Dialogue Response Generation using Neural Networks with Attention and Background Knowledge

Sofiia Kosovan[1], Jens Lehmann[2] and Asja Fischer[3]

[1] RWTH Aachen University, Germany
sofiia.kosovan@rwth-aachen.de
[2] University of Bonn, Germany
jens.lehmann@cs.uni-bonn.de
[3] University of Bonn, Germany
fischera@iai.uni-bonn.de

Abstract

Constantly increasing amounts of data on the Internet as well as fast GPUs available allowed for the research in AI. Google DeepMind and self-driving cars are showing how fast machine learning flows into industry. It helps humans to do their job, and the chatbots are a good example of it. We use them to play music, tell us the weather, order a cab, and much more. Chatbots are all about generating the response given the conversation so far. In this work we train a dialogue response generation model using neural networks. It is a sequence to sequence model which takes a dialogue as an input and produces the next response as an output. The data used is the Ubuntu Dialogue Corpus - a large dataset for research in unstructured multi-turn dialogue systems. Since the input sequences are the dialogues, the inputs are pretty long, and in this case the information at the very beginning of the sequence is often lost while training the model. That is one of the reasons we are using attention. It allows the decoder more direct access to the input and lets the model itself decide which inputs consider more for output generation. Also, we extend the Ubuntu Dialogue Corpus with the information from the man pages in order to enrich the input with the technical information. We use the short descriptions. In this way we do not overload the input with too much additional information, but still add some background knowledge.

1 Introduction

An explosion in the number of people having informal, public conversations on social media websites such as Facebook and Twitter presented a unique opportunity to build collections of naturally occurring conversations that are orders of magnitude larger than those previously available. These corpora, in turn, present new opportunities to apply data-driven techniques to conversational tasks [13].

The task of the response generation is to generate any response that fits the provided stimulus without mentioning the context, intent or dialogue state. Without employing rules or templates, there is the hope of creating a system that is both flexible and extensible when operating in an open domain [13]. Success in open domain response generation could be useful to social media platforms and provide conversation-aware autocomplete for responses in progress or providing a list of suggested responses to a target status.

Researchers have recently observed critical problems applying end-to-end neural network architectures for dialogue response generation [15, 16, 8]. The neural networks have been

unable to generate meaningful responses taking dialogue context into account, which indicates that the models have failed to learn useful high-level abstractions of the dialogue.

When we want a chatbot that learns from existing conversations between humans and answers the complex queries, we need the intelligent models like retrieval-based or generative models. The retrieval–based models pick a response from a collection of responses based on the query. They do not generate new sentences and have always grammatically correct sentences. The generative models are much more intelligent. They generate a response word by word based on the query. They are computationally expensive to train, require huge amounts of data and often have grammatical errors. They learn the sentence structure by themselves. But the very important advantage of generative models over all the other types of the models is that they are able to remember the previous conversations and handle previously unseen queries.

In this work we train the Sequence to Sequence model on the Ubuntu Dialogue Corpus [6, 9, 10]. We train the generative model, which is generating the responses from scratch and does not pick the response from the predefined set like retrieval-based models do. To our knowledge, there was only one paper where the attention mechanism was used on the Ubuntu Dialogue Corpus [11]. Our work was done independently and in parallel to the paper. We also visualise attention weights to be able to see to which words the model attends to when generating the output. Modelling was stopped when perplexity on the evaluation set stopped decreasing by calculating moving average over the last steps. We also extend the input data with user manuals in order to enrich it with the technical information. Extending the input was already done for different tasks, like classification or response selection. We enlarge the dataset for the response generation task.

2 Related Work

The Ubuntu Dialogue Corpus is the largest publicly available multiturn dialogue corpus and is used for the task of response selection and generation [9]. There was considered a task of selecting best next response using TF-IDF, Recurrent Neural networks (RNN) and Long Short-Term Memory (LSTM). In the next utterance ranking task on the Ubuntu Dialogue Corpus were evaluated performances of LSTMs, Bi-LSTMs and CNNs on the dataset and created an ensemble by averaging predictions of multiple models [6]. The best classifier was an ensemble of 11 LSTMs, 7 Bi-LSTMs and 10 CNNs trained with different meta-parameters.

