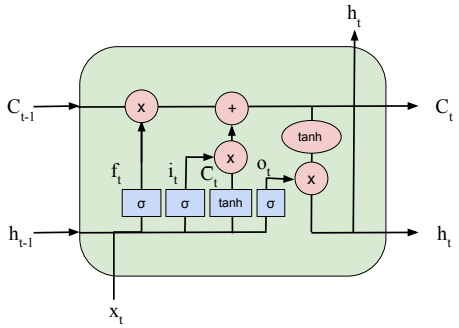

Figure 4: A Long Short-Term Memory (LSTM) cell

LSTM networks are augmented RNNs, the cells in the hidden layer h now have input, output and forget gates whereas the cells in RNNs only have a single *tanh* function connecting the input to other cells and the output. Furthermore, LSTM networks have a connection C_t between the states in each hidden layer cell. Figure 4 shows the structure of an LSTM cell, where f_t is the forget gate, i_t is the input gate, $\bar{C}_t$ is a vector of new candidate values, C_t is the new cell state, o_t is the output gate, and h_t is the output information of the cell. They are computed as follows:

$$f_t = \sigma\left(\mathbf{W}_f \cdot [h_{t-1}, x_t] + b_f\right)$$

$$i_t = \sigma\left(\mathbf{W}_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\bar{C}_t = \tanh\left(\mathbf{W}_C \cdot [h_{t-1}, x_t] + b_C\right)$$

$$C_t = f_t * C_{t-1} + i_t * \bar{C}_t$$

$$o_t = \sigma\left(\mathbf{W}_o\,[h_{t-1}, x_t] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Where $\mathbf{W}$ are weight matrices and b are bias vectors to be learned at training time. In this structure, the forget gate decides what information is deleted, the input gate decides what new information will be stored in the cell state and the output gate decides what information will be emitted from the cell.

Two LSTM layers can be combined to created a single, bidirectional LSTM network (Schuster and Paliwal 1997). Figure 5 shows such a network, the first layer trains on the input sequence as is, while the second layer trains on the input sequence in reversed order, giving the network access to information from the future. This allows the network to learn additional information from the input observation, which is beneficial in problems where future observations contain valuable information about the current observation. When tagging slot values in sentences future words can have lots of information about the current word, for example when a number is followed by the token `pm`, the number is always the value of the slot `time`.The slot filling problem for conversational agents therefore can benefit from the bidirectional nature of this LSTM network.

In natural language problems the input sequences are often encoded in a bag-of-words scheme where each word is represented in large sparse vectors.

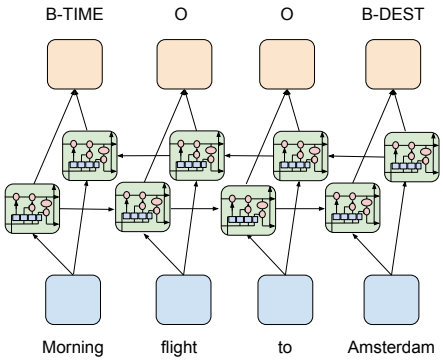

Figure 5: A bidirectional LSTM network

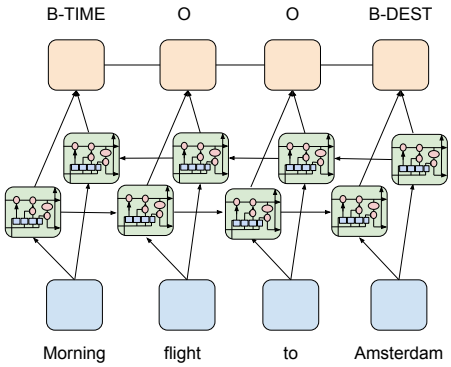

Figure 6: A bidirectional LSTM network with a CRF layer on top

As described in section 2.2, words can also be represented in smaller, dense vectors containing continuous representations of words. When the dataset is large enough, these word embeddings can be learned as part of the model, resulting in tailor made word representations but a slower learning process. On the other hand, pre-learned language specific word embeddings can also be used, requiring no data and no time to learn, but without tailored word representations.

LSTM networks can be further extended with an additional CRF layer, allowing the network to learn and use label transition probabilities on top of the bidirectional input observations (Huang, Xu and Yu 2015). Figure 6 shows a bidirectional LSTM network with a CRF layer on top, this layer makes the network less dependent on word embeddings, producing accurate results in natural language tasks without needing word embeddings.

The values in the weight matrices and bias vectors in LSTMs, and neural networks in general, can be learned by first defining a loss function, which measures the error in the output sequences. Common loss functions for LSTM networks that solve sequence tagging problems are cross entropy and mean squared error. These loss functions quantify the quality of the weight matrices, making it possible to optimise the weights using an optimisation step. Optimisation is done by repeatedly calculating the gradient of the loss function and then performing

a parameter update, this is called gradient descent. The gradient is the direction in which the loss function has the biggest rate of change, but to actually update the weight parameters we need to set the step size, or learning rate, of that direction. The learning rate is a setting that should be chosen carefully, a too high learning rate will overstep the optimal value of the loss function, while a too low learning rate will take very long to find that optimal value. Often the learning rate decreases while training the network, allowing the network to learn quick initially but still find the optimal weights later.

