Technische Hochschule Rosenheim

Improving AI Chatbots with Reinforcement Learning

John Lyons
Applied Artificial Intelligence
Department of Computer Science

WS 21/22

Abstract

The paper focuses in comparing two different approaches of improving chatbots using Reinforcement Learning (RL). Specifically, RL is used in order to improve performance of the underlying language model. For example, Ricciardelli and Biswas (2019) show that a pre-trained *score model*, which is able to predict binary scores for unseen user utterances, can automatically label data saved during development. Thus, only a small subset of data has to be initially labeled. Furthermore, the newly labeled data can be used to fine-tune a Deep Q-Network (DQN) which ultimately leads to an increase in chatbot performance by 25%. On the other hand, Li et al. (2016) intends to increase long-term conversation quality by forming special reward functions. These functions encourage coherent, informative and easy to answer utterances and are used in order to train an improved version of the initial language model and a policy. Unlike the approach of Ricciardelli and Biswas (2019), in which utterances are gathered from log files, Li et al. (2016) show how to explore state-action space using two separate chatbot agents talking to each other. Building upon the results of Li et al. (2016) the author shows that current state-of-the-art language models, such as *GPT-3*, can also construct long-lived conversations. Summing it up it can be said that using RL helps to improve performance of underlying language models. Tay.ai, a Microsoft chatbot experiment, could have made extensive profit of using RL in order to control output of abuse or racist language.

Contents

Acronyms

NLP Natural Language Processing

NLU Natural Language Understanding

RL Reinforcement Learning

ML Machine Learning

RNN Recurrent Neural Network

AUC Area under the ROC Curve

MDP Markov Decision Process

SEQ2SEQ Sequence to Sequence

MLE Maximum Likelihood Estimation

DQN Deep Q-Network

FAQ Frequently Asked Questions

LSTM Long Short-Term Memory

1 Overview

1.1 Introduction

Because Machine Learning (ML), specifically in the area of Natural Language Processing (NLP), is evolving very quickly, these days chatbots are common in various scenarios. For example, on websites chatbots help a user to perform specific actions or receive help without the need of a human support client.
Because a chatbot can typically handle multiple users at once and is available throughout the whole day cost can be reduced by a large margin compared to hiring human. The user also experiences superior response times and easiness of use.
The NLP engine of a chatbot can consist of the following components

- Intent classification
- Question Answering
- Entity extraction

The intent classification component tries to detect the initial intent of the user. In the simplest case this can be used to generate Frequently Asked Questions (FAQ)-style answers for detected intents.
In case the user wants to receive an answer, the question answering component, which nowadays most likely is based on a *BERT-like* architecture, helps to extract a question from a given source. Often there is also an entity recognition component which further helps to recognize the user's intent and needed action.
Other than that, there is still the field where chatbots are being used for entertainment purposes such as conversations between human and a chatbot. Tay.ai, a twitter chatbot developed by the Microsoft team, was such an experiment in the year 2019 which unfortunately did not end very well. Shortly after the initial release people began to abuse the bot's ability to learn abusive and racist content. Therefore, Microsoft felt the need to dispose the chatbot from twitter. In the end of this paper, we will discuss possible reinforcement learning approaches which could have helped in that specific case.
This paper highlights two very different approaches on how to improve chatbots by using reinforcement learning. The first part focuses on improving the intent classifier, which is then used in a classical chatbot architecture. The second part discusses improvement for conversational style chatbots, where the focus lies in providing the user with a comprehensive, informative and easy to understand dialog utterance as a response. (Li et al., 2016)
The next chapter will introduce two state-of-the-art language models, which became popular in recent years, along with some of the problems they bring with.