The more interesting task that the response selection is response generation. Generative models produce system responses that are autonomously generated word-by-word. They open up the possibility for flexible and realistic interactions. Generative models were used for building open domain, conversational dialogue systems based on large dialogue corpora [16]. There was also introduced the multiresolution recurrent neural network (MrRNN) for generatively modeling sequential data at multiple levels of abstraction [15]. MrRNN was applied to dialog response generation on two different tasks: Ubuntu technical support and Twitter conversations, and evaluated it in a human evaluation study and via automatic evaluation metrics.

To our knowledge, there was only one paper published recently with attention model on the Ubuntu Dialogue Corpus ("Coherent Dialogue with Attention-based Language Models") [11]. There was modeled coherent conversation continuation via RNN-based dialogue models. They investigated how to improve the performance of a recurrent neural network dialogue model via an attention mechanism. They evaluated the model on two dialogue datasets, the open domain MovieTriples dataset and the closed domain Ubuntu Troubleshoot dataset. There was also showed that a vanilla RNN with dynamic attention outperforms more complex memory models (e.g., LSTM and GRU) by allowing for flexible, long-distance memory. The paper "Coherent

Dialogue with Attention-based Language Models" was written independently and in parallel to our work.

3 Sequence to Sequence Models

Here we discuss the difference between retrieval–based and generative models, explain RNNs and LSTMs, describe encoder–decoder sequence to sequence architectures. After that we show an interesting example of the attention mechanism, and why it is popular.

3.1 Retrieval-Based vs. Generative Models

Retrieval-based models pick a response from a fixed set and do not generate any new text. A response is picked based on the input and context with the help of some heuristic (rule-based expression match or an ensemble of Machine Learning classifiers). The responses are predefined. Retrieval-based models are of course always grammatically correct. But picking the sentence out of predefined set rarely makes much sense in Dialogue Systems. There are so many different possibilities of conversations, that we really need to generate a new response instead of choosing it.

Generative models is a harder task. There are no predefined responses. The new responses are generated from scratch. These models are based on Machine Translation techniques, but are also widely used for dialogue response generation. Generative models give the feeling of talking to a human and unlike the retrieval-based models can refer back to the entities in the input. Generative models require huge amounts of training data and are very hard to train. Also they are likely to make grammatical mistakes, especially if the sentence is long. The research is moving into the generative direction and in this work a generative model is built.

3.2 RNNs and LSTMs

In some machine learning algorithms (as well as neural networks) it is assumed that the inputs and outputs are independent of each other. Obviously these models should not be used for the text generation, where each word in a sentence heavily depends on the previous words. This is where recurrent neural networks (RNN) come into play. They have a "memory" which captures information about what has been calculated so far. RNN performs the same task for every element of a sequence and that is why is called "recurrent".

RNN is neural sequence model that achieves state of the art performance on important tasks that include language modeling, speech recognition, machine translation and image captioning [12, 4, 7]. They are widely used for natural language processing. In most cases LSTMs (Long short-term memory), type of RNNs, are used since they are better at capturing long-term dependencies [5]. The main difference between RNNs and LSTMs is that they have a different way of computing the hidden state.

The LSTM contains special units called memory blocks in the recurrent hidden layer [14]. The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block in the original architecture contained an input gate and an output gate. The input gate controls the flow of input activations into the memory cell. The output gate controls the output flow of cell activations into the rest of the network. Later, the forget gate was added to the memory block [2].