Training the neural network is often done in batches: selecting a subset of the training set, predicting the outcomes, computing the loss and updating the weight matrices. When one complete pass through the dataset has been completed, this is called an epoch.

Neural networks, like other machine learning methods, are prone to overfitting, following the data in the training set too closely and not being able to generalise to unseen data. Regularisation aids the problem of overfitting, and there are several methods to do it. L1 regularisation is a technique that lets the network focus on the most important inputs, while becoming nearly invariant to noisy inputs. L2 regularisation penalises weight vectors with too high values and prefers weight vectors with diffuse values. Dropout is a regularisation technique that drops weight updates by setting the probability of keeping them, an effective way to prevent overfitting.

LSTM networks have been successfully applied to a variety of NLP tasks: speech recognition, machine translation, document categorisation and sequence tagging. However, LSTMs tend to be biased towards neighbouring items and performance degrades as the lengths of the input sequences increase (Cho et al. 2014).

2.5 Memory Networks

Memory networks have inference components combined with a long-term memory component that can be read and written to, allowing them to inference outputs using a large memory (Weston, Chopra and Bordes 2014). Memory networks can be built using multiple machine learning methods, including support vector machines, decision trees and neural networks. When memory networks are built using neural networks and learned end-to-end they are called end-to-end memory networks (MemN2N) (Sukhbaatar, Weston, Fergus et al. 2015). These networks consist of a set of input and output memory representations with memory cells. The input m_i and output c_i memory cells are obtained by transforming the input context $\mathbf{X} = \{x_1, ..., x_t\}$ using two embedding matrices A and C such that $m_i = A\Phi(x_i)$ and $c_i = C\Phi(x_i)$, where $\Phi(\cdot)$ is a function that maps the input to a feature vector. The input observation q is encoded in similar fashion $u = B\Phi(q)$ using another embedding matrix B. Now, a vector of attention weights p_i can be computed using the input memories m_i and the embedding of the input observation u:

$$p_i = \text{softmax}(u^T m_i)$$

The response o from the output memory can then be computed using the weighted sum

$$o = \sum_i p_i c_i$$

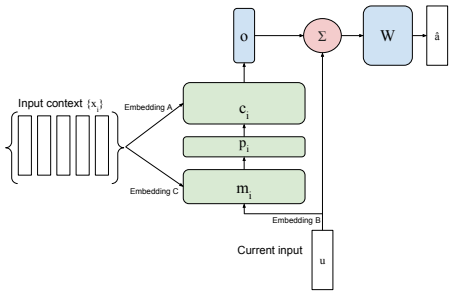

Figure 7: An end-to-end memory network with a single hop

In problems that require access to more memory the memory network model can be extended by stacking a number of memory layers. Each memory layer is called a hop and the $(k+1)^{th}$ hop takes as input the output of the previous hop k^{th}:

$$u^{k+1} = o^k + u^k$$

Now, an output distribution based on the input observation q can be computed using another softmax function

$$\hat{a} = \text{softmax}(\mathbf{W}(o^K + u^K))$$

where $\mathbf{W}$ is a weight matrix to be learned at training time. Figure 7 shows such an end-to-end memory network with a single hop.

End-to-end memory networks are trained in the same fashion as other neural networks, requiring a loss function, an optimisation method, a learning rate and adopting techniques such as learning rate decay and regularisation.

Memory networks outperform both RNNs and LSTMs in long-term memory encoding, achieving state-of-the-art performance in question answering tasks, sentiment analysis and part-of-speech tagging (Kumar et al. 2016). Memory networks have also been shown to perform well on the task of slot filling for conversational agents in a multi-domain setting (Bapna et al. 2017) and for knowledge carryover (Chen et al. 2016).

3 Method

This section discusses the model used to optimise the context for conversational agents, the dataset that was used and the preprocessing and feature engineering that were done. The model is then compared to both a CRF and bidirectional LSTM with CRF and is shown to outperform both at different dataset sizes.

3.1 Model

The model used in this work is a Memory-Enhanced Conditional Random Field (ME-CRF), taking inspiration from memory networks by integrating a memory component into a neural network (Liu, Baldwin and Cohn 2017). On top of that, it adds a CRF layer which takes as input the output of the memory layer, thereby allowing the model to have unrestricted access to the whole sequence.

