DEEP LEARNING A GYAKORLATBAN - HOMEWORK

Balazs Fodor
GU87AO
balazs.fodor@edu.bme.hu

Szilvia Hodvogner
W0VGDG
hodvogner.szilvia@edu.bme.hu

Gergely Dobreff
RZ3RVX
dobreff.gergely@tmit.bme.hu

1 Introduction

As a part of the Deep Learning Course, we chose the topic NLP2 (ChatBot based on deep learning) for our homework. Our goal is to build a Neural Network for the SMCalFlow challenge [1]. This competition was announced by Microsoft Semantic Machine, the motivation for this competition is that one of the central challenges in conversational AI is the design of a dialogue state representation that agents can use to reason about the information and actions available to them. They have developed a new representational framework for dialogue that enables efficient machine learning of complex conversations [2].

The main target of the challenge is to create algorithms that are able to maintain and update conversation states stored as a program (or as a graph). The output for each user message should be a so called lispress string which is a representation of the stored agent program. The predictions can be evaluated in codalab, using Microsoft's predefined scripts. Predictions need to be made for all the validation data, and submitted to the codalab platform, and the scripts will evaluate the accuracy.

There are several approaches for designing conversational agents, Gao et al. [3] differentiated three types of dialogue systems:

1. question answering agents: these allow users to query large scale knowledge bases (KB) or document collections via natural language
2. task-oriented dialogue agents: this type of dialogue systems consists of a natural language understanding (NLU) module, a state tracker for tracking and store the state of the conversation, a dialogue policy for selecting the next action based on the current state, and a natural language generator (NLG) for converting the action to a human readable response
3. social bots: nowadays the generation of conversational responses with the framework of neural machine translation (NMT) with encoder-decoder or seq2seq models has gained interest. Such end-to-end models have been particularly successful with social bot scenarios, these models cope well with free-form and open domain texts.

Considering the three types of dialogue systems above, we can state that our challenge belongs to the type of task-oriented dialogue agents, however, the generation of programs that keep track of the state of the conversation can be classified as a neural machine translation task too, because the program can be interpreted as a language for the computer, thus it is an interesting challenge to design and implement a solution that can translate from human to computer. As sequence to sequence models are widely used in NMT, we will explore these models in our work.

Ren et al. [4] proposed an LSTM-based encoder-decoder model with attention mechanism to generate search engine friendly queries from user messages. This approach is similar to our problem, because we also need to generate outputs, that can be processed by a program, but we need to take account the very strict output requirements of lispress format. Hu et al. [5] also used LSTM-based encoder-decoder model and were able to build a tone-aware chatbot, that could extract the tone of the input message, and generate a response considering it. In the original paper of the challenge [2] a baseline implementation was also introduced, similar to the previous ones, it used an encoder-decoder model with bidirectional LSTM in the encoder and LSTM in the decoder. These models are proven to be effective in sequence generation tasks, however another more complex model architecture, the so called Transformer gained popularity in the past few years [6], and research shows, that is can outperform the LSTM-based encoder-decoder models in several cases. Still, as we are new in the field of NMT, we chose to implement a simple encoder-decoder model for the challenge, and examine the possible extensions and several hyper-parameters of the model to better understand, how these parameters effect the overall accuracy of our model. The analysis of transformer-based models can be a continuation of our present work.

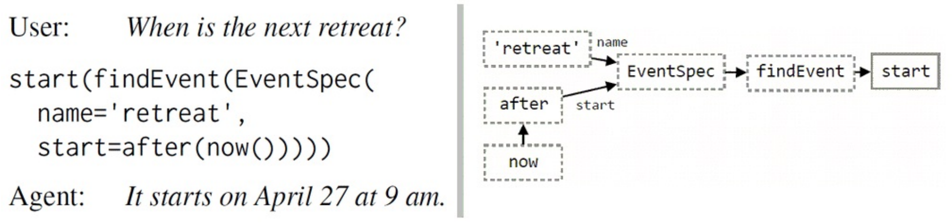
1.1 Dataset

The SMCalFlow challenge offers a large English-language dialogue dataset, featuring natural conversations about tasks involving calendars, weather, places, and people. It has 41,517 conversations annotated with dataflow programs. In contrast to existing dialogue datasets, this dialogue collection was not based on pre-specified scripts, and participants were not restricted in terms of what they could ask for and how they should accomplish their tasks. As a result, SMCalFlow is qualitatively different from existing dialogue datasets, featuring explicit discussion about agent capabilities, multi-turn error recovery, and complex goals.