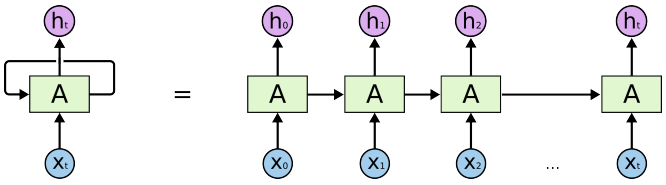

Figure 1: An unrolled recurrent neural network

On the Figure 1[1] is a simple representation of the RNN with inputs x_i and outputs h_i. RNNs have loops in them, allowing information to persist. It can be thought of as multiple copies of the same network, each passing a message to a successor. Recurrent neural networks are intimately related to sequences and lists because of it's chain-like nature.

3.3 Encoder-Decoder Sequence to Sequence Architectures

In many applications, such as speech recognition, machine translation or question answering the input and output sequences in the training set are generally not of the same length. RNN can map an input sequence to an output sequence which is not necessarily of the same length.

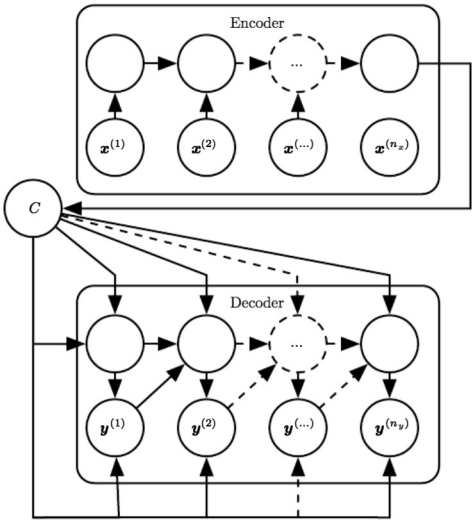

Figure 2: Example of an encoder-decoder or sequence to sequence RNN architecture

On the Figure 2 is an example of an encoder-decoder or sequence to sequence RNN architecture [3]. It is used for learning to generate an output sequence ($y^{(1)}$,..., $y^{(n_y)}$) given an input sequence ($x^{(1)}$,..., $x^{(n_x)}$). It is composed of an encoder RNN that reads the input sequence and a decoder RNN that generates the output sequence (or computes the probability of a given output sequence). The final hidden state of the encoder RNN is used to compute a generally

[1] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

fixed-size context variable C which represents a semantic summary of the input sequence and is given as input to the decoder RNN.

The input to the RNN is often called the "context". C is a representation of this context. It might be a vector or sequence of vectors that summarize the input sequence $X = (x^{(1)}, ..., x^{(n_x)})$.

The simplest RNN architecture for mapping a variable-length sequence to another variable-length sequence was proposed recently and the state-of-the-art translation using this approach was obtained [1, 17]. The idea is:

- an encoder or reader or input RNN processes the input sequence. The encoder emits the context C, usually as a simple function of its final hidden state.
- a decoder or writer or output RNN is conditioned on that fixed-length vector to generate the output sequence $Y = (y^{(1)}, ..., y^{(n_y)})$.

In a sequence to sequence architecture, the two RNNs are trained jointly to maximize the average of $logP(y^{(1)}, ..., y^{(n_y)}|x^{(1)}, ..., x^{(n_x)})$ over all the pairs of x and y sequences in the training set. The last state h_{n_x} of the encoder RNN is typically used as a representation C of the input sequence that is provided as input to the decoder RNN.

3.4 Attention Mechanism

An attention mechanism was introduced by Bahdanau, Cho, and Bengio [1]. It is used in the model to allow the decoder more direct access to the input. The decoder peeks into the different parts of the source sentence at each step of the output generation. The full source sentence is no longer encoded into a fixed-length vector. Now the model learns what to attend to based on what it has produced so far as well as on the input sentence.

On the Figure 3 below s_t is an RNN hidden state for time t, *y's* are the translated words produced by the decoder, and the *x's* are the source sentence words [1]. The *a's* are weights that define in how much of each input state should be considered for each output. For example, $a_{1,3}$ shows how much attention the decoder pays to the third state in the source sentence while producing the first word of the target sentence. Each decoder output word y_t now depends not only on the last state, but also on a weighted combination of all the input states.