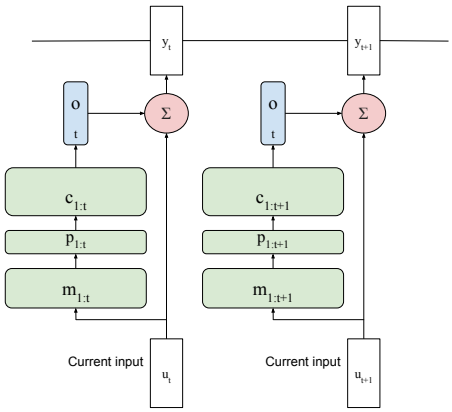

Figure 8: Simplified Memory-Enhanced Conditional Random Field model at time t and $t+1$, embeddings are not shown for simplicity.

This type of model has been shown to achieve state-of-the-art performance on named entity recognition and thread discourse structure prediction tasks. Figure 8 shows the network architecture at time t and $t+1$ without the embeddings, as they are left out for simplicity reasons. Compared to neural memory networks, the output of $u_t^k + 1 = o_t^k + u_t^k$ is directly fed into a CRF layer instead of a softmax over weight matrix $\mathbf{W}$. Furthermore, a temporal signal has been injected into the memory using a bidirectional GRU encoding, which mitigates the issue of insensitivity to temporal information between memory cells (Xiong, Merity and Socher 2016). GRU cells are similar to LSTM cells, only replacing the input and forget gates of the LSTM with a single reset gate.

The input memory cell m_i can now be computed as follows:

$$\overrightarrow{m}_i = \overrightarrow{GRU}\,(x_i, \overrightarrow{m}_{i-1})$$

$$\overleftarrow{m}_i = \overleftarrow{GRU}\,(x_i, \overleftarrow{m}_{i+1})$$

$$m_i = \tanh(\overrightarrow{\mathbf{W}}_m \overrightarrow{m}_i + \overleftarrow{\mathbf{W}}_m \overleftarrow{m}_i + b_m)$$

where $\overrightarrow{\mathbf{W}}_m$, $\overleftarrow{\mathbf{W}}_m$ and b_m are weight matrices and a bias vector to be learned at training time. The CRF layer uses the forward-backward algorithm to calculate the conditional probability $p(\mathbf{Y}|\mathbf{X})$ and at test time, the model predicts the output using the Viterbi algorithm.

The model is trained using the Adam technique which reduces the cross entropy loss (Kingma and Ba 2014). The input words are converted to dense vector representations by using pretrained word embeddings, which are kept fixed while training the model. The results are evaluated on the test set, while early stopping is used based on the results of the development set.

3.2 Data and Preprocessing

The parameters in the ME-CRF model can be learned by using a dataset with annotated slot values. To show that the model handles context better than previous methods the simulated dialogue dataset is used because it consists of

Dataset	Slots	Train	Development	Test
sim-R	category, rating, date, location, restaurant_name, time, num_people, price_range, meal	1116	349	775
sim-M	date, theatre_name, time, num_tickets, movie	384	120	264

Table 1: Simulated dialogue dataset details

multi-turn dialogues (Shah et al. 2018). This dataset is preferred over other multi-turn dialogues, such as the Ubuntu Dialogue Corpus (Lowe et al. 2015) which contains only unlabelled data and the Frames dataset (Asri et al. 2017) which focuses on the task of tracking frames instead of slots.

The simulated dialogue dataset was created by first generating outlines using dialogue self-play and then rewriting the template utterances using crowd sourcing. It consists of 3008 dialogues in two domains: booking a restaurant table (`sim-R`) and buying a movie ticket (`sim-M`). Each domain has its own set of slots as highlighted in table 1. `sim-M` has five different slots, whereas `sim-R` has nine slots to be filled. Across the 3008 dialogues, there are a total of 27120 utterances with 173576 tokens, of which 20526 tokens are annotated with a slot name.

The dataset is available through JSON files, which consist of a list of `dialogue`s with `turn`s with user and system utterances. Each utterance consists of a `text`, `tokens` and `slots` field. The `slot` field is a list of slots with their name, start index and end index, which makes it possible to convert this dataset to the IOB-style tagging scheme.

The first step in preprocessing is tagging each token with its part-of-speech (POS) tag, which is done by using the POS tagging component from spaCy[2]. As this is a resource intensive process, this is done once and the POS tags are added to the original JSON files. Upon loading the dataset into the model the dataset is further processed: features are extracted and the slot names are tagged using the IOB-tagging scheme.

To test the results of the model at different dataset sizes a number of datasets were created by randomly selecting dialogues from the training sets. The original training set sizes are 1116 (sim-R) and 384 (sim-M), datasets of 500, 250, 150, 100, 50, 20, and 10 dialogues were created and validated by checking if all possible slot values are present.

3.3 Feature Engineering

The original ME-CRF model has a feature extraction structure that adds features for each token, for example a feature that indicates whether the token contains only digits. This structure was extended by adding a start of sentence symbol (`<S>`) to which features can be added that apply to the whole input sentence. The model can then use these features by addressing that part of the

[2]NLP software package, retrieved from `http://spacy.io`

memory. One of such features is the turn type feature, indicating whether the current utterance is from the user or system.

The POS tags that were previously added to the dataset are now converted into features, specific subsets of POS tags were selected and indicate whether the current token is a verb, noun, symbol or punctuation mark.

