Policy Gradient Methods for Dialogue Response Generation

Michael Noseworthy Nicolas Angelard-Gontier
School of Computer Science
McGill University
Montreal, QC
{michael.noseworthy, nicolas.angelard-gontier}@mail.mcgill.ca

Abstract

Dialogue systems have recently seen a surge of progress from the use of deep generative models. These models try to maximize the likelihood of a dataset as their training objective. However, this objective may not always represent what we want the network to learn. In this work we explore using the policy-gradient framework from the Reinforcement Learning field to maximize an alternative objective: the *ADEM* score. We compare our results for the following algorithms: *REINFORCE*, *REINFORCE* with value baseline, *Actor-Critic*, and *Actor-Critic* with value baseline (or *Advantage Actor-Critic*). Due to the difficulty of the dialogue response generation task, we also consider a time series toy task. In the dialogue domain, we find that although we can optimize the *ADEM* score successfully, this is done by exploiting the weaknesses of the *ADEM* metric.

1 Introduction

With the availability of large dialogue corpora, building deep generative models for response generation has become a popular method to scale dialogue systems beyond simplistic domains. In such scenarios a model is given only a history of the dialogue in text form (with no other domain-specific information) and is tasked with generating a "high" quality response.

End-to-end dialogue response generation models are most commonly trained using a maximum-likelihood objective. However, there have been many observed problems with this method. When trained this way, models will generate uninformative responses such as "I don't know" or "Thanks". Although technically correct (these are responses that make sense and occur frequently in our dataset), these responses are usually not desired by users who interact with the agent. Furthermore, when trained on maximum-likelihood, dialogue agents suffer from cascading errors during test time.

Li et al. (1), have proposed various ways to tackle the problem of generic responses. We will describe the approaches in section 2, but they largely consist of modifying the training objective to maximize scores according to hand-crafted heuristics and training using variants of *REINFORCE*.

In previous work, instead of relying on hand-crafted heuristics, Lowe et al. learned "A Dialogue Evaluation Metric" (*ADEM*) (2), which produces a score from 1 to 5 that represents the quality of a response to a given context. *ADEM* was trained using human scores and thus the scores represent a general notion of quality. One of the motivations for training *ADEM* in such a way was so that we can train a generation model to maximize this score, without worrying about how to combine multiple heuristics.

Our first contribution is to train a model to maximize the *ADEM* score. In doing so, we can conduct an analysis of the quality of the metric and ideally avoid the limitations of the maximum-likelihood objective. Our second contribution is in how we train the model. In order to train our model with respect to a non–differentiable metric, we borrow methods from the policy-gradient framework.

Instead of using just *REINFORCE*, we consider actor-critic methods (inspired by Bahdanau et al. (3)). We hope that by introducing some bias, we can greatly reduce the variance suffered by having a large action space (the size of our vocabulary), and a sparse reward (the *ADEM* score).

2 Related Work

2.1 Dialogue Generation

Generative dialogue models aim to provide a coherent response to some dialogue history. With the rise of deep learning, neural architectures consisting of *Recurrent Neural Networks* (*RNNs*) are the state of the art in this domain (4; 5; 6; 7). There are two kinds of dialogue generation models: retrieval based, and purely generative. The first will encode a context and return an utterance from the training set that best fits the conversation (5). The latter is usually an encoder-decoder architecture where the context is encoded and the decoder outputs the next tokens (7). Both of these method are usually trained to maximize the log–likelihood of the training data, and are thus prone to generate short utterances that close the conversation such as "I don't know" or "Thanks".

Recently, an effort has been made to overcome this issue by training models to maximize hand-crafted rewards. Li et al. (1) define three measure that their generator should maximize: ease of answering, information flow, and semantic coherence. Each of these metrics were defined by heuristic computations, and the final reward signal was a fixed combination of the three characteristics. Although intuitively these heuristics make sense, no study was performed to see if they correlate with how humans would evaluate the quality of a dialogue response. Furthermore, due to variety of plausible dialogue responses, we cannot hope to cover all the possible characteristics of a good response with hand–crafted rewards.

Another alternative is to train a dialogue generative network using an adversarial setup (8). Here the authors used a discriminator network in a reinforcement learning manner to generate better answers. Although a promising approach, we do not pursue this avenue.

