Adversarial Discrete Sequence Generation

Ankit Vani
New York University
ankit.vani@nyu.edu

Abstract

Generative adversarial networks (GAN) have been successful in modeling high-dimensional continuous spaces. These models minimize the some distance between a target distribution and a generator distribution, which can often produce more perceptually realistic generated samples as compared to training a generative model using maximum likelihood estimation (MLE). In this work, we present an approach to use adversarial objectives to learn discrete generative models for sequences. Continuous adversarial generative models are trained by backpropagating gradients from a discriminator to the generator, which is not possible if the generator output is discrete, as is in case of words in language. We propose using reinforcement learning to train a generator using costs assigned by a discriminator. We propose an actor-critic framework that receives signal from a discriminator to train a discrete generative model from scratch without MLE pretraining. We explore ways to stabilize training of such a model and report results on toy tasks, generation of single words and generation of sentences.

1 Introduction

Generative adversarial networks (GAN) [Goodfellow et al., 2014] have found success in modeling high-dimensional continuous spaces such as images [Radford et al., 2015, Denton et al., 2015, Reed et al., 2016]. Maximum likelihood estimation (MLE) minimizes the KL divergence $KL(P_{\mathbf{X}}||P_\theta)$, where $P_{\mathbf{X}}$ is the data distribution and P_θ is the model distribution parameterized by θ. Since KL divergence is not symmetrical between $P_{\mathbf{X}}$ and P_θ, we end up penalizing $P_\theta(\mathbf{x}) < P_{\mathbf{X}}(\mathbf{x})$ more than $P_{\mathbf{X}}(\mathbf{x}) < P_\theta(\mathbf{x})$. Essentially, because of this, the generative model tries to cover the entire data distribution $P_{\mathbf{X}}$ even at the cost of including regions that do not overlap with $P_{\mathbf{X}}$. GANs try to overcome this issue by minimizing symmetrical divergences such as the Jensen-Shannon divergence [Goodfellow et al., 2014], total variation distance [Zhao et al., 2016, Arjovsky et al., 2017], or the Wasserstein distance [Arjovsky et al., 2017].

While many variants of GANs for images have come up, the applications of adversarial generative models have been less prominent in natural language processing. Although MLE works much better for perceptually reasonable generations for text than it does for images, modeling the real world through language and learning linguistic subtleties remain open problems. It is possible that we might be able to tackle some of these issues by using more optimal loss functions, perhaps minimizing a distance measure other than KL divergence between the data and model distributions. In this work, we try to formulate a framework for generative adversarial discrete sequence generation, which readily applies to text generation since characters or words are discrete entities.

The generator parameters of a GAN are updated to produce a sample that moves the discriminator output to a desired value, such as 1 for the original GAN, or a specific value for Least Squares GAN [Mao et al., 2016], above a margin for Energy-Based GAN [Zhao et al., 2016], or ∞ for Wasserstein GANs (WGAN) (under the Lipschitz constraint) [Arjovsky et al., 2017]. This is generally done using the gradients of the loss on the discriminator value with respect to the generator output.

However, if the generator output is discrete, we cannot obtain these gradients[1] and it is not possible to backpropagate from the discriminator to the generator. Thus, we use policy gradients to train the generator based on a discriminator that can identify real from fake discrete samples.

We propose a framework for generative adversarial sequence generation that uses two recurrent neural networks (RNN) using the Gated Recurrent Units (GRU) [Cho et al., 2014]. At any given timestep, the generator RNN tries to produce a token that can fool the discriminator into believing that it is drawn from the data distribution conditioned on the previously generated tokens.