On a word level, the word patterns are analysed on character type level, converting words to their pattern. Letters are converted to `a` or `A` depending or their capitalisation and digits are converted to `0`. The resulting word patterns are encoded in a one-hot vector. For example, both `the` and `who` are converted to `aaa`, while `3pm` is converted to `0aa` before being encoded into a one-hot vector.

In some task-oriented dialogue domains certain slots will always contain values of the same format, such as order numbers, postal codes and bank account numbers. These tokens can be matched using regular expressions, for example the regular expression `/[\d]{4} ?\w{2}/` matches Dutch postal codes consisting of four digits, optionally a space and then two letters.

3.3.1 Dialogue acts

An important insight in the structure of conversations is that each utterance in a dialogue is a kind of action performed by the speaker (Austin 1975). These acts are called dialogue acts, and they express the intention of the speaker in saying what they said. A dialogue is not merely a series of independent dialogue acts, but a collective act performed by both parties to establish a common ground. People need closure for acts, making it clear that the parties have understood each others meaning and intention. Dialogue acts can therefore be of great value in the slot filling problem for conversational agents, using the different kinds of dialogue acts to improve the understanding of the utterances.

Each utterance in the simulated dialogue dataset has one or more associated dialogue acts, providing the underlying meaning of an utterance. For example when a system utterance has the associated dialogue act `NOTIFY-SUCCESS`, the system tells the user that the actions with a certain slot name and value have been completed successfully. Listing 1 shows how such a system utterance with its associated dialogue acts is represented in the dataset. Table 2 shows the possible dialogue acts and their meaning.

```
"system_acts": [
   {
       "slot": "num_tickets",
       "type": "NOTIFY_SUCCESS",
       "value": "2"
     },
     {
       "slot": "date",
       "type": "NOTIFY_SUCCESS",
       "value": "tomorrow"
     }
]
```

Listing 1: Example of the dialogue acts associated with the system utterance "2 tickets for tomorrow have been purchased"

Dialogue Act	Speaker	Description
GREETING	User/System	Greet the other speaker
INFORM	User/System	Inform a slot value
CONFIRM	User/System	Ask the other speaker to confirm a given slot value
REQUEST	User/System	Ask for the value of a slot
REQUEST_ALTS	User	Ask for more alternatives
OFFER	System	Offer a database entity to the user
SELECT	System	Offer more than one database entity to the user
AFFIRM	User/System	Agree to something said by the other speaker
NEGATE	User/System	Disagree to something said by the other speaker
NOTIFY_SUCCESS	System	Notify the user of a successful event, e.g. a booking is complete
NOTIFY_FAILURE	System	Notify the user of a failure event, e.g. a booking isn't available
THANK_YOU	User/System	Thank the other speaker
GOOD_BYE	User/System	Say goodbye to the other speaker
CANT_UNDERSTAND	User/System	Tell the other speaker that their utterance was not understood
OTHER	User	Unknown utterance type

Table 2: List of dialogue acts

The dialogue acts `REQUEST`, `SELECT`, `CONFIRM` and `OFFER` in system utterances provide valuable information about the next user utterance. For example, when the system requests a value for the slot `time` ("At what time would you like to make a reservation?") the dialogue act would be `request(time)` and the next user utterance will likely include the value for that slot. This information is encoded in one-hot feature vectors for the aforementioned dialogue acts for each slot name and for each utterance. As all other features, this is saved to the memory so it can be used when tagging the next user utterances.

Only the dialogue acts in system utterances are used, as the system would need a dialogue act classifier to get the dialogue acts from unseen user utterances. When building conversational agents it is possible to specify the dialogue act for each system utterance that will be generated, thus this information can be used.

3.4 Implementation

The ME-CRF model is implemented using TensorFlow[3], based on the implementation by the authors of the original ME-CRF model. That implementation is focused on the task of Named Entity Recongition (NER), which is a sequence tagging problem and can thus be adapted to the slot filling problem. The feature extraction step was modified to make it possible to add sentence level features such as the dialogue acts. The simulated dialogue dataset is in a different format compared to the CoNLL 2003 dataset for NER, which was originally used in the ME-CRF model. Therefore, the data preprocessing step was modified to correctly format the data, including converting the tags to the IOB-tagging scheme.

4 Experiments and Results

This section highlights the experiments that were performed using two baselines: CRF and bidirectional LSTM with CRF, and the ME-CRF model. Also, the

[3]Python library for dataflow programming, retrieved from `https://www.tensorflow.org`

parameter tuning and evaluation process are explained. Last, the results of the experiments are shown.

4.1 Experimental Setup

The simulated dialogue dataset has not been used for the slot filling problem for conversational agents before, therefore the ME-CRF cannot be compared to any previous results by other authors. Hence, two baseline methods were implemented: a CRF model and a bidirectional LSTM with CRF layer (BiLSTM-CRF).