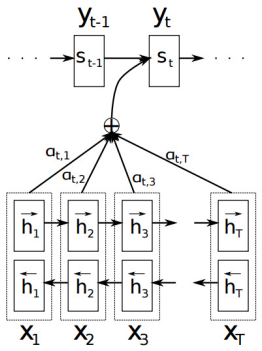

Figure 3: The graphical illustration of the model generating the t-th target word y_t given a source sentence $(x_1, x_2, ..., x_T)$

Illustration of the attention mechanism is on the Firgure 4 [1]. The source sentence is in English and the generated translation is in French. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word in grayscale (0: black, 1: white). "La destruction" was translated from "Destruction" and "la Syrie" from "Syria", so the corresponding weights are non-black.

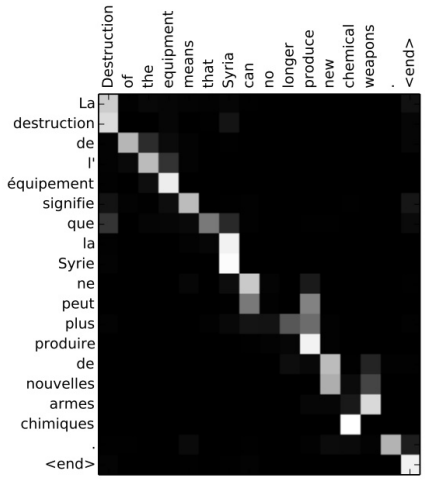

Figure 4: Attention mechanism example for word sequences

4 Experimental Results

Due to very long input sequences and a very large dataset, the maximum batch size we could fit into TITAN X with 12 GB for the model with 2 layers, 1024 units each was 30. This is very small compare to 0,5 million data instances, which made the training process of such a large model very slow.

Responses generated by the model were always grammatically correct and different, suitable as response or not, but very generic. That is why we did not evaluate them using different metrics such as BLEU or METEOR, but used perplexity, visualized attention and looked at the generated responses.

When training the model statistics was printed and model was saved as checkpoint every 50 steps. During a step the model was trained using one batch. We also calculate moving average over last 3000 steps since the perplexities every 50 steps were very different and hard to follow.

We stopped the training process when the perplexity on the evaluation set stopped decreasing. On the Figure 5 is an example of how the perplexity on the evaluation set decreases over time. On the X axis are the number of steps, and on the Y axis is the perplexity. This model was trained on TITAN X with 12 GB for 10 days. The training takes that long due to large input sequences, big number of dialogues and using attention, which further makes the model bigger and more difficult to train. This model had 2 layers with 1024 units each.

An interesting observation on Figure 5 is that the perplexity of the smallest bucket is very close to the perplexity on the training set. The perplexities for bigger buckets are larger: the smallest perplexity is reached for the smallest bucket, and the largest perplexity is reached for