2 Related Work

Yu et al. [2017] proposed SeqGAN, a GAN where the generator is learned using policy gradients. The discriminator D for their model is trained to produce a single value (1 for 'real' and 0 for 'fake') based on an entire sequence. The generator is then trained using the REINFORCE algorithm, considering $D(\mathbf{x}_{1:T})$ as the reward, where $\mathbf{x}_{1:T}$ is the generated sequence. This implies doing a Monte Carlo policy evaluation by completing the sentence from a given timestep, repeated for each timestep. This is a very difficult search problem, since the generator has to keep producing entire sequences and the feedback from the discriminator is only given at the global level. Thus, the authors found it useful to pretrain the model using MLE, and finetune the resulting model with their adversarial approach. Unlike SeqGAN, we chose not to do Monte Carlo policy evaluation to estimate the action-value function at a timestep, but instead learn a Q-network at each timestep with parameters shared across time. We hope that this gives the model higher quality signal to improve generations, eliminating the need for MLE pretraining.

One of the ways to perform imitation learning involves inverse reinforcement learning (IRL) to find a cost function that an expert is behaving optimally under (any other policy would have higher expected cost under that cost function), and reinforcement learning (RL) to learn a policy that minimizes cost under that function. Ho and Ermon [2016] present an adversarial approach to imitation learning, where they discuss different GAN formulations depending on the regularization applied on the reward function during IRL. A regularization is derived under which the $RL \circ IRL$ objective behaves like a standard GAN. Hjelm et al. [2017] present a general GAN formulation which readily applies to discrete distributions. They show that moving the generator generations towards the discriminator boundary results in stable learning. Although they do not talk about it in the paper, their discrete objective uses policy gradients with the action-value function proportional to $D(\mathbf{x})/(1 - D(\mathbf{x}))$, where D is the output of the discriminator. In our work, we use the WGAN [Arjovsky et al., 2017] formulation, building upon what turns out to be a special case of the general framework presented by Ho and Ermon [2016].

Gulrajani et al. [2017] improved upon the original WGAN work by presenting a gradient penalty term to enforce the Lipschitz constraint in the WGAN objective instead of weight clipping. They show that with this modification, they are able to use the WGAN to learn a language model. However, they learn this language model in continuous space, by considering the one-hot vectors of characters in text as continuous values to model. Rajeswar et al. [2017] build upon this idea and use the improved WGAN on recurrent networks to get higher quality generations such as for Chinese poetry.

Bahdanau et al. [2016] presented an actor-critic framework to train recurrent neural networks. Here, the critic is trained to predict the value of an output token given the policy of an actor. This lets the actor train conditioned on its previous guessed tokens, which is much closer to the setting during testing. We use a similar actor-critic framework to train the discrete sequence generator, which uses costs from a discriminator to guide its training.

More recently, Press et al. [2017] trained discrete GANs for language generation using recurrent neural networks without MLE pretraining. Although we also tackle the problem of generating text without MLE pretraining, the tricks used to get the model to converge are different. Press et al. [2017] use curriculum learning based on sentence length to aid learning. They also use teacher helping, where the generator is conditioned on shorter ground truth sequences and an adversarial loss in incurred on the subsequently generated tokens. They show sentence generations with valid English words.

[1]For example, it is not possible to move a generated word some δ amount in a direction to change the discriminator loss.

3 Background

In this section, we briefly discuss the background needed for deriving our sequence generation model.

3.1 Generative Adversarial Networks

GANs learn a generative model by playing a minimax game between a generator G and a discriminator D. The discriminator is trained to classify input samples as coming from the data distribution ('real') or coming from the generator ('fake'). The generator, on the other hand, is trained to fool the discriminator by producing samples that are indistinguishable from real for the discriminator.

The GAN objective can thus be formulated as

$$\min_{\theta} \max_{\phi} \mathop{\mathbb{E}}_{\mathbf{x} \sim P_{\mathbf{X}}} [\log D_{\phi}(\mathbf{x})] + \mathop{\mathbb{E}}_{\hat{\mathbf{x}} \sim P_{\theta}} [\log(1 - D_{\phi}(\hat{\mathbf{x}}))] \tag{1}$$

Here, $\hat{\mathbf{x}} \sim P_{\theta}$ is often accomplished by the generator taking as input a noise vector z, coming from a prior distribution, and mapping it to a point in the data space. That is, $\hat{\mathbf{x}} = G_{\theta}(\mathbf{z})$ where $\mathbf{z} \sim P_{\mathbf{z}}$. The prior noise distribution $P_{\mathbf{z}}$ usually has zero mean and is a Gaussian or a uniform distribution.