1.2 Intent classifiers and text-generation LMs

Intent classification plays an important role in the chatbot architecture because it helps to choose the next action of the chatbot based on the users first utterance. For example, a simple FAQ-style chatbot can have multiple predefined answers which can be used after analyzing the users question and generalizing it into an intent.
A recent advancement in machine learning and natural language processing is *BERT*, which stands for "Bidirectional Encoder Representations from Transformers". With sufficient pre-training and fine-tuning this language model pushes many scores higher than any language

model did before. For example, the *GLUE benchmark* is nowadays often used in order measure the quality of a language model. It contains multiple tasks, such as measuring the similarity between two sentences or classifying a sentence as grammatically correct. *BERT* reaches a *GLUE* score of 80.5% and thus beats the previous best result by 7.7%. Nowadays, *BERT* is often used in intent classification and question answering tasks. (Devlin et al., 2019)
As far as text generation language Models are concerned there are also many newcomers such as *GPT-2*, *GPT-3* and "Aleph Alpha's" language model. Those models increasingly profit from the huge amount of parameters. Unlike *BERT*, *GPT-3* does not need to be fine-tuned to achieve strong performance on many NLP tasks. Instead, a few-shot learning strategy is applied, in which specially crafted prompts are prepended to the input text. Given the prompt and the input text the *GPT-3* model generates output in a similar fashion and thus can be used for translation, question-answering and many other tasks. (Brown et al., 2020)
However, there are some drawbacks by using those models. First the increasing size in parameters limits the use of the language model in a production environment because huge models often need a lot of RAM, a graphics card with hardware acceleration (Nvidia + CUDA or AMD + ROCm) and enough VRAM. In addition to that, huge models need a lot of data for the pre-training steps which essentially results in long training time when hardware resources are limited.
But there still exist older language models, such as classic Sequence to Sequence (SEQ2SEQ) Recurrent Neural Network (RNN)s. Even though the inference and training time can take longer due to the nature of RNNs the reduced memory model size can compensate for that by allowing deployment on low resource hardware. Other than that, because those models exist longer, a lot of scientific work has been put into improving them.
Therefore, this paper is going to discuss an interesting approach on how to integrate reinforcement learning into a classical RNN based SEQ2SEQ model in order to achieve good quality conversations between two dialogue partners.
The next chapter will show a solution which allows improvement of FAQ-style chatbot performance by using a score model and a DQN.

2 Improving FAQ-style chatbots with RL

2.1 Basic idea

Nowadays fine-tuning existing chatbots or language models on specific domain or downstream tasks can require a lot of data. To overcome this problem and increase performance in existing solutions Ricciardelli and Biswas (2019) propose an approach which uses reinforcement learning to solve this issue.
The main idea is to make use of a score model which allows the generation of rewards given specific actions and states. In this particular case the states are user utterances and the actions are chatbot responses. The backbone of the architecture is an intent recognition Natural Language Understanding (NLU)-unit which holds predefined answers for user intents. It is interchangeable and as a result can be replaced by any industry standard NLU-unit, such as *IBM Watson*. (Ricciardelli and Biswas, 2019)

Fig. 1 depicts the complete architecture. At the top there is the initial intent detection NLU-unit which is responsible for classifying the user's utterance. In this particular example it always returns the response along with the corresponding confidence level. In the case of a low confidence level a fallback answer is returned. (Ricciardelli and Biswas, 2019)
The *DQN Agent* is trained in order to predict the best response action given a user utterance.

