Comparing Generative Adversarial Networks to Deep RNN Language Models

David Hershey
Department of Computer Science
Stanford University
Stanford, CA 94305
dshersh@stanford.edu

Blake Wulfe
Department of Computer Science
Stanford University
Stanford, CA 94305
wulfebw@stanford.edu

Abstract

Generating long sequences of text is a challenging problem in natural language processing. Although current models using deep recurrent neural networks (RNNs) have attained high performance on the generation of short text sequences, these models break down when generating longer text. In this work we examine the use of Generative Adversarial Networks (GANs), as a learned metric of the quality of produced text with which to train a generative model. We demonstrate the effective training of this model with policy gradient methods, which circumvent the non-differentiable sampling procedure required for use with a sequence level metric. We then compare the performance of the GAN with that of a state of the art deep learning model on the task of creating a simple conversational agent.

1 Introduction

The generation of sequences of text is a challenging problem in the field of natural language processing. Text generation has a wide range of potential applications, ranging from automated summarization [8] to conversational agents [11]. These technologies could help provide news stories and customer service both more quickly and cheaply then traditional methods.

Although people value different qualities of generated text between applications, it is generally the case that the realism of the generated text, i.e., how closely it resembles text produced by a person, plays a central role. Many metrics used in NLP reflect this fact. For example, the research that originally introduced BLEU cites the correlation between the metric and human evaluators as the primary argument for its adoption [9], and the cross-entropy training loss explicitly maximizes the probability of observed data. These metrics capture different characteristics of text, so what impact does maximizing one over the other have?

Ranzato et al. recently introduced a method, which they call sequence level training, for training models to directly maximize a test-time evaluation metric such as BLEU or ROUGE [7]. The researchers demonstrated sequence level training significantly improves performance on the chosen metric, and claim that the method yields better generated text. The researchers assert that, because sequence level training evaluates text as a collective set of words rather than discrete choices, it is better able to guide the training of the model. This hypothesis raises the question that if certain evaluation metrics enable a model to produce more realistic text, then what is the best evaluation metric to use?

In this research, we hypothesize that training a model to maximize a sequence level evaluation metric learned directly from the training data can improve the quality of text generated by that model. We test this hypothesis by adapting generative adversarial networks (GANs) [6] for use with discrete-valued outputs using policy gradient methods [13], so that the discriminative component of the model may be employed as a learned evaluation metric. By applying this model to a toy dataset, we

demonstrate both the basic viability of the proposed model, as well as explore certain aspects of its training and use.

In order to more realistically evaluate the performance of GANs, we construct a state-of-the-art attention-aided recurrent neural network model, and train both the RNN and GAN to perform a conversational modeling task. RNN models have shown significant success on the task of conversational modeling, but they often fail to express consistent content or style [11]. Our findings indicate that GANs trained with simple variations of policy gradient methods do not outperform the RNN model, but that the discriminator of the GAN shows promise as a sequence level training metric.

2 Background and Related Work

2.1 Text Generation with Language Models

The most basic form of text generation is the language model, wherein one models the probability of each word in a vocabulary given a context of other words. The most successful current language models rely on the Recurrent Neural Network [3], a powerful model that can process arbitrarily long inputs by maintaining a hidden state that incorporates information over time. These models have been deployed on a wide variety of different types of text, and have proven very capable at modeling language. The basic RNN is not ideal for generating long sequences of meaningful text, as it produces text word by word relying on its relatively limited hidden state to maintain consistency between generated outputs.

2.2 Sequence to Sequence Models

In order to improve the performance of the language model when there is a clear input-output structure of the text, researchers developed the sequence to sequence model [10]. This model inputs and processes a sequence of words using an "encoder" RNN, and then uses a "decoder" RNN to translate the hidden state into an output sequence. These models have been widely deployed for machine translation [2], where they have achieved state-of-the art performance. For many applications of text generation these models are more applicable than a standard language model, as the generated text often has a purpose that can be formulated as an input text sequence.

A useful tool for sequence to sequence models is the application of an attention decoder, also introduced by [2]. This mechanism learns to selectively sample the hidden states of the encoder in order to extract important information from the input text when producing the output text. This method has been shown to have a positive effect on model accuracy when expanding to longer input-output sequences.