4.1.1 CRF

The open source natural language understanding package Rasa NLU[4] provides a Python implementation to train a CRF model for slot filling. This implementation uses CRFsuite[5] under the hood, which implements multiple optimisation algorithms to calculate the state and transition probabilities, since it not only allows linear-chain CRFs but also general CRFs. Rasa uses the limited-memory BFGS algorithm for training and the Viterbi algorithm for prediction by default. The following features are used:

- Boolean indicating whether a word is lowercase
- Boolean indicating whether a word is uppercase
- Boolean indicating whether a word is titlecase
- Boolean indicating whether a word is a digit
- The first 3 and 5 letters of a word
- The last 2, 3 and 5 letters of a word

The simulated dialogue dataset was converted to a new input scheme, specific to Rasa NLU, while the output was modified so it could be evaluated using the method from section 4.3.

4.1.2 BiLSTM-CRF

An open source bidirectional LSTM with CRF layer[6] for NER was adapted to use the simulated dialogue dataset as input. The model does not have an explicit feature extraction step besides word and character embeddings. To make the comparison to the ME-CRF model fair, the character embeddings were disabled. Training is done using the Adam optimiser, which reduces the cross entropy loss from the CRF layer and predictions are computed using the Viterbi algorithm. The learning rate is set to 0.001 with a decay of 0.9, while a batch size of 20 and dropout rate of 0.5 were used. The hidden layer size of the word embeddings LSTM cell is 300, while the word embeddings themselves have a dimension of 50.

The simulated dialogue dataset was converted to the input scheme required by this implementation and the output was modified so it could be evaluated using the method from section 4.3.

[4]Retrieved from `https://nlu.rasa.com`
[5]`http://www.chokkan.org/software/crfsuite/`
[6]Retrieved from `https://github.com/guillaumegenthial/sequence_tagging`

4.2 Parameter Tuning

Parameters were optimised using grid search with the following parameter sets: learning rate $\in \{0.001, 0.01, 0.1\}$, gradient clipping $\in \{1, 5, 10\}$ batch size $\in \{16, 32, 64\}$, dropout keep probability $\in \{0.5, 0.8, 1\}$ and as non-linearity function either the hyperbolic tangent ($tanh$) or a Rectified Linear Unit (ReLU). The grid search was performed on the sim-M dataset with 20 dialogues with the goal of searching for the best parameters when there is limited data and computing power available.

The learning rate is an input parameter to the Adam optimisation method which influences how fast the model learns from the input sequences, a too high value causes the model to learn fast initially but misses details in the information and never reaches high scores while a too low value causes the model to learn too slow.

Gradient clipping is a method to prevent the exploding gradients problem, when large error gradients accumulate and result in very large updates to the weight matrices while training the model. The value of the gradient clipping is a threshold which the weight updates cannot exceed. Another method that can be used to mitigate the exploding gradients problem is using a rectified linear unit (ReLU) as activation function, which is defined as $f(x) = max(0, x)$. Moreover, the rectified linear unit also solves the problem of vanishing gradients, when training slows down because the updates to the weight matrices are too small.

The batch size is the number of input sequences per learning epoch, a large batch size uses more memory and updates the weight matrices slowly while a small batch size causes large fluctuations in the gradient.

The dropout keep probability value is the probability of keeping units in each layer of a neural network, when the value is set to 1 the network is susceptible to overfitting, learning the data in the training set too closely and not being able to generalise to new unseen data.

Since the input word embeddings are fixed during training, the selected word embeddings can make an important difference to overall performance. Hence, different word embeddings with different dimensions were used as input to the ME-CRF model and then trained on the sim-M dataset with 20 dialogues. The following word embeddings were tested: GloVe (50 and 300 dimensions), word2vec (300 dimensions) and fastText (Bojanowski et al. 2016) (300 dimensions), all of which were trained on an export of the English Wikipedia. In addition, GloVe was also trained on the Gigaword dataset (Parker et al. 2011).

The results of hyperparameter tuning are in table 3, while the results of the different word embeddings are in table 4. The results are discussed in section 4.4.

4.3 Evaluation

The output tag sequences were evaluated using the CoNLL evaluation script [7], which measures the span level precision, recall and F1-score in general and for each tag individually. Even though this script was built for evaluating NER tasks, it works just as well for slot the filling task, since both rely on the IOB-tagging scheme. The measures can be calculated as follows:

[7] Retrieved from `https://www.clips.uantwerpen.be/conll2000/chunking/output.html`

$$\text{precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \times 100$$

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \times 100$$

$$\text{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Tokens that are tagged and predicted as outside (O) count as true negative and therefore they do not influence these measures. The F1-score is the harmonic mean between precision and recall, punishing large differences between these two values. Therefore, the F1-score is the measure that is used to compare the different models.

Parameter	Value
Learning rate	0.01
Gradient clipping	5
Batch size	32
Dropout	0.5
Non-linearity function	tanh

Table 3: Optimal hyperparameters found using grid search on the sim-M dataset of size 20, the values that achieved the highest score on the development set are shown.