2.2 ADEM

In previous work, Lowe et al. developed "A Dialogue Evaluation Metric" (*ADEM*) (2), which produces a score from 1 to 5 that represents the quality of a response to a given context. This model consists of three encoder *RNNs*: one for the context c, one for the true response y, and one for the proposed response $\hat{y}$ (coming from a generative model). The encoders have shared weights which were pre-trained in an encoder-decoder manner to maximize the likelihood of the data. Eventually the score is given by

$$score(c, y, \hat{y}) = (c^T M \hat{y} + y^T N \hat{y} - \alpha)/\beta$$

where matrices M and N are learned by the model and α, β are scalar constants used to make the model's predictions closer to $[1, 5]$. *ADEM* was trained using human scores in the range $[1, 5]$ and thus the scores represent a general notion of quality. As described in the following sections we will use this scoring function as a reward signal for our network.

3 Dialogue in the Policy Gradient Framework

When we train our generative model to maximize the *ADEM* score, we no longer have a differentiable metric like we did in the maximum–likelihood scenario. Instead, we adopt the policy-gradient framework to give us gradients that maximize our score. We use standard methods from the field to reduce the variance in our updates (i.e. employing a baseline and using a critic).

Similar work has been done which uses actor-critic methods for machine translation (3) (where the authors try to maximize the *BLEU* score).

3.1 Problem Formulation

We consider a scenario where we are given a context c, and are tasked with generating a response, $\hat{y} = \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T\}$. We are also given a reference response, y. This is a response that actually occurred after c in our data set.

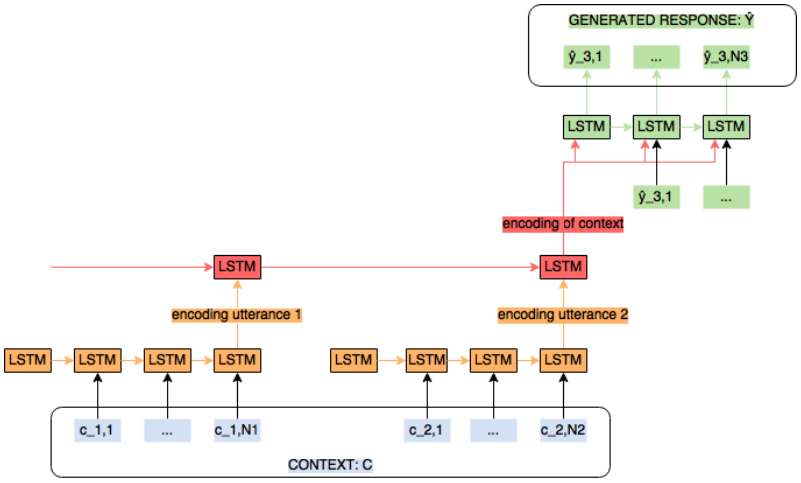


Figure 1: Hierarchical Recurrent Encoder-Decoder architecture used for the actor network.

Each token of the response $\hat{y}_t$ is considered an *action*. The *state* is the context c, and all the tokens generated up until the current time step, $\hat{y}_{1\ldots t-1} = \{\hat{y}_1, \ldots, \hat{y}_{t-1}\}$. Our reward at each time step, r_t, is 0 if $t \neq T$ and *ADEM*$(c, y, \hat{y})$ if $t = T$. Thus we are working with a sparse reward signal. We use a discount factor of $\gamma = 1$.

We use a parametrized policy $\pi(\hat{y}_t|\hat{y}_1 \ldots \hat{y}_{t-1}, c, \theta)$. Our objective is to maximize the expected *ADEM* scores of the responses generated by this policy. Let $J(\theta)$ be our expected return. Next, we will describe how we implement the actor and critic architectures.

3.2 Actor Network

Our actor is the *Hierarchical Recurrent Encoder-Decoder (HRED)* model previously used for dialogue response generation (6). The low-level encoder, which we refer to as the *turn encoder*, is a *RNN* that encodes a single turn of the context. The high-level encoder, the *context encoder*, is another *RNN* that takes the embeddings from the *turn encoder* as input. We end up with a single embedding for the entire context, c_{emb}.

In the decoder, at each time step we model $p(a_t|\hat{y}_{1\ldots t-1}, c_{emb})$ by another *RNN* where a_t represents all the possible tokens we could emit at time step t. The output layer is a softmax over the tokens of our vocabulary and thus we can view this as our policy. See Figure 1 for a depiction of this model.

We consider four different updates to maximize our return with respect to our policy. The first is simply *REINFORCE*. The second uses a value function baseline. The key difference with typical baselines is that we also give our baseline access to the true response, $\hat{V}(c, \hat{y}_{1,\ldots,t}|y, \psi)$.