2.3 Generative Adversarial Networks

A GAN consists of a generative model G and discriminative model D, which can be viewed as playing a minimax game in which G tries to generate samples that D is unable to differentiate from real ones [6]. The solution to the game has G recovering the generating distribution and D randomly guessing between samples.

D is trained with a binary cross entropy loss and G is trained to maximize $log(D(G(z)))$, where z is noise randomly sampled from a prior. For learning with discrete-output samples, a recurrent GAN can be trained using policy gradient algorithms, which directly parameterize a policy and update it in accordance with the gradient of the cost with respect to those parameters[13] [7]. By conditioning on certain information, for example previous statements in a conversation, the GAN can be used to generate output relevant to that context.

2.4 Sequence-Level Training with Reinforcement Learning

It is not possible to train a GAN producing discrete-valued output with backpropagation. This is because one must sample at each timestep from the model in order to generate a word, and this is not a differentiable procedure. To overcome this issue, one can use policy gradient methods such as REINFORCE. Ranzato et al. use this approach with a new training method called sequence level

training, which uses REINFORCE to allow for end-to-end training of generative models [7]. In this method, the loss is taken to be the negative expected reward (e.g., the BLEU or ROUGE score for an output):

$$L_\theta = -\sum_{w_1^g,\ldots,w_T^g} p(w_1^g, ..., w_T^g) r(w_1^g, ..., w_T^g)$$

The derivative of this loss at a single timestep with respect to the output of the network o_t is then the following:

$$\partial L_\theta / \partial o_t = (r(w_1^g, ..., w_T^g) - \bar{r}_{t+1}) * (\hat{p}_t - 1\{w_{t+1}^g\})$$

Where $\bar{r}_{t+1}$ is a baseline subtracted from the actual reward to reduce the variance of the update, $\hat{p}_t$ is the softmax over output scores of the model, and $1\{w_{t+1}^g\}$ is an indicator vector with a one in the position of the word produced by the generator. As Ranzato et al. point out, this update has a nice interpretation in comparison with the derivative of the cross entropy loss: in the cross entropy loss we subtract 1 from the probabilities in the position of the *target word*. In the policy gradient case, we no longer have a target word, so we instead use the *generated word* as a proxy, and scale or invert the update proportionally to the reward (i.e., whether or not we want more of that action in the given context).

The primary disadvantage of this approach is that the action space of the agent is taken to be the vocabulary dimension, which grows very large for realistic datasets. To overcome this challenge, Ranzato et al. introduced MIXER, which performs hybrid training using both cross entropy and policy gradients, so as to direct the actions of the agent even within such a high dimensional space.

3 Approach

3.1 Recurrent Generative Adversarial Networks

We adapt sequence level training to GANs by replacing the reward metric (e.g., BLEU or ROUGE) with a discriminator network. In our recurrent GAN formulation, we pass in a single random sample z, which is converted to the initial hidden state of the network, and then forward propagate, sampling a word at each timestep. The sequence of output words is then provided to a discriminator network, which can either be used to produce a single score for the entire sequence or a score for each timestep of the sequence. We experimented with both variations and found the latter to speed learning. See figure 1 for a depiction of the network.

We implemented the baseline using a linear regression model in which the input was the generator hidden state and the output was the predicted reward from the discriminator. So long as this regression model minimizes squared error, it does not bias the gradient. We additionally approximated the MIXER hybrid training method by alternating between training the generator with an epoch of cross entropy and policy gradient methods. The benefits of these different components are considered in the results section.