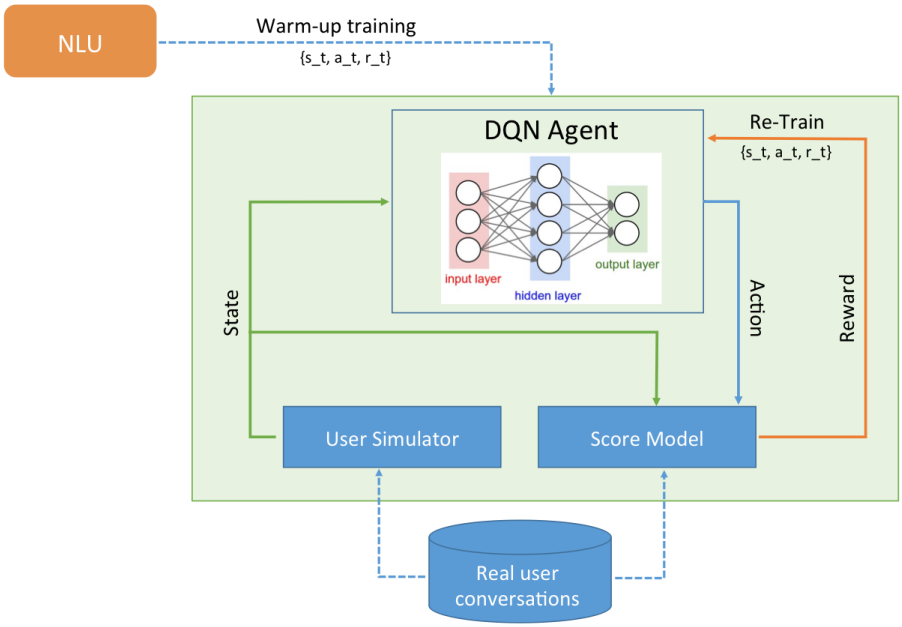


Figure 1: FAQ-RL architecture overview. Source: (Ricciardelli and Biswas, 2019)

In the beginning the model is trained in a warm-up phase where the confidence level of the underlying NLU-unit is used as a reward. That results in the policy performing roughly as good as the NLU-unit (in the beginning). (Ricciardelli and Biswas, 2019)
In order to further train the agent in a supervised manner the *score model* and the *user simulator* come into play. The *user simulator* picks utterances from a database, which holds many entries gathered from user logs, and sends them batch-wise to the agent. Following that, the agent learns the optimal policy by being trained with the utterances and rewards (the rewards being generated by the *score model*). Because the *score model* rates the generated responses and the agent is trained using those scores, the performance of the resulting *policy model* is highly influenced by the *score model*.
Because the amount of log entries is extremely large it is not affordable to label all the data by hiring human. So, in order to automatically label unseen utterances in a way that increases quality of the baseline model a special *score model* has to be trained.

2.2 Modeling scores

The *score model* is trained on a small subset of labelled data. These data were acquired during development of an actual chatbot. The historical data contains the user utterance, the generated response, the NLU confidence level and the user feedback. (Ricciardelli and Biswas, 2019)

Table 1 shows an example of the data. The user feedback is in a binary format so that every feedback is either zero or one.
Because the utterances cannot be directly fed into the score model, they have to be preprocessed by a *universal sentence encoder* available on Tensorhub (which is described by Cer et al. (2018)). The transformation results in a vectorized representation of the sentence which can be further used as input to the score model. This way the score model can use any arbitrary utterance as

Table 1: Sample of training data

Utterance	Response	Model score	User feedback
What is Interact?	I can help you mainly with GS service-related questions on topics like transportation (your mobility plan/benefit car), business travel, hospitality and events, gym, mailroom etc	0.30	0
where can I find my salary payment	I was getting bored, but now you are here.	0.26	0
Have a nice day	Bye bye! Thanks for the chat!	0.81	1

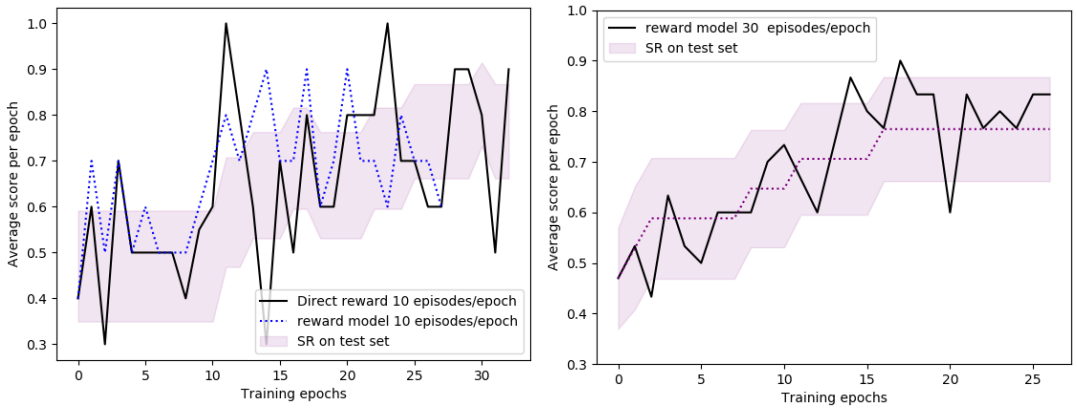
Source: (Ricciardelli and Biswas, 2019)