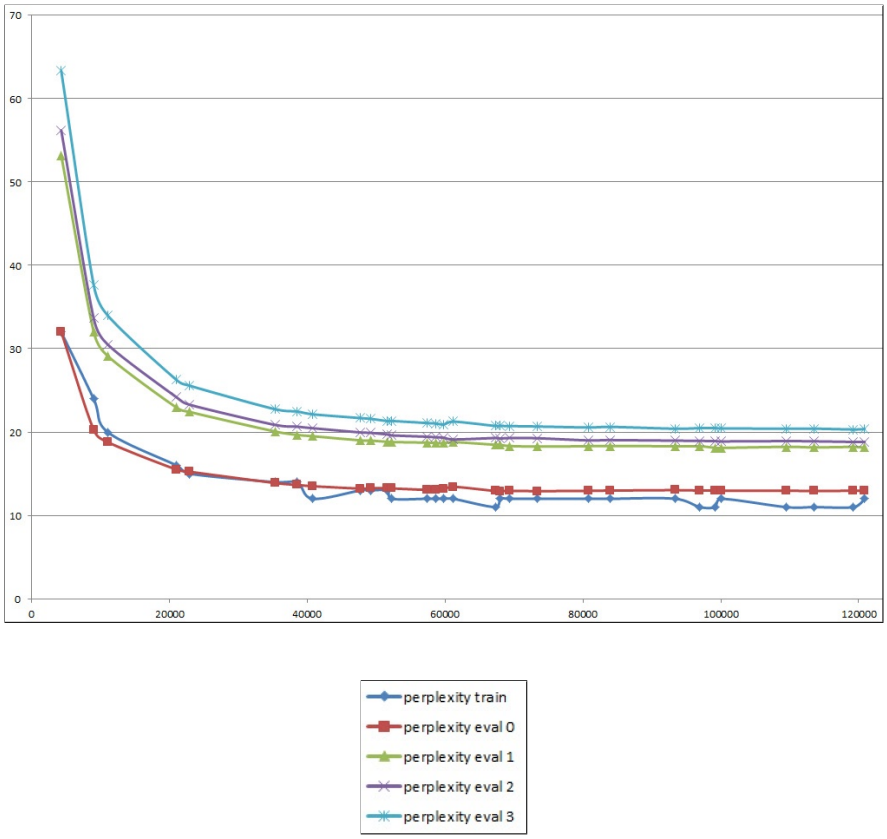

Figure 5: Decreasing of perplexity while training the model

the biggest bucket (bigger bucket - larger perplexity). This shows that it is more difficult for the model to memorize longer sequences then shorter sequences, as expected.

The responses generated by the models were very generic. The reasons for that may be that the beam search was not used and that the "I don't know" and "I am not sure" are indeed the most common answers on the Ubuntu Dialogue Corpus. We show some examples of the response generated (***** indicates the response generated by the model and + is the real answer by the user).

Example

B: sorry i have no idea what that is
B: You can disable luks password prompt at boot by adding "rd_NO_LUKS" kernel flag to grub.conf
A: yah!! where, grub.cfg? syntax please. thanks
A: whats the syntax for rd_NO_LUKS? where to put in grub file
B*: I don ' t know , sorry
B+: it doesn't say
B+: can you reformat the disk?

When we take a closer look on the other words the model generated as candidates, we notice much more variety among them (for now we cannot see the other possible responses, as we did not use beam search and this is an idea for the future work). So on the Figure 6 we show the top 10 generated tokens - candidates for each word in the output response:

```
['"', 'grub', 'http', '_UNK', 'I', 'the', 'https', '/boot/grub/menu', 'it', 'read']
['I', 'grub', '_UNK', 'it', 'sorry', 'the', 'no', '"', 'read', 'It']
['don', "'", 'dont', 'am', 'have', 'think', 'can', 'would', 'dunno', 'just']
["'", ';', '.', ',', 't', '__eou__', 'understand', 'really', 'sure', '_UNK']
['t', 'T', 'm', '_UNK', 'see', 'think', 'd', 've', 'know', 'remember']
['know', 'think', 'understand', 'use', 'remember', 'see', 'have', 'believe', ',', 'recall']
[',', '.', 'what', 'about', 'how', 'if', 'where', 'the', 'grub', 'that']
['sorry', 'I', 'but', 'it', 'i', 'you', 'just', 'use', 'please', 'grub']
```

Figure 6: Top 10 most probable tokens for each word in the generated response

First items from each row on Figure 6 form the output "I don't know, sorry" for the dialogue above, since it is one of the most safe answers.

An interesting way to see how the attention is used by the model is to visualise it. On the Figure 7 are the attention weights visualised (darker square - bigger weight). The X axis represents the input dialogue and the Y axis shows the output response.