3.2 Attention-Aided GRU Recurrent Neural Network

Our baseline state-of-the-art model for sequence to sequence modeling is a three layer, GRU-based, attention decoding RNN. This model recreates the sequence to sequence machine translation model implemented in [2]. As we are experimenting with same language input-output pairs, we modified the model to use a single shared vocabulary for the encoder and decoder. The decoder uses an attention mechanism, which creates a context vector by learning to sample the encoder hidden states such that it will extract the information from the encoder most important for generation of the current word. Word embeddings were randomly initialized in our model, as our experiments were on a large dataset so pretraining word embeddings was not necessary. We also added temperature sampling for text generation, wherein words are generated by a probabilistic sampling of the output distribution

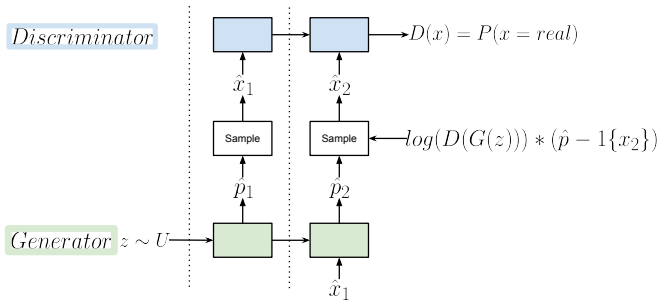

Figure 1: The recurrent generative adversarial network model. The derivative of the reward rather than the loss is shown for the gradient ascent case.

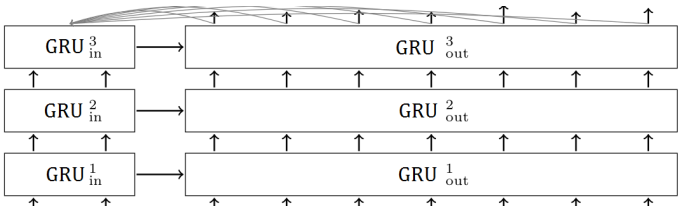

Figure 2: Three Layer GRU RNN Encoder/Decoder with Attention

produced by the decoder. This allows generated text to take on more variety and style than simply choosing the most likely word at each state.

We elected to use the Gated Recurrent Unit [4], as it has been shown to have far better performance on long sequences than the vanilla RNN. This model uses "gates" to select the relative importance of each value of the previous hidden state versus the input at each timestep. The GRU expands on the hidden state update of the vanilla RNN as follows:

$$
\begin{aligned}
z^{(t)} &= \sigma(W_z x^{(t)} + U_z h^{(t-1)}) \\
r^{(t)} &= \sigma(W_r x^{(t)} + U_r h^{(t-1)}) \\
\widetilde{h}^{(t)} &= tanh(W x^{(t)} + r^{(t)} \circ U h^{(t-1)}) \\
h^{(t)} &= z^{(t)} \circ h^{(t-1)} + (1 - z^{(t)}) \circ \widetilde{h}^{(t)}
\end{aligned}
$$

The decoder output is evaluated with a cross entropy loss, which compares the projected probability of the correct word in each location with the actual correct word. This loss is then back-propagated through the network to train both the weights of the model and the word embeddings.

4 Experiments and Results

We performed two sets of experiments. In the first set, we considered a toy example in order to demonstrate the viability of training recurrent GANs with policy gradient methods as well as to explore certain properties of the model and its training. In the second set of experiments, we evaluate the performance of the recurrent GAN and the attention-aided GRU network on a larger-scale conversational modeling dataset.

4.1 Toy Dataset Example

In order to ensure that the GAN trained with REINFORCE worked correctly, as well as to explore some of the design options and characteristics of the model, we created a toy dataset consisting of the letters of the alphabet in order. In experiments on this dataset, the model was trained to generate two characters, and evaluation of the model was performed by computing the ratio of generated samples

in the dataset to those not in the dataset. For example, if the model randomly generates $[a, b]$ that would be in the dataset, whereas if it generates $[b, a]$ that would not be. While this experiment neither reflects the ability of the model to generalize to unseen data nor to learn more complex distributions, it does make understanding and analyzing the model easier.

4.1.1 Training Method Comparison

Figure 3 shows the loss over time when training the model with different methods. In all cases, the model is trained for 1000 epochs, without dropout, a per-timestep sampling temperature of 1.0 that is annealed to 0.01, 128 hidden units, an embedding dimension of 32, and LSTM hidden units. We explored three training variants. The first is simple REINFORCE as described in the background section. The second incorporates a baseline [12] in order to reduce the variance of the parameter updates, which is computed by training a linear classifier to predict the reward of the discriminator given the hidden state of the generator [7]. The baseline is generally subtracted from the reward term in the REINFORCE gradient expression. In order to accomplish this in our model implemented in TensorFlow [1], we subtracted the baseline from the reward computed in the loss, which is reflected in the fact that the loss of the generator can be negative. Figure 4 shows the loss over time of the baseline regression model. The third variation we considered was an approximation of the MIXER algorithm [7], where we alternate training the generator with cross entropy (as a traditional RNN sequence model) and policy gradient methods. Table 1 shows the evaluation results for the different training procedures.