The signal for the generator to update its generative model comes from the discriminator itself, by using gradients of the discriminator loss with respect to the generated image. This GAN formulation minimizes the Jensen-Shannon divergence between the generator distribution and the data distribution. It has been shown that as the discriminator gets better, the gradient of the generator vanishes [Arjovsky and Bottou, 2017].

In practice, however, gradients from the discriminator are often reversed and backpropagated to the generator, thereby asking the generator to maximize the discriminator's loss. This variant, although avoiding vanishing gradients, suffers from unstable behavior. Arjovsky and Bottou [2017] showed that this approach minimizes $KL(P_{\theta}||P_{\mathbf{X}}) - 2JSD(P_{\theta}||P_{\mathbf{X}})$, where minimizing the KL divergence term encourages mode dropping ($P_{\mathbf{X}}(\mathbf{x}) < P_{\theta}(\mathbf{x})$ is penalized more than $P_{\theta}(\mathbf{x}) < P_{\mathbf{X}}(\mathbf{x})$). Furthermore, they showed that under an imperfect discriminator, the generator gradient updates get massively unstable.

3.2 Wasserstein GAN

WGANs minimize the Wasserstein distance between the data distribution and the generator distribution. This formulation provides a useful generator gradient everywhere and the quality of the gradient updates improve as the discriminator (also known as the critic) reaches optimality, unlike the original GAN. Minimizing the Wasserstein distance constitutes of moving the generator probability mass to the data distribution, where both false positives and false negatives are identically penalized. Thus, the model also does not suffer from mode dropping.

The Wasserstein distance W can be computed by optimizing

$$W(P_{\mathbf{X}}, P_{\theta}) = \sup_{\|f\|_L \leq 1} \mathop{\mathbb{E}}_{\mathbf{x} \sim P_{\mathbf{X}}} [f(\mathbf{x})] - \mathop{\mathbb{E}}_{\hat{\mathbf{x}} \sim P_{\theta}} [f(\hat{\mathbf{x}})] \tag{2}$$

The WGAN objective can then simply be written as

$$\min_{\theta} W(P_{\mathbf{X}}, P_{\theta}) \tag{3}$$

where θ parameterizes the generative model. The function f is implemented by a critic, which is updated under the Lipschitz constraint.

In the original work, WGAN proposed clipping the model weights to enforce the Lipschitz constraint. However, this limits the expressiveness of the model, preventing it from taking full advantage of its architecture. Gulrajani et al. [2017] fixed this limitation in a soft way by adding a gradient penalty term to the WGAN objective. Thus, the improved WGAN objective can be written as

$$\min_{\theta} \max_{\phi} \mathop{\mathbb{E}}_{\mathbf{x} \sim P_{\mathbf{X}}} [f_{\phi}(\mathbf{x})] - \mathop{\mathbb{E}}_{\hat{\mathbf{x}} \sim P_{\theta}} [f_{\phi}(\hat{\mathbf{x}})] - \lambda \mathop{\mathbb{E}}_{\tilde{\mathbf{x}} \sim P_{\tilde{\mathbf{x}}}} \left[(\|\nabla_{\tilde{\mathbf{x}}} f_{\phi}(\tilde{\mathbf{x}})\|_2 - 1)^2 \right] \tag{4}$$

where λ is a hyperparameter and $\tilde{\mathbf{x}} \sim P_{\tilde{\mathbf{x}}}$ is short for $\epsilon \sim U[0,1], \mathbf{x} \sim P_{\mathbf{X}}, \hat{\mathbf{x}} \sim P_{\theta'}$ and setting $\tilde{\mathbf{x}} = \epsilon\mathbf{x} + (1-\epsilon)\hat{\mathbf{x}}$. Here, $P_{\theta'}$ is a clone of P_{θ}, indicating that the generator parameters are not updated when optimizing the gradient penalty term.