When doing actor-critic updates, we also give our critic, $\hat{Q}$, access to the true response. We show the gradient for the advantage actor-critic update here since it is the most complex:

$$\nabla J(\theta) = \sum_{t=1}^{T} \nabla \log \pi(\hat{y}_t|\hat{y}_{1\ldots t-1}, c, \theta) \left[\hat{Q}(\hat{y}_t|\hat{y}_{1\ldots t-1}, c, y) - \hat{V}(\hat{y}_{1\ldots t-1}, c|y, \psi)\right] \tag{1}$$

The next sections explains how we learn the critic to be used as a baseline or for the actor-critic method.

3.3 Critic Network

The critic's goal is to predict meaningful q-values to give feedback to the actor. These q-values should be an approximation of the reward function R. To do so, the network takes as input the current context embedding c_{emb}, the true response embedding y_{emb}, and the predicted response $\hat{y}$ produced by the actor. Note that c_{emb} and y_{emb} are encoded using the actor's hierarchical encoder. This greatly reduces the number of parameters for the critic, making it easier to learn the q-value approximation.

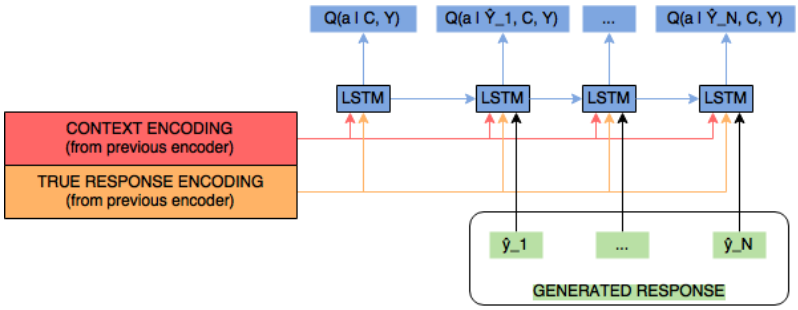


Figure 2: Recurrent Decoder architecture used for the critic network.

The critic network is essentially a decoder *RNN*. At each time steps it receives as input a predicted token $\hat{y}_t$ and models $\hat{Q}(a_{t+1}|\hat{y}_{1,\dots,t}, c, y)$ where a_{t+1} represents all the possible tokens the actor could emit in the next round. See Figure 2 for a depiction of this model. It is important to mention that in order for $\hat{Q}$ to better estimate the true Q-function Q, we gave the true response y as input to the critic at each time step. Since y is also required by R (e.g. *ADEM*) we believe that this is a reasonable design decision which will help our critic.

To approximate Q, we set the training objective of the critic to a one-step TD estimate:

$$q_t = r_t(\hat{y}_{1\dots t}|y) - \bar{r} + \sum_{a \in Vocab} \pi(a_{t+1}|\hat{y}_{1\dots t}, c_{emb}) * \hat{Q}(a_{t+1}|\hat{y}_{1\dots t}, c_{emb}, y_{emb}) \tag{2}$$

where $\pi(a_{t+1}|\hat{y}_{1,\dots,t}, c)$ is the policy of the actor, $\hat{Q}(a_{t+1}|\hat{y}_{1,\dots,t}, c, y)$ is our current estimate of the next state-action value, and $r_t(\hat{y}_{1,\dots,t}|y)$ is the reward that we received by taking action $\hat{y}_t$. Note that we subtract by the mean reward over all time steps to reduce variance in the reward signal. This is of particular interest since *ADEM* only gives a score at the end of a generated response.

The training objective of the critic is then to minimize the squared error loss between the target defined above and its current estimate:

$$J(\psi) = \sum_{t=1}^{T} (q_t - \hat{Q}(\hat{y}_t|\hat{y}_{1\dots t-1}, c_{emb}, y_{emb}))^2 + \lambda C_t \tag{3}$$

where C_t is a regularization term that reduces the variance of the emitted Q-values by bringing them closer to their average:

$$C_t = \sum_{a \in Vocab} \hat{Q}(a|\hat{y}_{1\dots t-1}, c, y) - \frac{\sum_{b \in Vocab} \hat{Q}(b|\hat{y}_{1\dots t-1}, c, y)}{|Vocab|} \tag{4}$$

As mentioned in previous work (3), this trick prevents actions that are rarely sampled from having artificially high Q-values and is particularly useful with large action spaces.