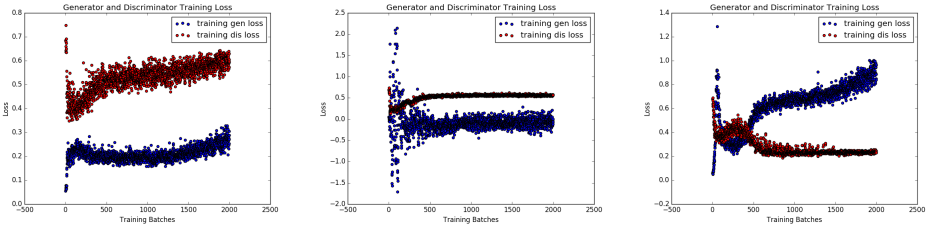

Figure 3: Training loss for generative and discriminative models using policy gradients (left), policy gradients with a baseline (center), and hybrid policy gradient and cross entropy (right).

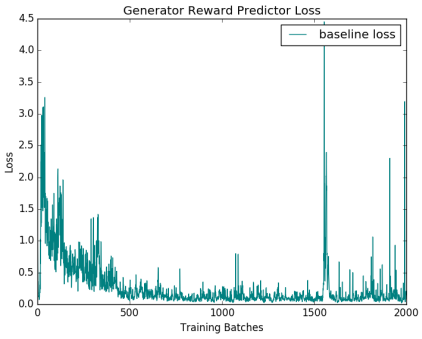

Figure 4: Training loss of the baseline reward predictor over time.

4.1.2 Results

Model	In-Sample Percentage
Policy Gradient	96.1%
Policy Gradient with Baseline	86.5%
Policy Gradient with Cross Entropy	93.2%

Table 1: In-sample results for toy dataset

The loss graphs for GANs can be difficult to interpret because of the competing nature of the model subcomponents. For example, it is frequently the case that model performance can improve drastically with only small changes to the loss values because the losses express the relative, not absolute, performance of the generator and discriminator. This relationship can be controlled somewhat using a hyperparameter k, which is the number of training epoch runs for the discriminator for each epoch of the generator. In this toy experiment, we trained both for only one epoch, which contributed to the discriminator having greater loss. Furthermore, because our implementation of the GAN alternates between training the subcomponents for complete epochs, the training losses are not complementary. Nevertheless, they do provide some insight. The GAN trained with only REINFORCE shows that the relative loss of both models stays approximately the same over time. Anecdotally, this seems to be a strong indicator of successful training, as is reflected in the model's high in-sample percentage.

The models trained using a baseline and with alternating cross entropy perform worse than the basic model. This may be due to the fact that the hyperparameters (e.g., learning rate, k, hidden units, embedding dimension, z dimension) were tuned for the basic model. In the cross entropy case, the generator tends to learn quickly that a certain pair of characters is common, and then produces those characters with high frequency. This trend results in the discriminator easily recognizing the outputs of the generator as fake, which is reflected in the large and increasing loss of the generator over time. This same result occurs in the larger action space of the Twitch.tv dataset as well.

4.2 Twitch.tv Experiments

4.2.1 Dataset

In order to test the relative capabilities of the RNN and GAN models on a large scale sequence generation task, we trained the models to emulate chatting agents on the popular live-streaming website Twitch.tv. This website hosts "channels" on which a person live-streams themselves playing video games. Anyone watching this channel has the opportunity to participate in an IRC chatroom associated with the channel. Large channels produce millions of lines of chat each year, creating a sufficient dataset to train a sequence to sequence conversational agent.