Embedding	Dimensions	Vocabulary size	Precision	Recall	F1-score
word2vec	300	3000000	85.08	81.97	83.49
fastText	300	2519000	87.58	86.08	86.82
GloVe	300	400000	92.81	82.79	87.51
GloVe	50	400000	91.41	83.68	87.37

Table 4: Word embedding results. Trained on the sim-R dataset with 20 dialogues. Scores were measured on the development dataset.

4.4 Results

Results of the hyperparameter search and the different word embeddings are in table 3 and 4 respectively. The different word embeddings performed similar, with the exception of the word2vec embeddings. The GloVe embeddings achieved the highest F1-scores, even when limited to 50 dimensions, the smaller size also resulted in large performance improvements in terms of computational efficiency, making the load times of the model as much as 19 times as fast. Therefore, with the benefits of faster training times — and thus faster prototyping — and limited loss in F1-score, the GloVe embeddings of 50 dimensions were chosen. These word embeddings and the optimal hyperparameters from

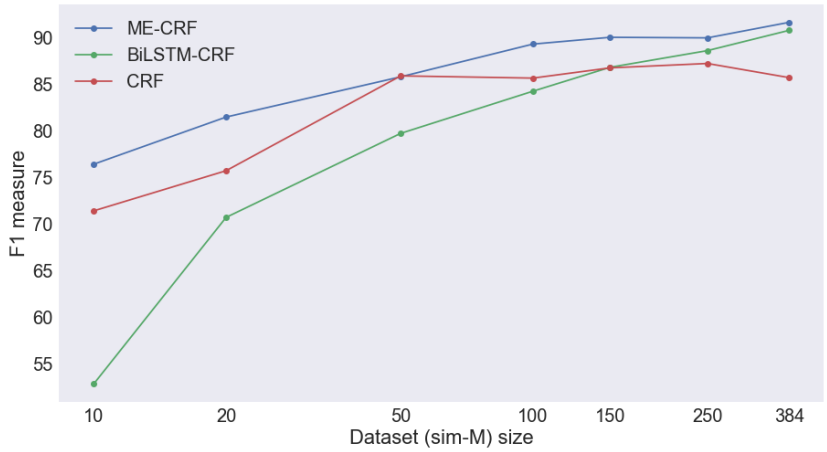

Figure 9: Results from CRF, BiLSTM-CRF and ME-CRF models at different dataset sizes on the sim-M dataset

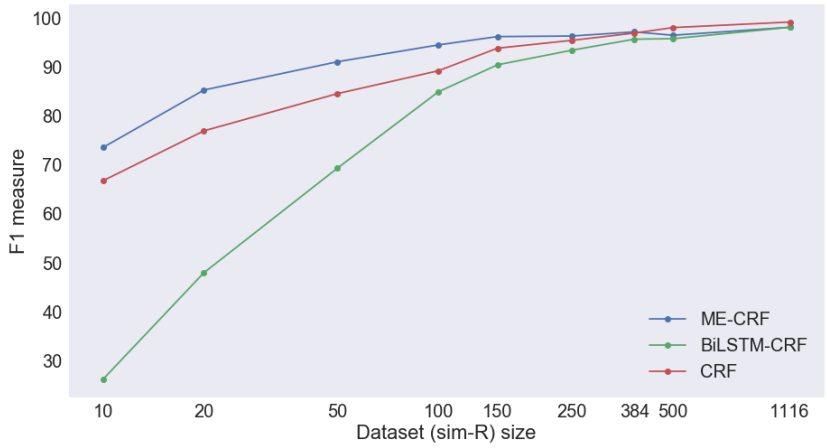

Figure 10: Results from CRF, BiLSTM-CRF and ME-CRF models at different dataset sizes on the sim-R dataset

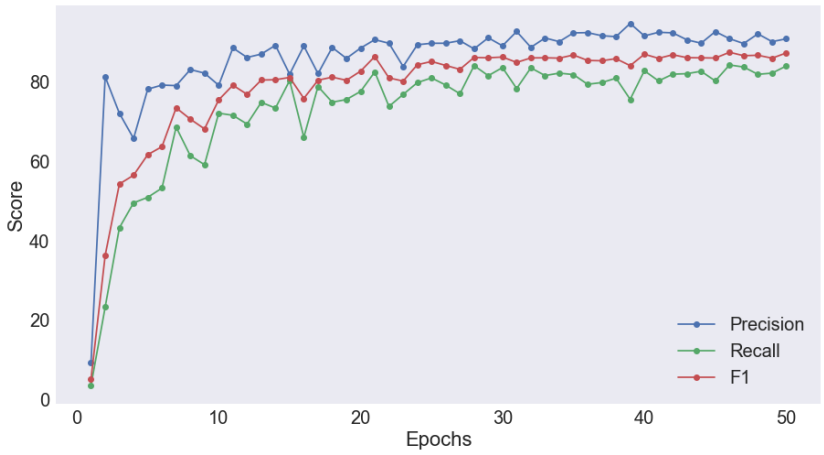

Figure 11: Precision, recall and F1-score evaluated on the test set over 50 epochs when being trained on the sim-R dataset with 20 dialogues.

table 3 were used to train the ME-CRF model on the different dataset sizes for both domains, the baseline models were evaluated on the same datasets.