3.4 Training

The general training loop of our algorithms is as follow:

- Given a context–response pair (c, y), the actor generates $\hat{y} = \{\hat{y}_1, \dots, \hat{y}_T\}$.
- Given the triple $(c, y, \hat{y})$, we get rewards $r(\hat{y}_1|y), \dots, r(\hat{y}_{1\dots T}|y)$.
- Depending of the algorithm, we then compute the returns (for *REINFORCE* updates), or we ask the critic to produce q-value estimates (for *Actor-Critic* updates, or in order to compute the estimated value function $\hat{V}(\hat{y}_{1\dots t-1}, c|y, \psi)$ when using a baseline).
- We update the actor parameters θ using the appropriate policy gradient update that can be generalized with Equation 1.
- We update the critic parameters ψ by taking the gradient of $J(\psi)$ described in Equation 3.

We repeat the above training loop over batches of data and report our results on different domains in the next sections.

4 Experiments

We now describe two experiments we performed. The first is a toy domain - partly to verify our implementation, and partly to compare different characteristics of various model in a controlled environment. The second is the full dialogue response generation task. All our experiments were implemented in *Theano* (10), and used *Adam* (9) as the optimizer. We detail the hyper-parameters and configurations of the actor and critic network for each domain in Appendix A.

4.1 Toy Domain

The toy domain we consider is a domain where we reward the actor for generating a fixed token y^* from a fixed vocabulary of size $|V|$. Thus the model has to explore until it finds the correct token. More formally, $r_t = 1$ if $\hat{y}_t = y^*$ and $r_t = 0$ otherwise. We limit the number of tokens in a response that can receive a reward to be the same length as the given context (which otherwise is not used). We set $|V|$ to 20 and 5000 to compare the effects of the size of the action space on a simple task. Furthermore, we also consider the case when the true response y given to our critic network is the best possible response or not. From the reward function we just presented, the ideal response to a context c of length n would be $y = \{y^*, y^*, \ldots, y^*\}$ with $|y| = n$. We did our experiments with y set to the ideal answer and with y set to a random answer to compare the effects of the true response condition we introduced on the critic architecture.

4.2 Dialogue Domain

The dialogue domain is the problem formulation we introduced in Section 3.1. We use a data set collected from Twitter where each dialogue is a sequence of Tweets between two users. The data set consists of $749,060$ dialogues. As described in Section 3.4, we train in a batch mode where each batch consists of a list of contexts with their appropriate reference responses. The actor then generates a response conditioned on this context which is scored by the *ADEM* model. Finally, we calculate the batch updates depending on which update rule we are using.

Due to the size of the action space in natural language (typically the size of the vocabulary is greater than $20,000$), we use *Byte Pair Encoding* (*BPE*) to reduce the dimensionality to $5,000$. *BPE* was applied to natural language as a way to generate sub-word level tokens (11). It works by initializing its vocabulary to just character tokens and iteratively adds new tokens of longer length that appear frequently in the data set. Thus each word can be represented by one or more *BPE* token. Along with reducing the size of our action space, *BPE* also means we will not have out-of-vocabulary words.

We recognize that the metric *ADEM* has its shortcomings and it is unlikely to be a strong enough training signal to generate complete responses from scratch. Thus we initialize the actor to a model that was pre-trained with a Maximum-Likelihood (*ML*) objective. This defaults to the *HRED* model (6) with the following objective (notice here we are simply maximizing the likelihood of the reference response):

$$J(\theta) = \sum_{t=1}^{T} \log p(y_t | c, y_{1\ldots t-1}) \tag{5}$$

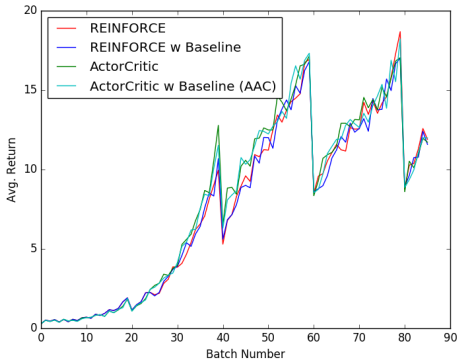
We employ two different training schedules. The first only optimizes the *ADEM* score and the second alternates between the *ADEM* and *ML* objectives every 250 batches. The motivation behind the latter approach is to make sure our policy stays close to the maximum likelihood solution if the *ADEM* metric causes it to deviate too much.

Note that we only consider *REINFORCE* with Value function baseline here as it was able to optimize the desired objective and we did not have enough GPU time to extensively evaluate all the actor-critic methods. We plan to continue these experiments in future work.