WGANs are more stable to train than standard GANs. In our work, we use the principles of WGAN to build a GAN for discrete sequences.

3.3 Imitation Learning

Imitation learning refers to learning a policy that mimics the behavior of an expert. There are two common strategies for approaching imitation learning: behavioral cloning and inverse reinforcement learning. Behavioral cloning considers a simple supervised learning problem with expert state action pairs. While simple, it only performs well if we have large amounts of data. This is because prediction errors from individual state action pairs can start compounding over time to magnify the faults in the learned policy.

Inverse reinforcement learning (IRL) learns a reward function under which an expert is behaving optimally under. In other words, it learns a cost function that an expert minimizes. Then, imitation learning consists in learning a policy through reinforcement learning under this recovered cost function. This school of thought sees imitation learning as $RL \circ IRL$.

In this work, we consider the data as generated by an expert, and perform imitation learning to build a discrete sequence generative model.

3.4 Generative Adversarial Imitation Learning

The result of IRL is a cost function c, which Ho and Ermon [2016] define as

$$IRL_{\psi}(\pi_E) = \underset{c \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}}{\arg\max} \; -\psi(c) + \left(\min_{\pi\in\Pi} -H(\pi) + \mathop{\mathbb{E}}_{\pi}\left[c(s,a)\right] \right) - \mathop{\mathbb{E}}_{\pi_E}\left[c(s,a)\right] \tag{5}$$

Here, π_E refers to the expert policy, from which we can obtain trajectories from for learning. $\mathcal{S}$ is the set of states and $\mathcal{A}$ the set of actions. $\psi : \mathbb{R}^{\mathcal{S}\times\mathcal{A}} \to \mathbb{R}$ is a regularization function for c, which is necessary to prevent overfitting given a finite number of expert trajectories.

The occupancy measure is the distribution of state-action pairs that an agent encounters under a policy π. For an occupancy measure ρ, Syed et al. [2008] showed that there is a unique policy π_ρ corresponding to it. Note that $\mathbb{E}_\pi\left[f(s,a)\right] = \sum_{s,a} \rho_\pi(s,a) f(s,a)$ for a function f and policy π. Ho and Ermon [2016] derived the imitation learning framework

$$RL \circ IRL_{\psi}(\pi_E) = \underset{\pi\in\Pi}{\arg\min} \; -\lambda H(\pi) + \psi^*(\rho_\pi - \rho_{\pi_E}) \tag{6}$$

where ψ^* is the convex conjugate of ψ. Here, a hyperparameter λ is added to control the entropy regularization of the learned policy.

They propose the following regularizer

$$\psi_{GA}(c) = \begin{cases} \mathbb{E}_{\pi_E}\left[g(c(s,a))\right] & \text{if } c < 0 \\ \infty & \text{otherwise} \end{cases} \quad \text{where } g(x) = \begin{cases} -x - \log(1-e^x) & \text{if } x < 0 \\ \infty & \text{otherwise} \end{cases} \tag{7}$$