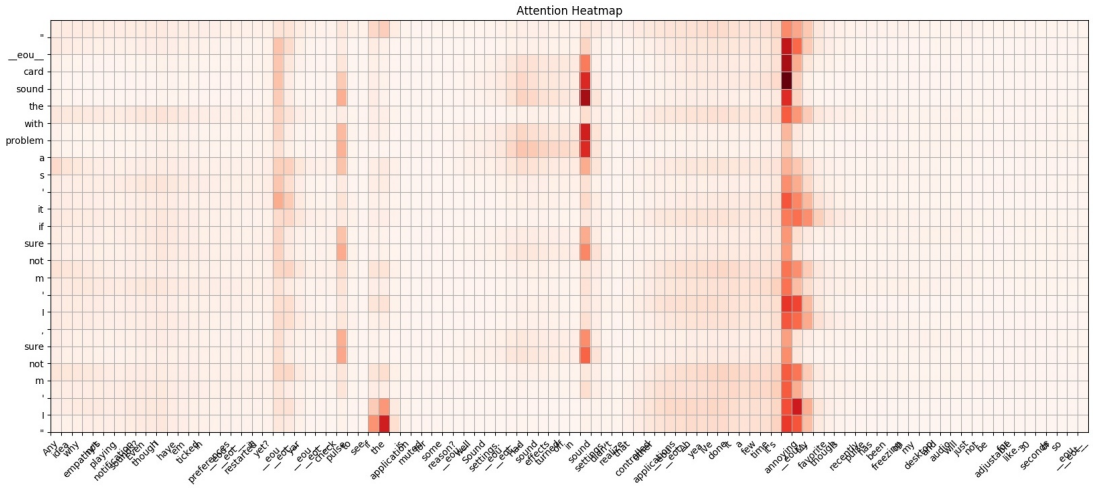

Figure 7: Visualisation of the attention weights

On the Figure 7 we can see, that the model was paying more attention to the words "pulse" and "sound" when generating the output response for the dialogue below. The generated response is **A***.

Example

A: Any idea why empathy's not playing notification sounds? Even though I have 'em ticked in preferences
B: restarted it yet?
A: yar

B: check pulse to see if the application is muted for some reason?
B: well Sound settings.
A: Had sound effects turned off in sound settings, didn't realize that controlled other applications
B: Ah yea, ive done it a few time it's annoying
B: My favorite though is recently pulse has been freezing on my desktop and audio will just not be adjustable for like... 30 seconds or so
A*: I'm not sure, I'm not sure if it's a problem with the sound card.

In conclusion we can say, that the model tends to give grammatically correct and different, but generic responses. We believe that adding beam search will give the possibility to see the other responses as well and choose the one among them.

Consider the next example. The user **B** at the end was giving the instructions what to do. The model generated response "thanks, I'll try that", which is suitable in this case. In any case, there is enormous number of different possible responses, and every human would answer differently.

Example
A: how do I create a folder from the command line?
A: ooooookay, how do I restart a process? kill and start? or is it easier? :P
B: depends on the process... who is it owned by? is it a system service?
A: nautilus
A: it seems to randomly lock up, so I was going to assign a keyboard shortcut to restart it, only to find out I don't know how to restart a process, or if it's even possible...
B: Use the "kill" command to send processes signals telling them to exit. You need the process id to use "kill" e.g. kill -TERM 1234. You can also use "killall some-process-name" e.g. killall nautilus
A*: thanks, I'll try that
A+: so there's no restart?

5 Challenges with Application in Industry

Machine learning and text mining methods are now widely used in industry. However, there are still many open challenges when implementing the chatbots. The deep learning based dialogue response generation models make it very hard to control the flow of the conversation. It is difficult to know when a discussion was successful and a case can be "closed". It is as well problematic to incorporate company knowledge into the process. In those "non goal driven" dialogue generation methods, it's of course also hard to include process knowledge (i.e. the chatbot should always ask for some insurance number in certain cases) - it may learn this from previous conversations or not.

Another task would be to keep track of the "state" of the conversation. For instance, a customer wants to know whether he has warranty for a broken notebook screen. In this case, the system needs to know when the notebook was purchased and whether it was damaged in an accident (or is a manufacturing problem). Once the system has this information, it can actually answer the original request of the customer.