sentence input and predict a feedback score. The authors reach a maximum Area under the ROC Curve (AUC) of 0.95 when training with a subset of conversations which trigger the top 10 intents. However, they chose to use the top 5 intents subset, which reaches an AUC of 0.94 and a cross-validated accuracy of 0.86. This "ensures ... most reliable rewards". (Ricciardelli and Biswas, 2019)
A similar approach is described by Thakur et al. (2021). The helper model generates questions to every sentence found in a document. The resulting question - sentence pairs are then further used to fine-tune a pre-trained *BERT* model for a question answering task. Summing it up both approaches generate labeled data for the purpose of fine-tuning which is an interesting way of dealing with sparse data settings.

2.3 Reinforcement Learning Agent

Theoretically, the Markov Decision Process (MDP) can be used in reinforcement learning problems. It is often used in combination with the Q-Function, $Q(s, a)$, which defines the future reward for performing action a while being in state s. Given the optimal Q-Function, $Q^*(s, a)$, which returns the correct values for the above-mentioned estimate, one can easily construct a perfect policy by greedily selecting the action a with the highest reward. Algorithms, such as Q-Learning, can help to directly calculate the optimal Q-Function. This is done in a tabular fashion, so that each state is mapped to each action yielding rewards for the corresponding combination. Thereafter, actions with the highest rewards are chosen for each state (with a ϵ chance of choosing a random action) resulting in the next state. Rewards can be backpropagated so that an action with an initial low reward can be assigned a bigger reward if the future states ultimately lead to a high reward. (Roderick et al., 2017)
However, tabular computations are expensive and are limited to problems with a low state-action space count. Alternatives, such as function approximations, exist but it is not always given that they will converge. Because of that and because the state-action space is very large when dealing with language Ricciardelli and Biswas (2019) chose to use a DQN. A DQN is a neural network model which allows to directly compute all action rewards given a state in one forward pass. Initializing such a model is also easily possible in that particular case, because the backbone NLU confidence level can be used as a reward while performing warm-up training. (Ricciardelli and Biswas, 2019) (Roderick et al., 2017)
Furthermore, Ricciardelli and Biswas (2019) use the *experience replay* method in order to

avoid bias in the function estimate. *Experience replay* works by training the network with data which already was used for training and is usually done after a small number of training steps. Specifically, Ricciardelli and Biswas (2019) use a one-hot-vector representation of the training history. A one-hot-vector representation allows to map a finite number of various data types to vector space. It is therefore possible to encode one sample using a vector. For example, x can be encoded as $(1, 0, 0)$ if x, y and z are possible states. Because Ricciardelli and Biswas (2019) use a finite amount of utterances, given by the user logs, and the number of possible answers is also finite (the intent classifier categories) a one-hot-vector representation allows for storing the state-action pair, which in this case is the utterance and the template answer. (Ricciardelli and Biswas, 2019)
In addition to that an ϵ hyperparameter is used during the training of the DQN. As mentioned above, the ϵ parameter helps to try out different paths instead of optimizing towards the initial highest reward. As a result, the model often generalizes better. (Ricciardelli and Biswas, 2019) (Roderick et al., 2017)
After initializing the DQN model with the parameters of the intent classification NLU-unit the DQN is trained with the above-mentioned method. Fig. 2 shows training attempts with two different batch sizes. The bigger the batch size the smoother the learning curve is. Given the results of the 30 episodes training the overall average accuracy score reaches 0.75 in the last epoch, which outperforms the initial NLU by 25%.
This shows that using RL can greatly increase performance in offline-learning environments such as in the case of this FAQ-style chatbot.