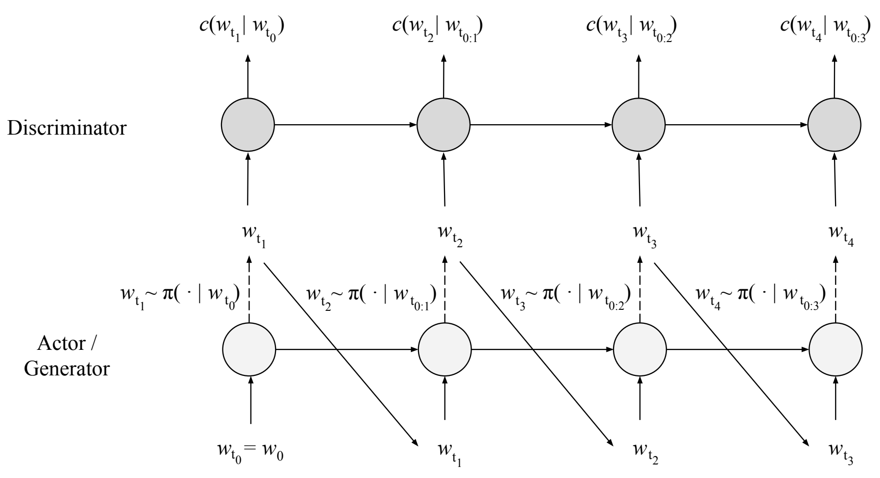
and show that

$$\psi^*_{GA}(\rho_\pi - \rho_{\pi_E}) = \max_{D\in(0,1)^{\mathcal{S}\times\mathcal{A}}} \mathop{\mathbb{E}}_{\pi}\left[\log D(s,a)\right] + \mathop{\mathbb{E}}_{\pi_E}\left[\log(1-D(s,a))\right] \tag{8}$$

Using this result in Equation 6, we see that the imitation learning objective becomes similar to the standard GAN objective. Thus, with this cost regularizer, a policy π is learned that minimizes $JSD(\rho_\pi || \rho_{\pi_E}) - \lambda H(\pi)$.

4 Discrete Sequence Generation

We now present our discrete sequence generation model. First, we establish the connections with the work of Ho and Ermon [2016] and Arjovsky et al. [2017], and then present novel techniques for stable and faster training of GANs with policy gradient.



$c(w_{t_1} | w_{t_0})$
$c(w_{t_2} | w_{t_{0:1}})$
$c(w_{t_3} | w_{t_{0:2}})$
$c(w_{t_4} | w_{t_{0:3}})$
Discriminator
w_{t_1}
w_{t_2}
w_{t_3}
w_{t_4}
$w_{t_1} \sim \pi(\cdot | w_{t_0})$
$w_{t_2} \sim \pi(\cdot | w_{t_{0:1}})$
$w_{t_3} \sim \pi(\cdot | w_{t_{0:2}})$
$w_{t_4} \sim \pi(\cdot | w_{t_{0:3}})$
Actor / Generator
$w_{t_0} = w_0$
w_{t_1}
w_{t_2}
w_{t_3}

Figure 1: The discrete sequence GAN architecture with a sequence length of 4.

4.1 Generative Adversarial Apprenticeship Learning

Syed and Schapire [2007] presented a game-theoretic formulation of apprenticeship learning, where the algorithm can actually find a policy that performs better than the expert across a class of cost functions $\mathcal{C} \subseteq \mathbb{R}^{\mathcal{S} \times \mathcal{A}}$. This is done by optimizing the objective

$$\min_{\pi} \max_{c \in \mathcal{C}} \mathop{\mathbb{E}}_{\pi} [c(s,a)] - \mathop{\mathbb{E}}_{\pi_E} [c(s,a)] \tag{9}$$

As Ho and Ermon [2016] pointed out, without entropy regularization, this is equivalent to Equation 6 with

$$\psi(c) = \begin{cases} 0 & \text{if } c \in \mathcal{C} \\ \infty & \text{otherwise} \end{cases} \tag{10}$$

Previous work on apprenticeship learning have considered $\mathcal{C}$ to be the set of linear [Abbeel and Ng, 2004] or convex [Syed et al., 2008] functions. However, these classes are generally too restrictive to accurately capture the dynamics of a real-world system.

By setting $\mathcal{C}$ to be the set of all 1-Lipschitz functions, we overcome many of these limitations by being able to learn complex non-linear cost functions that are still constrained in interesting ways. For instance, although the objective encourages low costs for expert trajectories and high costs for trajectories sampled from the learned policy, these costs cannot abruptly change at some decision boundary. If the occupancy measures of the expert and the learned policies are indeed close, then their costs will be similar. It is easy to see that using this setting of $\mathcal{C}$, we recover the WGAN objective from Equation 3:

$$\min_{\pi} \max_{\|c\|_L \leq 1} \mathop{\mathbb{E}}_{\pi} [c(s,a)] - \mathop{\mathbb{E}}_{\pi_E} [c(s,a)] \tag{11}$$

The objective minimizes the Wasserstein distance between the occupancy measure of the expert policy π_E and that of the learned policy π. We use WGAN with gradient penalty to enforce the Lipschitz constraint, which requires interpolating real and generated samples to take the gradient with respect to. We do this for two sequences by interpolating their embeddings over each timestep by an amount fixed for each real-generated pair.