The dataset models dialogues between a (human) user and an (automated) agent as an interactive programming task where the human and computer communicate using (English) natural language. A dialogue can consist of several turns, where a turn is the request-reply pair between the user and the agent. At each turn, the agent's goal is to translate the most recent user utterance (i.e. user request) into a program. The program is stored in a Lispress format. A Lispress program is an s-expression [7], which is intended to be human-readable. The program is used to generate human-readable response to the user's request.

The state of a given dialogue at a given turn is represented with a dataflow graph. At each turn, the predicted program for the given user request extend the dataflow graph, construct any newly requested values or real-world side-effects, and finally describe the results to the user.

The simplest example of interactive program synthesis is question answering. Fig. 1 shows the question asked by the user and the predicted program by the agent. In this case, the agent predicts a program that calls an API (findEvent) on a structured input (EventSpec) in order to produce the desired query. The predicted program on the left can be interpreted as a dataflow graph as it can be seen in the right figure.



User: *When is the next retreat?*

```
start(findEvent(EventSpec(
  name='retreat',
  start=after(now()))))
```

Agent: *It starts on April 27 at 9 am.*

Figure 1: User utterance, predicted program, and the agent's response (left). Corresponding dataflow graph (right). [2]

Since the user can refer back during each turn of the dialogues, it is important to maintain the graph, but it is also challenging for the agent to guess from the user's question which subject the user is referencing to. For example, if after the question in the previous figure, the user asks that: "What day of the week is that?" the agent must recognize that the user is referring to a particular response. The user can even refer to their own question. There is often a so-called variable in a user's question, such as a person's name or a date (e.g. "retreat" in Fig.1). These variables must be extracted from the text and exactly the same values must be included in the program.

2 System plan

In this chapter, we describe our goals and what network architectures we used, what variations of it, and the reasons for our decisions.

2.1 Goals

We targeted the part of the challenge where the agent predicts the program for the input user question. In part of the homework, we did not elaborate on generating a data flow graph from the predicted program and evaluate it and generate meaningful, readable output for the user. We chose this because the essential part of the challenge is to create a program that can be evaluated by a suitable processor. We assumed that such a processor was available. Thus, we aimed to train a suitable neural network for the user question received at the input, which predicts the output program as a natural language.

2.2 Network architecture

In order to achieve our goals we utilized a simple sequence to sequence model with LSTM encoder and decoder. Seq2seq models are used for language processing, e.g. language translation, text summarization and conversational model building. The architecture of this model is shown in Fig.2.

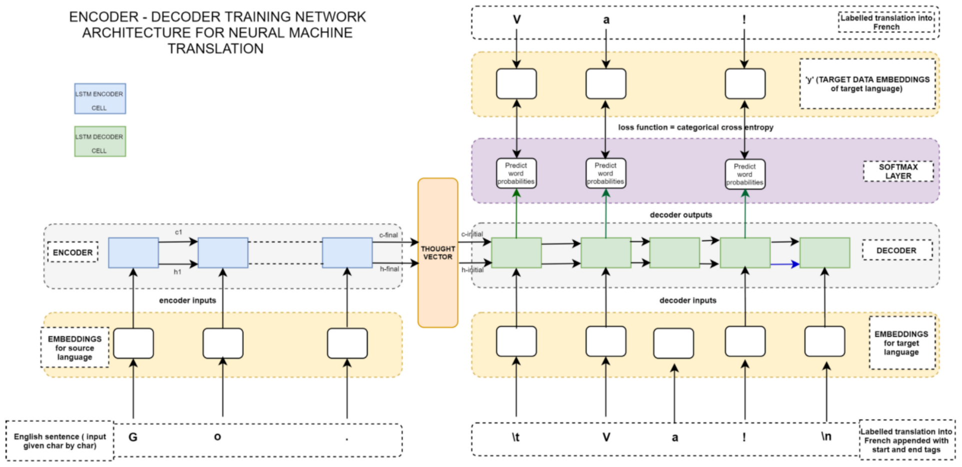

ENCODER - DECODER TRAINING NETWORK ARCHITECTURE FOR NEURAL MACHINE TRANSLATION
LSTM ENCODER CELL
LSTM DECODER CELL
ENCODER
c1
h1
c-final
h-final
THOUGHT VECTOR
c-initial
h-initial
DECODER
encoder inputs
decoder inputs
decoder outputs
EMBEDDINGS for source language
EMBEDDINGS for target language
English sentence (input given char by char)
G
o
.
\t
V
a
!
\n
Labelled translation into French appended with start and end tags
Predict word probabilities
SOFTMAX LAYER
loss function = categorical cross entropy
'y' (TARGET DATA EMBEDDINGS of target language)
Labelled translation into French

Figure 2: Architecture of Seq2Seq model. [8]

As it can be seen after the tokenized input is given to the Input layer of the decoder/encoder an Embedding layer takes each token (word) and converts them to a fixed sized vector. For embedding we used the GLoVe.6B.100d [9] which was trained on 6 billion tokens and has a 400K big vocabulary and each token is represented by a 100 long vector.