Our problem is formulated as a sequence to sequence text generation problem where the model is given the previous three messages sent in chat as context and is asked to generate the subsequent chat message. This is a similar setup to the conversational models developed in [11].

Most conversational models are trained on the logs of chatrooms with two people, and as such they model the interaction of two people in a coherent, flowing conversation. Twitch chats often have upwards of 10,000 participants, so the interactions tend to be less conversational and more "group think". In the larger rooms, attempting to type meaningful sentences is often useless, as the chat message is not on screen for long enough for it to be read. As such, most messages are short jokes, or emoticons. Smaller channels often have more conversational chats, as chat agents are more likely to recognize each other and read messages.

4.2.2 Training

We trained two separate RNN models on the logs of two Twitch channels, one small channel with roughly 4,000 participants, and one large channel with upwards of 20,000 participants. Training was done with the AdaGrad optimizer [5], and learning rate was decayed when the training loss plateaued. Each model was shown roughly 3.5 million training examples from the Twitch chat dataset.

The GAN was trained on the logs of only the small channel, using the identical hyperparameters as those described above in the training of the toy dataset, though with larger networks.

4.2.3 Results

The state of the art model clearly outperforms the GAN in this sequence generation task. The state of the art model achieved very low test loss, which is expected on this relatively simple task with a very large dataset.

The GAN failed to reach this level of performance on the test dataset. Since the GAN uses reinforcement learning in training, it takes an action at each generation step (i.e., samples a word) and

Model	Test Perplexity
GAN with Policy Gradient on Small Channel	8560
GAN with Policy Gradient and Baseline on Small Channel	1.60×10^{18}
GAN with Policy Gradient and Cross Entropy on Small Channel	∞
RNN on Small Channel	24.8
RNN on Large Channel	17.8

Table 2: Perplexity Results on Twitch Dataset

the size of this action-space is the size of the vocabulary. This leads to both slow and sparse learning when using REINFORCE both with and without a baseline.

The performance of the different GAN models provides insight into the training variants and their impact on the models. The simple GAN with policy gradient achieves slightly above random performance (as the vocabulary size in testing was 10,000). The GAN with baseline begins to make actual predictions, for instance predicting a single, frequently seen word for all words. This leads to much higher perplexity, as it then assigns near zero probabilities to all other words. The GAN trained with hybrid cross entropy loss does this to an even greater extent, assigning zero probability to most words and very high probability to a few much more common words, which leads to infinite perplexity on the test set.

4.2.4 Analysis

The two RNN twitch chat agents behave very differently. The bot trained on the large channel either says short joking fragments or parrots input, similar to the "group think" described above. The bot trained on the small channel will also parrot some input, like emoticons, but will also put together longer sentences when prompted with text. These sentences are often not relevant to the input prompt, as chat is mostly directed at the streamer instead of other chatters in the channel. Examples of the outputs of each channel are shown below in figures 5 and 6.

Figure 5: Example Output of Twitch Chat Large Channel RNN

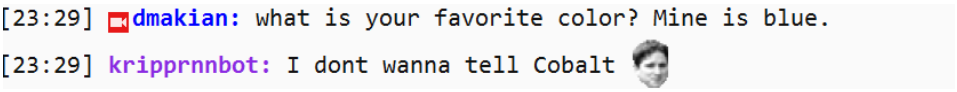

Figure 6: Example Output of Twitch Chat Small Channel RNN

Validation loss for both channels is shown in figure 7. It was easier for the RNN to learn the behaviour of the large channel, which is expected since the large channel tends to consist of shorter, less coherent messages. The larger channel both sees and has to predict longer sentences that are more reliant on previous chat messages, making it a significantly harder task.

4.2.5 t-SNE Visualization of Word Embeddings

We have visualized the word embeddings learned by the RNN model using t-SNE. This technique reduces the 128-dimensional embeddings into two representative dimensions, so they can be more easily visualized. Words close together in this space are seen frequently in similar contexts or have similar meanings. The t-SNE visualization is shown in figure 8.

There is a lot of interesting structure in this visualization of the 100 most common words used by the large model. Notably, the emoticons are divided into two distinct groups. The bottom group is the set of "funny" or "happy" emoticons. The leftmost cluster is a group of "sad" emoticons. The rightmost cluster is a set of unicode characters frequently used to make text-drawings in chat.