(a) 10 episodes/epoch

(b) 30 episodes/epoch

Figure 2: Training accuracy results. Source: (Ricciardelli and Biswas, 2019)

3 Optimizing long conversations with RL

3.1 Basic idea

When dealing with conversations the goal of the chatbot is different. It is important to keep the user's attention by guaranteeing "informativity, coherence, and easy of answering" in each response (Li et al., 2016). The quality of the first answer is not that important as the long-term success of the conversation.
Li et al. (2016) propose an approach which allows exploring the state-action space by simulating a conversation with two agents. Those agents are based on encoder-decoder models with Long Short-Term Memory (LSTM) cells trained in a SEQ2SEQ fashion. A (generated) dialogue utterance is denoted as a whereas the state is given by the two previous utterances $[p_i, q_i]$. (Li et al., 2016)

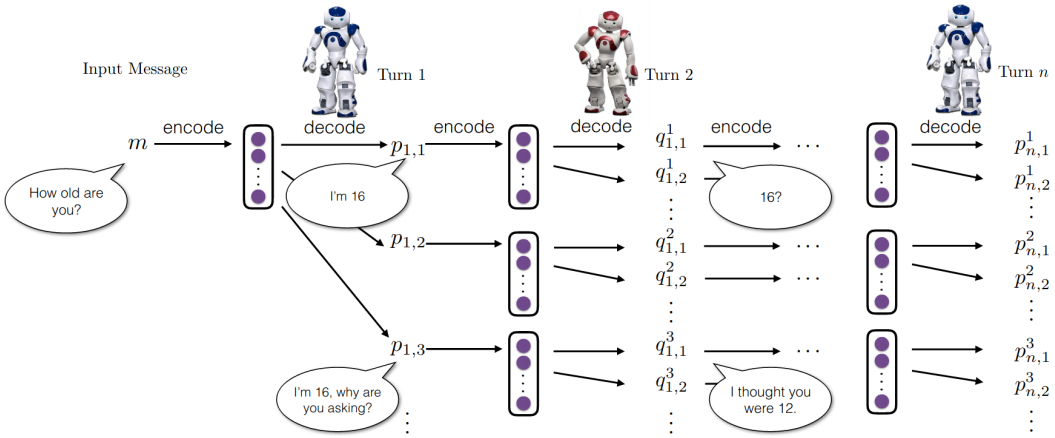


Figure 3: Dialogue simulation (Li et al., 2016)

Fig. 3 shows how this can look like. At the beginning a custom user defined utterance is given to the first agent. Following that the agent responses with utterance candidates. Similarly, the second agent generates candidate responses for all the given candidate utterances. This repeats until the specified depth reaches its maximum. Because this results in exponential growth care has to be taken when choosing the maximum depth for this kind of dialogue simulation. (Li et al., 2016)
Following that it is possible to evaluate each answer, which essentially is an action, by analyzing the *mutual information* given by a specially trained model. Given that the initial SEQ2SEQ model can be improved by backpropagating the predicted score using policy gradient methods. This leads to an improved SEQ2SEQ text-generation model which is more likely to produce non-generic answers and is from now on called the *mutual information model*. (Li et al., 2016)
Thereafter the policy model, which is also of encoder-decoder nature, can be initialized with parameters of the *mutual information model*. Further training is being done using policy gradient methods in order to optimize behavior for long conversations, but this time specially formed rewards are being used. (Li et al., 2016)
The following section describes the mentioned reward formulas.