The Encoder is a stack of several LSTM units, where each unit accepts the corresponding element of the embedded input vector, then collects information for that element and propagates it forward to the next LSTM unit. The Thought Vector is the hidden state of the last LSTM unit of the Encoder. This vector aims to carry the extracted information of all the input elements in order to help the decoder make accurate predictions. This vector is the initial hidden state of the Decoder.

The Decoder - similar to the Encoder - is a stack of LSTM units where each unit accepts the hidden state from the previous unit and the corresponding element of the embedded target vector then predicts an output word. Then a Softmax Layer is used to create the probability vectors from the decoder outputs which will be used to determine the final output

The basic seq2seq model described above was tested with several variations. The encoder and decoder parts were replaced by a bidirectional LSTM network. Additionally, we examined how the use of Attention layer affects the performance. The theoretical background to these amendments is discussed below. The results of the modifications are presented at the end of the thesis.

2.3 Bidirectional LSTM

In the simple seq2seq model, both the encoder and the decoder use a unidirectional LSTM network. This means that, for example, the first word is processed, then the second, then the third, and so on, meaning that the encoder extracts the information from the input in order. In order to increase the amount of input information available to the network bidirectional LSTM network [10] is used. The output of the Encoder can get information from past (backwards) and future (forward) states simultaneously. This allows us to extract information about the context of the input.

The two-way LSTM network is formed by duplicating the first LSTM layer in the network so that there are now two layers side-by-side. The input sequence is provided to the first LSTM layer as-is, and a reversed copy of the input sequence is provided to the second layer. The first layer responsible for the positive time direction (forward states) and the second layer for the negative time direction (backward states).

2.4 Attention Layer

The Attention layer [11] [12], as its name suggests, tries to focus attention on more important parts of the input text while learning [6]. This is necessary because the simple encoder-decoder model has drawbacks. If the encoder makes a bad summary of the input, the prediction will also be bad.It has been observed that the encoder creates a bad summary when it tries to understand longer sentences, this phenomenon is called the long-range dependency problem of RNN/LSTMs. RNNs are not capable of remembering longer sequences due to the vanishing/exploding gradient problem. It can remember the parts which it has just seen.

The Attention layer tries to solve the above mentioned problem by using not only the hidden state of the last LSTM unit, but the hidden state of every unit to create the Thought Vector [13]. It is done by calculating the weighted sum of the hidden states. The weights are determined by using a feed-forward network. First, we implemented the first Attention mechanism by Bahdanau [12]. Using this Attention layer, our models learning became unstable, so we implemented a more improved and more complex layer called Hierarchical Attention Networks [11]. This model has two levels of attention mechanism, one at the word level and one at the sentence level. This attention type is more fitting for text which has a hierarchical structure and the importance of words and sentences are highly context-dependent.

3 Implementation

We used the popular tensorflow python library [14] to implement our solution, we also reused code from the original challenge to be able to tokenize the lispress programs. In the followings we will present the methods used for the data preparation, the training and testing process and we will analyse the results of the different training configurations, highlight the important parameters, architectural or data extensions worth to use for better accuracy.

3.1 Data preparation

In order to achieve our goal and train an agent that receives the user's question as input (user utterance) and predicts a program that generates an appropriate answer to this, which is in Lispress form, we need to prepare the given dataset to create train, test and validation datasets.

Both the question and the answer are tokenized. The Tokenizer class of tensorflow was used to tokenize the input questions. However, to tokenize Lispress, we used the tokenizer used by the competition announcers due to the syntactic characteristics of the language. After tokenization, <bos> and <eos> tokens were added to the beginning and end of the Lispress. Due to the limited resource we had, and after the deep examination of the target lispress programs, we decided to use only the programs up to 100 long in length, therefore we could reduce the training time and the resource usage, and could analyse the effect of different parameters. We made this decision, because analyzing the data, we found that only 3.3% of the lispress programs in the train data are longer than 100 characters.