To optimize Equation 11, we train a discrete sequence GAN comprising of a recurrent discriminator and a recurrent generator, as illustrated in Figure 1. The generator models the policy learned π, and we want trajectories sampled from this recurrent generator to be similar to trajectories sampled from the expert policy π_E according to the discriminator.

More specifically, at each timestep i, the generator produces a categorical distribution over V possible tokens $w_{0:V}$, conditioned on the previously generated tokens $w_{t_{0:i-1}}$. The token at the ith timestep

w_{t_i} is then sampled from this conditional distribution. Simultaneously, a recurrent discriminator consumes the generated tokens to produce the scalar cost $c(s, a)$ at each timestep, conditioned on previously generated tokens. The discriminator and generator recurrent networks use GRUs, and the final generator distributions and the discriminator costs are affine transformations of the corresponding GRU hidden states. For both recurrent networks, the input token is used to look up an associated learned embedding for that token, before being fed into the GRU.

The generator in this model produces discrete outputs, and gradients from the discriminator are backpropagated to the generator through reinforcement learning. With this adversarial framework in place, we now explore suitable architectures for sequence generation and techniques that we found useful in training such models.

4.2 Advantage Actor-critic Framework

Actor-critic methods combine the advantages of value iteration and policy iteration algorithms. An actor network is trained to learn the policy, modeling the distribution over actions conditioned on a state. A critic is used to estimate the action-value function, which is the expected reward accumulated starting from an action at a state and then acting according to the same policy. The actor updates its parameters to improve the policy it represents towards higher rewards, by using the critic's estimates of the value of a state-action pair. The low variance estimates by the critic help in stable training of the actor, alleviating the usual problem of high variance gradient estimates in policy iteration methods.

Since we want to update π to minimize $\mathbb{E}_{\pi}\left[c(s, a)\right]$ as seen from Equation 11, we can have the critic estimate this value for any given policy, and then update the actor to minimize it. The actor is improved by updating its parameters in the direction of the policy gradient:

$$\mathop{\mathbb{E}}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q(s, a)\right] \tag{12}$$

where $Q(\bar{s}, \bar{a}) = -\mathbb{E}_{\pi}\left[c(s, a) \mid s_0 = \bar{s}, a_0 = \bar{a}\right]$. $Q(s, a)$ in the above expression can be replaced by the advantage function $A(s, a)$ without changing the expectation but possibly reducing the variance, making training more stable. The advantage function is defined as $A(s, a) = Q(s, a) - V(s)$, the advantage of choosing action a over any other action at state s. The critic is trained to be an accurate estimator of $V(s)$ by minimizing the $\mathbb{E}_{\pi}\left[A(s, a)^2\right]$.

4.3 Smoothed Absolute Costs

In the WGAN, the discriminator assigns costs in the range $(-\infty, \infty)$. The generator tries to produce samples that push costs $-\infty$, whereas the discriminator tries to push costs for generator's samples towards ∞ and real samples towards $-\infty$. With the inherent instability of adversarial learning and reinforcement learning, this makes it difficult for the discriminator to learn and provide useful signal to the generator. A simple trick is to ground the lowest value the discriminator can produce to zero, which makes learning significantly more stable. This does not change the apprenticeship learning formulation discussed previously.

To ground the lowest discriminator value to zero, we take a real valued discriminator output c and use $g_s(c)$ instead of c as the cost, where

$$g_s(x) = \begin{cases} |x| - s/2 & \text{if } |x| \geq s \\ x^2/2s & \text{if } |x| < s \end{cases}$$

This function is also known in machine learning literature as the Huber loss.