The results of training the ME-CRF model on datasets of different sizes can be found in figure 9 and 10 for the movie (sim-M) and restaurant (sim-R) domains respectively. In the sim-M domain, which contains 5 different slots to be filled, the ME-CRF model outperforms both baseline models, especially at smaller dataset sizes. The CRF model performs similarly at dataset size 50, but stagnates with bigger dataset sizes. The BiLSTM-CRF model performs poorly at lower dataset sizes but achieves similar results to the ME-CRF model at sizes 250 and 384. In the sim-R domain, which contains 9 different slots to be filled, the ME-CRF model outperforms both baseline models at dataset sizes up until 150. The CRF model achieves the highest score with the full dataset of 1116 dialogues, but is outperformed by the ME-CRF model at smaller sizes. The BiLSTM-CRF model needs a lot more data to achieve any significant scores, achieving less than 30% F1-score when the training set consists of 10 dialogues.

Figure 11 shows the results of training the ME-CRF model on the sim-R domain with 20 dialogues over 50 epochs, where the model is evaluated on the test set at each epoch. The model achieves high precision from epoch 2, tagging only tokens when it is confident that they must be tagged with a certain slot name and therefore having a lower recall. Both precision and recall continue to improve until about epoch 30, when the improvements start to slow down. The best F1-score on the development set is achieved at epoch 34, therefore that version of the model is saved to disk and used even though the F1-score on the test set is higher at later epochs.

Table 5 shows the results for each individual slot label after 50 epochs of training the ME-CRF model on the sim-R domain with 20 dialogues. Occurrences is the number of times the label occurred as prediction, for example the label `restaurant_name` has 2056 occurrences and a recall score of 68.51, the total number of tokens that should have been tagged with `restaurant_name` would then be $\frac{2056}{68.51/100} = 3001$. The labels `meal` and `rating` only occur 3 and

Label	Precision	Recall	F1	Occurrences
category	87.54	93.79	90.55	345
date	91.26	95.31	93.24	824
location	78.64	75.19	76.88	501
meal	91.89	43.59	59.13	37
num_people	96.53	100.00	98.23	777
price_range	96.54	97.99	97.26	405
rating	24.00	13.79	17.52	50
restaurant_name	86.58	68.51	76.49	2056
time	98.74	99.75	99.24	1590
Total	90.89	84.05	87.33	6585

Table 5: Precision, recall and F1-score evaluated on the test after training on the sim-R dataset with 20 dialogues

2 times in the training set respectively, so the model has a hard time generalising from such a low number of occurrences. The label `restaurant_name` often consists of multiple tokens that are also often not in the vocabulary of the word embedding, making it harder to predict correctly. Most other labels are straightforward to learn, achieving high precision, recall and F1 scores. Even though both the labels `num_people` and `time` often consist solely of a number they can be differentiated using the regular expression feature with input `/[ap]m/`, matching both tokens `am` and `pm`.

5 Conclusion

In this thesis the slot filling problem for task-oriented conversational agents is investigated, focusing on solving the problem with little data. The contributions are twofold: first, the Memory-Enhanced Conditional Random Field (ME-CRF) model is used to store previous utterances of the dialogues in a memory component. Second, the dialogue acts from system utterances are used in an explicit feature extraction step, making it possible to model the dialogue. Combined, the resulting model outperforms two baseline models: Conditional Random Fields (CRF) and bidirectional LSTM network with a CRF layer (BiLSTM-CRF) on datasets consisting of 10 to 100 dialogues. The model effectively uses the benefits of both baseline models, requiring little data and using an explicit feature extraction step (CRF) and having access to a memory component and word embeddings (LSTM). These benefits allow the model to be used in practice for building task-oriented conversational agents, constructing a task-specific dataset of several dialogues with dialogue act annotations is the only prerequisite. Compared to the two baseline models this does introduce an extra step: labelling the system utterances in the dialogues with their dialogue acts.

5.1 Discussion and Future Work

The ME-CRF model used in this thesis shows promising results for the slot filling problem for task-oriented conversational agents. However, the model has only been applied to a dataset with two domains: booking a restaurant table and

buying a movie ticket. To assess whether the model can be adequately used in practice in different domains a new dataset must be obtained. Also, the dialogue acts that were used as features in system utterances provided great value in tagging the slots correctly, making it particularly interesting to experiment with dialogue acts for user utterances. However, the dialogue acts are not available when the conversational agent is deployed and receives unseen user utterances. This would require a new component: a dialogue act classifier. Furthermore, even though the model does not have any language specific components, it has not been tested on other languages, it will be interesting to see how it performs in other languages. There are word embeddings available for almost all languages[8], so adapting the model to a new language is a matter of constructing a new dataset. Last, the word embeddings that were used in the experiments were trained on Wikipedia, a resource that has more formal language compared to chat dialogues. Word embeddings that are learned on less formal datasets, such as data from chat dialogues, data from social platforms like Twitter, Facebook or Instagram, and even emoji data might be beneficial to the slot filling problem for conversational agents.