Because the input layer of the seq2seq model expects a fixed-length input, all input and output data had to be augmented to be of equal length, this is padding. As mentioned earlier, we maximized the length of tokenized inputs in 100 to reduce resource demand. Thus, all inputs that were shorter than this were supplemented with padding tokens (i.e. empty strings) to be 100 long.

The peculiarity of the dataset is that the questions often contain variables such as: 2pm, Jhonny, and so on. These can be names, datetimes, eventnames or places. Naturally, the model should work accurately even if a name or date is mentioned by the user that it has never seen before. However, it is almost impossible to learn all the possible values of the variables. Therefore, an important point of the data preparation steps is to replace all the variables in the input text with a placeholder. An example of such a replacement is "Jhonny" $\rightarrow$ "<nameparameter0>". Since the user can mention more than one name for example, the variables are numbered. The same replacements were then made in the expected output, i.e. Lispress program. With this, we have achieved that the model can focus on the essential part of the questions - the structure of the input - during training, instead of the ever-changing variable.

3.1.1 Training set

Since we are utilizing the Seq2Seq model which contains an Encoder and a Decoder the training set for our Model is not obvious. For training the input for the Encoder, the input for the Decoder and the output for the Decoder must be given. Therefore our trainingset looks like the following, where the input sentence is the user's request and the current program is the lispress for the input sentence, and MAX_LEN_DEC denotes the length of the decoder input and output. The length of the encoder input is set to the maximum length of the user utterance sequences, and padding is added to the end of them. The same approach is used for the decoder inputs and outputs, MAX_LEN_DEC is set to the number of tokens in the longest agent program, and padding is added to the others.

3.2 Training process

A widely used method for training sequence generation models is teacher-forcing [15], which means that when training sequence generation models, we feed the previous ground-truth tokens as inputs of the decoder to predict the next target words. With this approach the error propagation can be eliminated from the training phase [16].

3.3 Evaluation

In total, we designed six different types of models. Our baseline model was a fundamental seq2seq model that contain only LSTM layers. The other models are all combinations of LSTM or Bidirectional LSTM layers, Attention or no Attention layers, and with or without the parameter replacement described above. Our evaluation plan was to train the models on the training set as defined in 3.1.1 and test them with 1000 samples. We measured all of the epochs' training and validation losses and also the accuracies. We calculated these accuracies according to how many words were right, however, in the SMCalFlow challenge, they computed the absolute accuracy, which means that every word of the result must be entirely right to get points. We also calculate our model's goodness for the test set like this, counting only those predictions when all the program tokens were predicted correctly.

We also planned to tune the hyperparameters of those models. We decided to first try out with six epochs long learns with 32 sized batches and varying only the model's hidden layer dimensions with 32, 64, and 128. After that, we chose the best models and fine-tuned the epoch and batch size.

Currently no regularization techniques are considered due to the limited resource and time, but it is essential that before submitting our solution, we investigate the impact of dropout, l1 and l2 regularization techniques to our model.

3.4 Testing

Our evaluation plan included 31 tests. In table 1, the parameters of each test and their results are shown. There are severe differences between the validation's accuracy and the test's accuracy. We already mentioned the reason behind this; the validation's accuracy is based on words; the test's accuracy is based on sentences. The test results show that using the Bidirectional LSTM layers, Attention layers, and our parameter replacement technique, we can achieve better accuracy. We also observed that increasing the dimensions of the hidden layers is beneficial. Our baseline seq2seq model's (line one of table 1.) final accuracy was 9.96%, but our best model has reached 54%, which is a significant improvement.

Our results show that such popular techniques like applying Attention layers and using Bidirectional LSTM layers in the encoder can significantly improve the accuracy of the model. We would like to highlight the impact of the parameter replacement: in the 29th and 30th rows of the table the same model configuration can be seen, the only difference is that the one in the 29th row uses parameter replacement, the other one does not. The difference is more than 10% in accuracy in favor of the one that uses parameter replacement. The disadvantage of parameter replacement, that we need a mechanism that is able to detect and replace parameters in a user message. Although we did not conduct an investigation in this regard, we assume that this can be done with another sequence to sequence model, whose only function is to replace messages with a new one with parameters. With this approach the challenge can be divided into two simpler tasks. The generation of such sequence to sequence model is planned in the future.