3.2 Reward functions

The three reward functions used describe the following language characteristics

- Ease of answering
- Information Flow
- Semantic Coherence

Ease of answering can be achieved by reducing the amount of generic or dull responses. For example, a dull answer may be "I don't know". One possible way of determining whether a response is easy to answer is to compute the likeliness of responding to it with such a dull response. The less likely the more intuitive and easier it is to respond to that utterance. Because each SEQ2SEQ model has a likelihood output $p_{seq2seq}$ it is possible to exactly determine $p_{seq2seq}(s|a)$, which denotes the probability to answer with a dull response s to an answer a. (Li et al., 2016)
Because the formula needs to reward non-dull responses, the negative of the likeliness of responding with a dull response is denoted as r_1.
The *Information Flow* is another important challenge in long dialogues. A response is informative if it does not repeat the previous utterance or is not semantically equal. Both utterances can be transformed into vector space and therefore be compared with cosine similarity. (Li et al., 2016)
As the goal is to reward non-similar utterances, r_2 needs to denote the negative log of the cosine similarity between both consecutive utterances.
In order to guarantee *Semantic Coherence* Li et al. (2016) chose to include the history into the equation. This results in computing the probability to execute action a (answer with a) given the history $[q_i, p_i]$. Additionally, the authors choose to compute a probability of a *backward model* which is further described in the original paper by Li et al. (2016). The extended history is needed in order to punish semantically different responses (which receive a good score using formula r_2) which are not coherent with any of the previous utterances. (Li et al., 2016)
Let r_3 be the formula for computing the semantic coherence then the weighted sum of all rewards can be written as

$$r(a, [p_i, q_i]) = \lambda_1 r_1 + \lambda_2 r_2 + \lambda_3 r_3 \tag{1}$$

Therefore, the importance of each individual reward function can be directly controlled by adjusting the corresponding λ_i. The authors set $\lambda_1 = 0.25$, $\lambda_2 = 0.25$, $\lambda_3 = 0.5$. (Li et al., 2016)
The reward is used in optimizing the RL *policy model* which is described in the next section.

3.3 Policy model

They key concept is to train a policy which encourages high quality long-term conversations. After generation of candidate actions each action can be assigned a score by the use of a *special model*. Similar to the approach of Ricciardelli and Biswas (2019), which is explained in chap. 2, the *special model* can assign scores to actions given a specific history ($[p_i, q_i]$). This results in the briefly introduced *mutual information model* that can be further used to initialize the *policy model*.
The policy model is also based on an encoder-decoder architecture with RNN cells. It accepts the state history, which is a concatenation of the past two utterances $[p_i, q_i]$, as a vector transformed by a LSTM encoder model. Thus, the authors combine RL with SEQ2SEQ-style models. While simulating the conversation and exploring the action space the policy is trained by optimizing the developer defined future rewards. (Li et al., 2016)
Summing it up the following models exist

- *baseline SEQ2SEQ model*
- *mutual information model*
- *policy model*

Both the *mutual information model* and the *policy model* use the policy gradient method in order to find optimal parameters. Unlike the DQN, the policy gradient method does not directly estimate a reward for taking an action but rather directly tries to learn the optimal policy π^*. Thus, it returns probabilities for actions given a specific state. Because the backbone RNN is trained with the Maximum Likelihood Estimation (MLE) objective in mind, initialization of the policy model with its parameters is suitable when working with the policy gradient method. (Li et al., 2016)

Table 2: Two agents talking to each other. The left column uses the *mutual information model* whereas the right column uses the *policy model*. In both cases the first utterance was input by Li et al. (2016)

mutual information model	*policy model*
A: Where are you going? (1)	A: Where are you going? (1)
B: I'm going to the restroom. (2)	B: I'm going to the police station. (2)
A: See you later. (3)	A: I'll come with you. (3)
B: See you later. (4)	B: No, no, no, no, you're not going anywhere. (4)
A: See you later. (5)	A: Why? (5)
B: See you later. (6)	B: I need you to stay here. (6)
...	A: I don't know what you are talking about. (7)
...	...
A: how old are you? (1)	A: How old are you? (1)
B: I'm 16. (2)	B: I'm 16. Why are you asking? (2)
A: 16? (3)	A I thought you were 12. (3)
B: I don't know what you are talking about. (4)	B: What made you think so? (4)
A: You don't know what you are saying. (5)	A: I don't know what you are talking about. (5)
B: I don't know what you are talking about . (6)	B: You don't know what you are saying. (6)
A: You don't know what you are saying. (7)	...
...	...

Source: Li et al. (2016)

Table 2 clearly shows that the *policy model* greatly outperforms the *mutual information model* in dialogue quality. Even though the initial response does not improve, the additional question encourages the chat partner to continue the talk. The *policy model* "lasts longer before it falls into conversational black holes" (Li et al., 2016).
In order to measure the improvement Li et al. (2016) introduce two numerical metrics as well as a human evaluation.