The discriminator tries to assign a value of zero to real samples, and the generator tries to produce samples that the discriminator would judge as having a value close to zero.

4.4 Real Cost Multiplier

The generator learns through the discriminator's idea of the true data manifold. We found it useful in our work to couple smoothed absolute costs with a higher weight for real samples when training the discriminator. That is, it is more important for the discriminator to produce costs closer to zero for real samples than it is for it to produce higher costs for generated samples. This further helps ground the costs for real samples and helps stabilize training.

The updated objective is thus

$$\min_{\pi} \max_{c} \mathop{\mathbb{E}}_{\pi} [g_s(c(s,a))] - \lambda \mathop{\mathbb{E}}_{\pi_E} [g_s(c(s,a))]$$

where we use $\lambda = 10$.

We observe better convergence with a real cost multiplier. We further notice that the discriminator gradient norm approaches zero near convergence, which helps the model to remain stable even after the task has been solved.

4.5 Experience Replay

Although experience replay has not been found useful in training continuous GANs, we found that it does in fact help in the setting where the generator is trained via policy gradient. In this setting, the generator and discriminator are less strongly coupled with each others' exact parameters. The generator's training is facilitated by the rewards (costs) assigned by the discriminator, and the discriminator does a better job at learning these rewards by using previously generated samples as fake samples.

4.6 Optimizing Each Possibility

When doing sequence learning with MLE, one maximizes the likelihood of w_t conditioned on $w_{1:t-1}$. One could imagine trying to optimize our problem similarly without the MLE objective by setting $\gamma = 0$. In this case, at each timestep the generator only gets signal through its immediate next action. In this formulation, it is possible to optimize for each action at every timestep by collecting costs for each action conditioned on the current state from the discriminator. We can use these costs along with the probability distribution over actions from the actor to compute the expectation in Equation 12 over all actions at a timestep without sampling. However, we then decide one action and the next timestep is optimized based on the action chosen. This allows lets the model use more signal per training step, especially for rarer actions, leading to faster and more stable convergence.

4.7 Other Details

To prevent the model from collapsing to a single action, it is important to use entropy regularization when training the generator. Entropy regularization also helps encourage exploration. We also find out useful to linearly decay entropy regularization to a minimum amount over a fixed number of training steps to help the model eventually converge sharply to the correct actions.

Additionally, the defaults of 5 discriminator iterations and 1 generator iteration per turn in the original WGAN work do not converge for our setting. We found that it is necessary to take a higher number of generator and discriminator steps at any turn depending on the task. However, it is important for the discriminator iterations to be higher than the generator iterations. We also have an initial burn-in period where we take 100 discriminator iterations per generator iteration for the first 50 or 100 steps.

5 Results

We evaluated our results on different tasks of varying difficulties. For all these tasks, the actor, critic and the discriminator networks have a single recurrent layer, trained using the Adam optimizer [Kingma and Ba, 2014] with learning rate 0.0001, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. We clipped gradients to a norm of 15.0 for stable RNN learning.

5.1 Toy Tasks

We evaluated our method on simple toy tasks, that the model should be able to solve fairly quickly to be able to tackle more challenging problems. The first toy problem involves producing a random integers uniformly sampled from a range at specific positions in a sequence, separated by zeros. We call this the positioning task, and it tests the model's capability to learn to remember positions across long ranges of time in a sequence. For the results reported here, we used a vocabulary size of 20 with embedding size 32 and recurrent hidden state size 128. We did not need to use a real cost multiplier for this task to converge. We used 10 most recent generators for experience replay.

Figure 2 shows how our model converges to a solution with and without the tricks mentioned in the previous sections. With smoothed absolute loss, real costs multiplier, experience replay and optimizing each possibility, we can consistently solve this task within 600 steps, which takes about two minutes on a modern GPU. We noticed that without bounding the costs from one side, the model has a hard time converging and shows a lot of variance during training. Table 1 shows real samples and samples generated by a converged model.