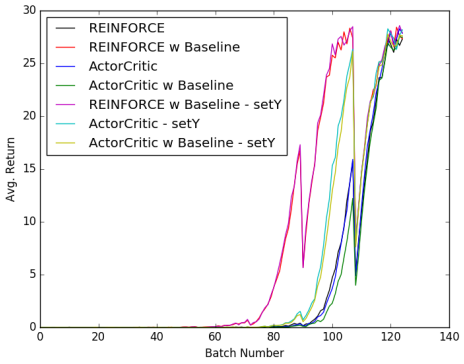
5 Discussion

5.1 Toy Domain

As a sanity check we ran the four different algorithms on the small toy domain where the size of the action space (size of the vocabulary) was set to $|V| = 20$. We can see from Figure 3a that all the



(a) Small toy domain: $|V| = 20$.



(b) Large toy domain: $|V| = 5000$ & use the ideal answer $y = y^*, \ldots, y^*$ in " - setY" plots.

Figure 3: Average return/batch for training with REINFORCE, REINFORCE with value baseline, Actor-Critic, and Actor-Critic with value baseline (AAC).

algorithms properly maximize the expected reward after a few batches. It is important to mention that the oscillations in the plots are due to the fact that we sort our batches by context length. Since the reward function (described in section 4.1) depends on the length of the response, this variation in the return signal is expected. Due to the simplicity of the task, we see that all algorithms learn at the same speed. We also note that the decision of setting y to the ideal answer or not in our critic didn't change the learning curves. We thus explore a larger toy domain.

In our second toy domain we increased the action space (vocabulary size) to $|V| = 5000$. Note that for the same reason as in the small toy domain, the oscillations in the return signal is expected. From this experiment we get two key observations. First we can see from Figure 3b that *REINFORCE* with a Value function baseline is the fastest algorithm to find the rewarded token y^*. In addition, it performs as efficiently when y is set to a random and useless response. This is a strong result for the *REINFORCE* with value baseline algorithm as it performs better than our *Actor-Critic* methods. Perhaps this is because it is hard to learn a good critic in a large action-space. This would mean that it's okay to use the critic for just the value function because it adds no bias, but it may not help when we use the critic to bootstrap. Future work would involve using delayed weights to update the critic estimates.

Second, we notice from Figure 3b that in the case of *Actor-Critic*, having the value baseline doesn't influence the learning, however, having the response set to the optimal solution according to the reward function benefits the learning. Indeed, when training the critic with $y = \{y^*, \ldots, y^*\}$ the actor sees its return increasing after epoch 90. On the other hand, training the critic with y set to random tokens performs as bad as the simple *REINFORCE* algorithm, which has to wait until epoch 100 to have increasing returns. In this case $\hat{Q}$ can be seen as a teacher. Including an optimal trajectory in the critic deserves further investigation. In the dialogue domain, we include it since the true reward model also has access to y (although y may no longer be the optimal trajectory according to our reward model: *ADEM*).

5.2 Dialogue Domain

There are two key observations we would like to make about training using the *ADEM* score as our reward model. First, as seen in Figure 4, *REINFORCE* with a Value function baseline allows us to improve the *ADEM* score in a reasonable number of training batches.

At first glance this may look promising, but after inspection of the samples being generated, we see that the actor is in fact not generating higher "quality" results. Instead, the actor learns to exploit the weaknesses of *ADEM* by generating either nonsense responses (which *ADEM* was not trained on and will not generate reliable scores for), or simple responses (that generally get higher scores). Indeed, when trained with just *REINFORCE*, the actor converges to generating "<second_speaker> <at> hey </s>".

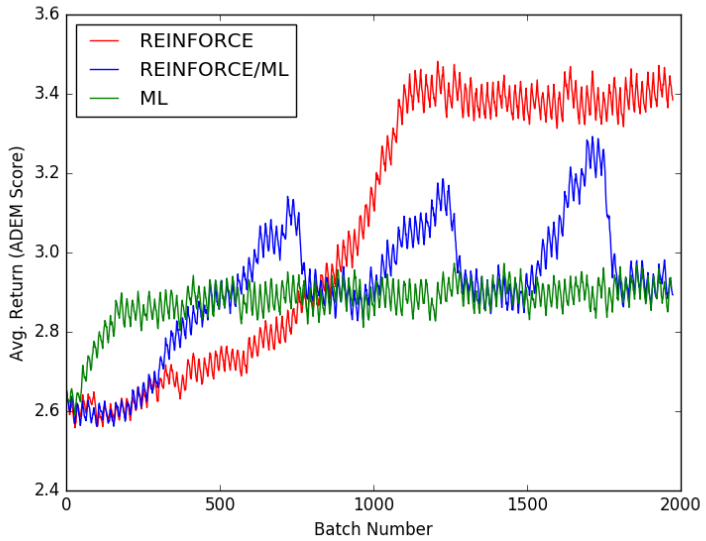


Figure 4: Average ADEM score/batch for training with ML and REINFORCE objectives (or a schedule of the two).