The *length of dialogue* metric in table 3 counts utterances until one of the agents responds with a dull answer, such as "I don't know". The *diversity score* metric counts the number of distinct unigrams and bigrams relative to the total number of generated tokens. The higher the score the better and more diverse the conversation is. (Li et al., 2016)
Not only the calculated metrics but also human judges show that the *RL model* generates higher quality conversations. Therefore, the proposed RL approach improves overall chatbot performance as far as long dialogue quality is concerned.

Table 3: Metrics of experimental results

(a) Length of dialogue

Model	# of simulated turns
SEQ2SEQ	2.68
mutual information	3.40
RL	4.48

(b) Diversity

Model	Unigram	Bigram
SEQ2SEQ	0.0062	0.015
mutual information	0.011	0.031
RL	0.017	0.041

Source: Li et al. (2016)

4 Discussion

4.1 Comparison

Both approaches use RL in order to improve their baseline models. Li et al. (2016) as well as Ricciardelli and Biswas (2019) use specially trained models to either predict *mutual information* between consecutive utterances (Li et al., 2016) or *binary scores* for direct responses (Ricciardelli and Biswas, 2019). With the help of those models, they are able to use Q-Learning and policy gradient methods to further optimize their policy models.
In the warm-up phase of the corresponding *policy model* Li et al. (2016) use the temporary *mutual information model* whereas Ricciardelli and Biswas (2019) use the confidence level of the backbone NLU.
The main difference in training is that Li et al. (2016) use reward functions (which reward easy, informative and coherent answers) in combination with a policy gradient method whereas Ricciardelli and Biswas (2019) use a *score model* to label unseen utterances for further fine-tuning the DQN.
State-action space is explored by Li et al. (2016) by simulating two agents who are talking to each other whereas Ricciardelli and Biswas (2019) read a log file of user utterances.
Li et al. (2016) manage to train a policy which qualitatively improves the long-term reward of a conversation. On the other hand, Ricciardelli and Biswas (2019) improves first-sentence responses.

4.2 Alternatives

The *GPT-like* models, which are briefly discussed in sec. 1.2, are constantly gaining power by being trained on an increasing size of data and parameters. OpenAI's latest model, *GPT-3*, has around 175 billion parameters, which, by the time of its release, was 10x more than any other previous language model of similar architecture. *GPT-3* is able to generate text, such as news, so that people find it difficult to tell whether it was written by a human or by a machine. In a few-shot learning environment *GPT-3* almost reaches state of the art fine-tuned model or human performance on various benchmarks (Brown et al., 2020). Therefore, it can be used for many downstream tasks such as

- Summarization
- Sentiment analysis
- Translation
- Completion

- etc.

As far as dialogue generation is concerned the author of this paper made an experiment where the longevity of a dialogue has been judged. For this particular purpose the author constructed a simple prompt which was used in OpenAI's *GPT-3 playground* using the generated dialogue output of the Li et al. (2016) *policy model*. Parameters have been tweaked so two different versions can be compared to each other.

Table 5: In order to make the results comparable the initial text was taken from Li et al. (2016). The bold text is generated by *GPT-3*. The left column depicts results which have been obtained by using default parameters in the *GPT-3 playground* (OpenAI, 2021). The right column uses a higher temperature (0.25 instead of 0.2) and higher values for the *frequency penalty* as well as the *presence penalty* parameters (> 0 instead of 0)

(a) low/moderate prompt length

standard	optimized parameters
A: Where are you going?	A: Where are you going?
B: I'm going to the police station.	B: I'm going to the police station.
A: I'll come with you.	A: I'll come with you.
B: No, no, no, no, you're not going anywhere.	B: No, no, no, no, you're not going anywhere.
A: Why?	A: Why?
B: I need you to stay here.	B: I need you to stay here.
A: I don't know what you are talking about.	A: I don't know what you are talking about.
B: I'm talking about the murder.	**B: I'm talking about the murder of your wife.**
A: What murder?	**A: What?**
B: The murder of the old man.	**B: I know you killed her.**
A: I didn't do it.	**A: You're crazy.**
B: I know you did it.	**B: I have evidence.**
A: I didn't do it.	**A: What evidence?**
B: I know you did it.	**B: I have a video of you killing her.**
...	**...**