#	Encoder type	Hidden dimension	Attention	Batch size	Epochs	Parameter replacement	Validation accuracy	Test accuracy
1	LSTM	32	False	32	6	False	0.9276	0.0996
2	LSTM	32	False	32	6	True	0.9478	0.150
3	LSTM	32	True	32	6	True	0.9462	0.1562
4	LSTM	32	True	32	6	False	0.9323	0.1655
5	LSTM	64	False	32	6	False	0.943	0.1778
6	LSTM	64	False	32	6	True	0.9559	0.1993
7	LSTM	64	True	32	6	True	0.9601	0.2169
8	LSTM	64	False	32	6	False	0.9484	0.1819
9	LSTM	128	True	32	6	True	0.9593	0.2189
10	LSTM	128	False	32	6	True	0.9649	0.2826
11	LSTM	128	True	32	6	False	0.9597	0.2508
12	LSTM	128	False	32	6	False	0.9546	0.2261
13	BiLSTM	32	False	32	6	False	0.9464	0.1932
14	BiLSTM	32	False	32	6	True	0.9600	0.2446
15	BiLSTM	32	True	32	6	True	0.9590	0.2497
16	BiLSTM	32	True	32	6	False	0.9478	0.2035
17	BiLSTM	64	False	32	6	False	0.9582	0.2579
18	BiLSTM	64	False	32	6	True	0.9696	0.3751
19	BiLSTM	64	True	32	6	True	0.9695	0.3638
20	BiLSTM	64	False	32	6	False	0.9569	0.2332
21	BiLSTM	128	True	32	12	True	0.9761	0.4665
22	BiLSTM	128	True	32	6	True	0.9733	0.3967
23	BiLSTM	128	False	32	6	True	0.9736	0.43268
24	BiLSTM	128	True	32	6	False	0.9653	0.3062
25	BiLSTM	128	False	32	6	False	0.9656	0.3042
26	BiLSTM	192	True	32	12	True	0.9783	0.5344
27	BiLSTM	192	False	32	12	True	0.9781	0.5128
28	BiLSTM	192	True	64	12	True	0.9783	0.5405
29	BiLSTM	192	False	64	12	True	0.9784	0.5364
30	BiLSTM	192	True	64	12	False	0.9718	0.4142
31	BiLSTM	256	True	64	12	True	0.9789	0.5396

Table 1: The model's accuracy for different configurations

4 Conclusion and future work

We are on our way to accomplish a solution to the SMCalFlow challenge problem, and according to our expectation and measuring, the model should score around 50-55% accuracy. Currently, the best model contains a Bidirectional LSTM layered encoder and an LSTM decoder. It was trained with 192 hidden dimensions and also with an Attention layer. We prepared the input data with our specifically defined yet manual parameter replacement. For submitting this solution, we will need to make improvements in the parameter replacement mechanism. Our future work includes creating an automatic method for finding and replacing parameters in the input question and restoring them back to the parameters' values in the output code. After that, we will examine the final accuracy with and without the preparation. We also plan to do more hyperparameter tuning and examine the impact of increasing the number of Bidirectional and LSTM layers to find the most suitable submission model.

References

[1] Smcalflow. `https://microsoft.github.io/task_oriented_dialogue_as_dataflow_synthesis/`. (Accessed on 12/08/2020).

[2] Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, et al. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571, 2020.

[3] Jianfeng Gao, Michel Galley, and Lihong Li. Neural approaches to conversational ai. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1371–1374, 2018.

[4] Gary Ren, Xiaochuan Ni, Manish Malik, and Qifa Ke. Conversational query understanding using sequence to sequence modeling. In *Proceedings of the 2018 World Wide Web Conference*, pages 1715–1724, 2018.

[5] Tianran Hu, Anbang Xu, Zhe Liu, Quanzeng You, Yufan Guo, Vibha Sinha, Jiebo Luo, and Rama Akkiraju. Touch your heart: A tone-aware chatbot for customer care on social media. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008, 2017.

[7] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.

[8] Akira Takezawa. How to implement seq2seq lstm model in keras. `https://towardsdatascience.com/how-to-implement-seq2seq-lstm-model-in-keras-shortcutnlp-6f355f3e5639`. Accessed: 2020-12-08.

[9] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[10] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997.

[11] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. pages 1480–1489, 01 2016.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[13] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional attention flow for machine comprehension, 2018.

[14] Tensorflow. `https://www.tensorflow.org/`. (Accessed on 12/11/2020).

[15] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[16] Lijun Wu, Xu Tan, Di He, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. Beyond error propagation in neural machine translation: Characteristics of language also matter. *arXiv preprint arXiv:1809.00120*, 2018.