6 Conclusion and Future Work

Dialogue response generation is still a challenge. Every human has his opinion and would answer differently to the questions in the same situations. Even the same person would answer differently to the same question depending on his mood and when you ask.

Dialogue response generation is more difficult then the question answering since the input to the model is not a question, but a dialogue, and we consider the dialogues up to 160 tokens long. Nevertheless we try to build the deep learning model to generate the responses the human would give.

As evaluation metric we used perplexity and stopped the training when the perplexity on the evaluation set stopped decreasing. Due to the large input size, long sequences and using attention mechanism, the model used was large (e.g. 2 layers, each with 1024 units). Training time of such a model was around 10 days on TITAN X with 12 GB. The responses produced by the models were always grammatically correct, different, but generic. We believe that further improvements to the model, like using beam search will help to overcome this issue.

The reason why our models produce such generic responses could be that the **beam search** was not used while decoding. It is currently not implemented in TensorFlow models, so there is still work to be done. As our research showed, the model without beam search produces very generic responses in the dialogue response generation task. When displaying the best suitable words for the response proposed by the model, we observe, that the model generated many more possible responses to choose from. With the beam search we will be able to see them and then pick the best one. For now the model picks the highest probable response, which is natural and logical since the answer "I am not sure, sorry" indeed can be given as response to any possible question.

In this work we used the manpages as additional knowledge. Another way would be to include the **different type of knowledge** from Ask Ubuntu[2] (a question and answer site for Ubuntu users and developers). In this case we would search through the website, find the most relevant question for each dialogue and then condition on the answer to the question (or both the question and the answer).

Another interesting idea for the future work would be to compare the performance difference between a model using attention and another **model not using attention**. Here the availability of a GPU with large RAM should be considered, as well as possible large amount of training time.

When analysing results we looked at perplexity, visualised attention weights and analysed the model responses. But due to very generic responses generated by the model we did not **evaluate** the results using all the evaluation metrics described before. During this work many different evaluation metrics where analysed and compared. So, when better results are available they can be analysed using another measures.

We implemented additional knowledge into the model appending it to the input sequence. Another way would be to add the manpage information in a **separate encoder** RNN, and then perform attention on this jointly. This would be better because the model can easily separate out dialogue history from the technical manpages. For example, the model could learn a separate attention mechanism for the dialogue history and for the man pages. It would also shorten the length of the encoding sequences, which helps to reduce long-term dependencies.

[2] `www.askubuntu.com`

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[2] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[4] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[6] Rudolf Kadlec, Martin Schmid, and Jan Kleindienst. Improved deep learning baselines for ubuntu corpus dialogs. *CoRR*, abs/1510.03753, 2015.

[7] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, volume 3, page 413, 2013.

[8] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *CoRR*, abs/1510.03055, 2015.

[9] Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *CoRR*, abs/1506.08909, 2015.

[10] Ryan Lowe, Nissan Pow, Iulian V. Serban, Laurent Charlin, Chia-Wei Liu, and Joelle Pineau. Training end-to-end dialogue systems with the ubuntu dialogue corpus. *Dialogue Discourse*, 8(1):31–65, 2017.

[11] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. Coherent dialogue with attention-based language models. *CoRR*, abs/1611.06997, 2016.

[12] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. *SLT*, 12:234–239, 2012.

[13] Alan Ritter, Colin Cherry, and William B. Dolan. Data-driven response generation in social media. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 583–593, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[14] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014.

[15] Iulian Vlad Serban, Tim Klinger, Gerald Tesauro, Kartik Talamadupula, Bowen Zhou, Yoshua Bengio, and Aaron C. Courville. Multiresolution recurrent neural networks: An application to dialogue response generation. *CoRR*, abs/1606.00776, 2016.

[16] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *CoRR*, abs/1507.04808, 2015.

[17] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.