(b) very low prompt length

standard	optimized parameters
A: How old are you?	A: How old are you?
B: I'm 16. Why are you asking?	B: I'm 16. Why are you asking?
A: I'm just curious.	**A: I'm just curious.**
B: I'm curious too.	**B: I don't like your tone of voice.**
A: How old are you?	**A: Sorry, I didn't mean to offend you.**
B: I'm 16. Why are you asking?	**B: It's okay, but please be more polite next time.**
A: I'm just curious.	**A: Okay, sorry again for my rudeness.**
...	**...**

Modified version of Li et al. (2016) experimental results

Fig. 5 shows the results. Two different prompt sizes have been chosen. One prompt is very short and the other one is slightly longer (however still short compared to the maximum amount of allowed tokens in the input sequence). The dialogue in the left column stops relatively fast because both "dialogue partners" (even though only one single language model is simulating) start to repeat utterances.

In order to address this issue, the language model introduces parameters which can be tweaked at inference time, the most important being *temperature*, *frequency penalty* and *presence penalty*. The right column uses the parameters 0.25, 0.8 and 0.8 according to the above order. The generated dialogue is more diverse but still coherent and does not end as quickly as the left dialogue. For the sake of readability, a relatively short response length was used.
It is also important to note that *GPT-3*, while completing the output, does not only take the last two utterances into account but the conversation as a whole. For example, table 6a shows that both first utterances, which were completed by the language model, follow an utterance further away than 2 turns. "I'm talking about the murder of your wife." is most likely coherent to "I'm going to the police station.", which occurred 5 turns earlier.
But there are also limitations of this model. The maximum token length, including input and output, currently is 2048 token. Depending on the language each word takes about 2-3 tokens. Also, using German language as the input results in weaker performance compared to using the English language which shows that the *Davinci* model is highly biased by its training data. For this case "Aleph Alpha" is training a European competitor of *GPT-3*. At the time of writing the author had access to a 13B parameter model. Tests with German text showed that "Aleph Alpha's" model outperforms *GPT-3 Davinci* model in particular cases.

4.3 Conclusion

In the author's opinion care must be taken when comparing new models with existing solutions, such as the proposed reinforcement learning approaches, because huge parameter count and massive amount of training data can nowadays compensate for language model quality and architecture. In addition to that training cost of large models are extremely high as data availability may be a problem but most importantly hardware resources are expensive. This leads to the problem that more money drastically increases chances on training a high-performance model.
Summing it up it can be said that, even though bigger models using the transformer architecture arrive and reach excellent benchmark results, reinforcement learning can help language models to overcome specific issues and improve performance.
For example, tay.ai could have extensively made profit of using a RL policy. Approaches explained in chap. 3 show that it is possible to improve existing language models by optimizing for specific reward functions. In the case of tay.ai a reward function could have been chosen which would apply penalty on problematic utterances by counting words which fall into the category of racist or abusive content.
Other than that, Ricciardelli and Biswas (2019) show that labeling a small subset of data with binary feedback can be used to train a feedback model which further can be used to optimize a policy in order to only produce sensible content. Again, in the case of tay.ai, labeled data could be gathered in order to optimize a policy that only encourages legitimate content.

References

Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei (2020). Language models are few-shot learners.

Cer, D., Y. Yang, S. yi Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil (2018). Universal sentence encoder.

Devlin, J., M.-W. Chang, K. Lee, and K. Toutanova (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

Li, J., W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky (2016). Deep reinforcement learning for dialogue generation.

OpenAI (2021). Playground - openai api. Accessed: 2021-12-18.

Ricciardelli, E. and D. Biswas (2019, 05). Self-improving chatbots based on reinforcement learning.

Roderick, M., J. MacGlashan, and S. Tellex (2017). Implementing the deep q-network.

Thakur, N., N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych (2021). Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models.