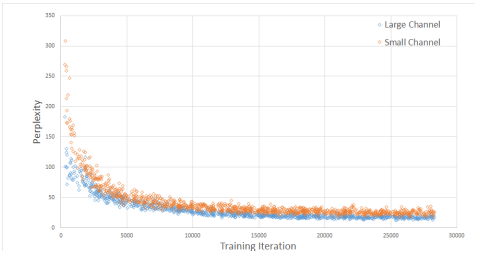

Figure 7: Validation Perplexity while Training RNN Model

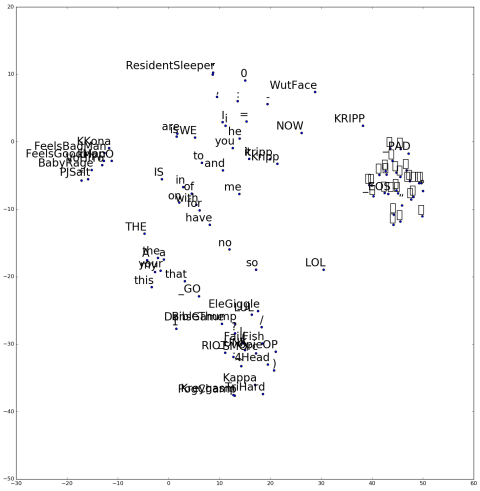

Figure 8: t-SNE Visualization of Word Embeddings Learned by RNN Model

5 Conclusions and Future Work

The Recurrent Generative Adversarial Network fails to scale to large action-spaces, even when aided by simple hybrid learning techniques. Applying reinforcement learning in vocabulary-sized action spaces poses a significant challenge when initializing with a random policy because learning proceeds too slowly, or, depending on sampling temperature, can fixate on a small set of words resulting in insufficient exploration. These challenges lead to significantly lower accuracy on large sequence generation task than the state of the art multi-layer RNN models.

Future work could explore training a GAN with MIXER [7], which would likely address training issues more effective than the naive hybrid method we used. Other training methods that allow the model to learn rewarding states more quickly could also significantly improve the model.

With a more successful model, future work could be done to compare the trained discriminator network to current text evaluation metrics like BLEU or ROUGE. In theory the discriminator should be similarly capable of grading generated text, and comparing its properties to those of designed metrics could lend insight into GAN performance as well as allow for further improving generative models.

References

[1] Martın Abadi et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”. In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *CoRR* abs/1409.0473 (2014). URL: http://arxiv.org/abs/1409.0473.

[3] Yoshua Bengio et al. “A Neural Probabilistic Language Model”. In: *J. Mach. Learn. Res.* 3 (Mar. 2003), pp. 1137–1155. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=944919.944966.

[4] Junyoung Chung et al. “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling”. In: *CoRR* abs/1412.3555 (2014). URL: http://arxiv.org/abs/1412.3555.

[5] John Duchi, Elad Hazan, and Yoram Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. Tech. rep. UCB/EECS-2010-24. EECS Department, University of California, Berkeley, Mar. 2010. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-24.html.

[6] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.

[7] Sumit Chopra MarcAurelio Ranzato, Michael Auli, and Wojciech Zaremba. “Sequence level training with recurrent neural networks”. In: *Proceedings of ICLR 2016* (2016).

[8] Ani Nenkova, Sameer Maskey, and Yang Liu. “Automatic summarization”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts of ACL 2011*. Association for Computational Linguistics. 2011, p. 3.

[9] Kishore Papineni et al. “BLEU: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318.

[10] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *CoRR* abs/1409.3215 (2014). URL: http://arxiv.org/abs/1409.3215.

[11] Oriol Vinyals and Quoc V. Le. “A Neural Conversational Model”. In: *CoRR* abs/1506.05869 (2015). URL: http://arxiv.org/abs/1506.05869.

[12] Lex Weaver and Nigel Tao. “The optimal reward baseline for gradient-based reinforcement learning”. In: *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2001, pp. 538–545.

[13] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4 (1992), pp. 229–256.