Batch #	Context	Sampled Response	*ADEM* Score
950 (ML)	<first_speaker> the tickets to our brisbane, melbourne and adelaide shows our selling fast grab yo copy soon <second_speaker> <at> is there door tickets for brisbane?!	<second_speaker> <at> your two on the id? mal just only sanc ryan bkg.	2.69
1200 (PG)	<first_speaker> coach cronin is reportedly leaving northeastern, possibly for the ahl providence bruins from <at>.	<second_speaker> <at> it's hot, hurt, here g.	3.96
1700 (PG)	<first_speaker> i can finally see the appeal of twitter - i haven't had to say a word to <at> all night, put the kettle on love	<second_speaker> <at> <at> <at> <at> hey!	5.47

Table 1: Samples generated according to the actor at different periods during training. ML or PG specify the objective being optimized at the time of the sample.

Including the schedule between *ML* and *REINFORCE* training objectives did not help the aforementioned problems. In Figure 4, we alternate between the two objectives every 250 batches (starting with the *REINFORCE* objective). When using the *ML* objective, our *ADEM* score reverts to the same level as if we only used *ML* training. When using the *REINFORCE* objective, we start to generate nonsense responses. Some sample responses from different batches can be seen in Table 1.

6 Conclusion

In this report, we show how the policy-gradient framework can be used with large action spaces which are common in natural language tasks. We demonstrate the differences between different algorithms in this framework and show that giving the critic access to a reference trajectory is a promising line of future work. Furthermore we use this framework to train a generative model to maximize the *ADEM* score for the dialogue response generation task. We observe that the actor can exploit the weaknesses of this model and gain insights in how to improve our metric.

One promising line of future work to improve the scoring model is to quantify the uncertainty of the *ADEM* score. Thus when the actor produces a response that *ADEM* has low confidence about, we can use the Maximum-Likelihood objective instead of the respective policy-gradient objective. The actor would thus learn to produce responses that *ADEM* assigns a good score with high confidence.

Another line of research would be to consider a multi-critic approach. Any single critic may not be sufficient to capture all the characteristics that make up a good dialogue response. Instead we could imagine a list of heuristics and train a critic for each: mutual information between the context and response, the *ADEM* score, and grammaticality for instance. Future research would investigate the proper way to combine the gradients from each critic in such a scenario.

Acknowledgments

We would like to thank Julian Serban for his help understanding the *HRED* code.

References

[1] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky, "Deep reinforcement learning for dialogue generation," 2016.

[2] R. Lowe, M. Noseworthy, I. V. Serban, N. A.Gontier, Y. Bengio, and J. Pineau, "Towards an automatic turing test: Learning to evaluate dialogue responses," 2017.

[3] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio, "An actor-critic algorithm for sequence prediction," 2016.

[4] R. Lowe, N. Pow, I. Serban, and J. Pineau, "The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems," 2015.

[5] R. Lowe, I. V. Serban, M. Noseworthy, L. Charlin, and J. Pineau, "On the evaluation of dialogue systems with next utterance classification," 2016.

[6] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models," 2015.

[7] I. V. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. Courville, and Y. Bengio, "A hierarchical latent variable encoder-decoder model for generating dialogues," 2016.

[8] J. Li, W. Monroe, T. Shi, S. Jean, A. Ritter, and D. Jurafsky, "Adversarial learning for neural dialogue generation," 2017.

[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016.

[11] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," 2015.

A Network Configurations

The following table specifies the configuration used for the actor and critic networks in the Toy and Dialogue tasks.

Parameter	Toy Task	Dialogue Task
Actor Network		
Token Dim.	100	400
Turn-Encoder Hidden Dim.	300	1000
Context-Encoder Dim.	300	1000
Decoder Dim.	300	1000
RNN Gating	GRU	GRU
Optimizer	Adam	Adam
Critic Network		
Decoder Dim.	300	600
C (Regularization)	0.001	0.001
RNN Gating	GRU	GRU
Optimizer	Adam	Adam