[8]fastText provides pretrained word embeddings in 294 languages: `https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md`

References

Asri, Layla El et al. (2017). "Frames: A corpus for adding memory to goal-oriented dialogue systems". In: *arXiv preprint arXiv:1704.00057*.

Austin, John Langshaw (1975). *How to do things with words*. Oxford university press.

Bapna, Ankur et al. (2017). "Sequential Dialogue Context Modeling for Spoken Language Understanding". In: *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pp. 103–114.

Bobrow, Daniel G et al. (1977). "GUS, a frame-driven dialog system". In: *Artificial intelligence* 8.2, pp. 155–173.

Bojanowski, Piotr et al. (2016). "Enriching word vectors with subword information". In: *arXiv preprint arXiv:1607.04606*.

Chen, Yun-Nung et al. (2016). "End-to-End Memory Networks with Knowledge Carryover for Multi-Turn Spoken Language Understanding." In: *INTERSPEECH*, pp. 3245–3249.

Cho, Kyunghyun et al. (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078*.

Collobert, Ronan et al. (2011). "Natural language processing (almost) from scratch". In: *Journal of Machine Learning Research* 12.Aug, pp. 2493–2537.

Elman, Jeffrey L (1990). "Finding structure in time". In: *Cognitive science* 14.2, pp. 179–211.

Harris, Zellig S (1954). "Distributional structure". In: *Word* 10.2-3, pp. 146–162.

Hemphill, Charles T, John J Godfrey and George R Doddington (1990). "The ATIS spoken language systems pilot corpus". In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

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

Huang, Zhiheng, Wei Xu and Kai Yu (2015). "Bidirectional LSTM-CRF models for sequence tagging". In: *arXiv preprint arXiv:1508.01991*.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kuhn, Roland and Renato De Mori (1995). "The application of semantic classification trees to natural language understanding". In: *IEEE transactions on pattern analysis and machine intelligence* 17.5, pp. 449–460.

Kumar, Ankit et al. (2016). "Ask me anything: Dynamic memory networks for natural language processing". In: *International Conference on Machine Learning*, pp. 1378–1387.

Lafferty, John, Andrew McCallum and Fernando CN Pereira (2001). "Conditional random fields: Probabilistic models for segmenting and labeling sequence data". In: *Proceedings of the 18th International Conference on Machine Learning*, pp. 282–289.

Liu, Fei, Timothy Baldwin and Trevor Cohn (2017). "Capturing Long-range Contextual Dependencies with Memory-enhanced Conditional Random Fields". In: *arXiv preprint arXiv:1709.03637*.

Lowe, Ryan et al. (2015). "The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems". In: *arXiv preprint arXiv:1506.08909*.

McCallum, Andrew, Dayne Freitag and Fernando CN Pereira (2000). "Maximum Entropy Markov Models for Information Extraction and Segmentation." In: *Icml*. Vol. 17. 2000, pp. 591–598.

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

Mikolov, Tomas et al. (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Parker, Robert et al. (2011). "English gigaword fifth edition, linguistic data consortium". In: *Google Scholar*.

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

Pieraccini, Roberto et al. (1992). "Automatic learning in spoken language understanding". In: *Second International Conference on Spoken Language Processing*.

Raymond, Christian and Giuseppe Riccardi (2007). "Generative and discriminative algorithms for spoken language understanding". In: *Eighth Annual Conference of the International Speech Communication Association*.

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

Serban, Iulian Vlad et al. (2016). "Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models." In: *AAAI*. Vol. 16, pp. 3776–3784.

Shah, Pararth et al. (2018). "Building a Conversational Agent Overnight with Dialogue Self-Play". In: *arXiv preprint arXiv:1801.04871*.

Sima'an, Khalil (2004). "Robust data oriented spoken language understanding". In: *New developments in parsing technology*. Springer, pp. 323–338.

Sukhbaatar, Sainbayar, Jason Weston, Rob Fergus et al. (2015). "End-to-end memory networks". In: *Advances in neural information processing systems*, pp. 2440–2448.

Ward, Wayne and Sunil Issar (1994). "Recent improvements in the CMU spoken language understanding system". In: *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, pp. 213–216.

Weston, Jason, Sumit Chopra and Antoine Bordes (2014). "Memory Networks". In: *arXiv preprint arXiv:1410.3916*.

Xiong, Caiming, Stephen Merity and Richard Socher (2016). "Dynamic memory networks for visual and textual question answering". In: *International Conference on Machine Learning*, pp. 2397–2406.