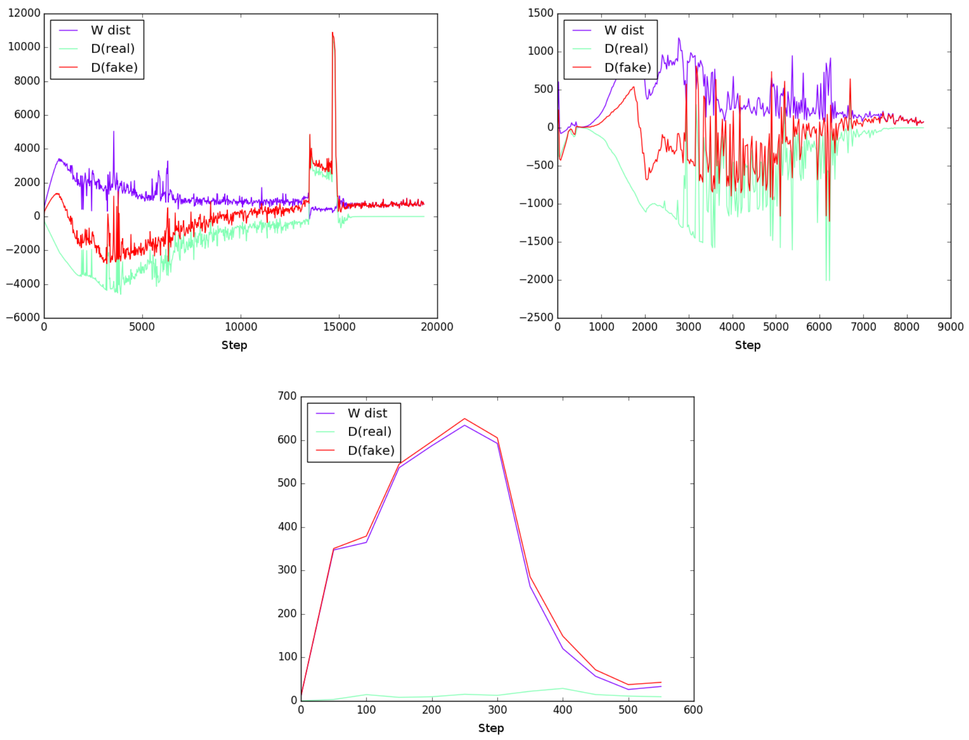


Figure 2: Assignments of costs and Wasserstein distance during training of the positioning task. **Left:** Occasionally unable to solve the task without smoothed absolute costs **Right:** Occasionally able to solve the task without smoothed absolute costs. **Bottom:** Fast convergence with smoothed absolute costs.

Real	Generated
[0 0 0 0 0 0 15 0 0 0 0 0 0 0 0 0 16 0 0 0]	[0 0 0 0 0 0 17 0 0 0 0 0 0 0 0 0 8 0 0 0]
[0 0 0 0 0 0 13 0 0 0 0 0 0 0 0 0 12 0 0 0]	[0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 7 0 0 0]
[0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 2 0 0 0]	[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 18 0 0 0]
[0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 14 0 0 0]	[0 0 0 0 0 0 10 0 0 0 0 0 0 0 0 0 15 0 0 0]
[0 0 0 0 0 0 19 0 0 0 0 0 0 0 0 0 1 0 0 0]	[0 0 0 0 0 0 11 0 0 0 0 0 0 0 0 0 19 0 0 0]

Table 1: Example generations for the positioning task.

The second toy problem consists of a simple grammar, where a sequence of consecutive integers within a range constitutes a word, and words are separated by zeros representing spaces. This tests the model's ability to accurately model local sequences. The vocabulary size for this task was 10 for the reported results with rest of the parameters unchanged.

Figure 3 shows the convergence plots for this task, and Table 2 shows example generations. We can see that the model sometimes produces incorrect tokens skipping to the next word without producing a zero or producing a wrong token in a word. Without using smoothed absolute costs and a real cost multiplier of 10, the model fails to converge for a vocabulary size greater than 4.

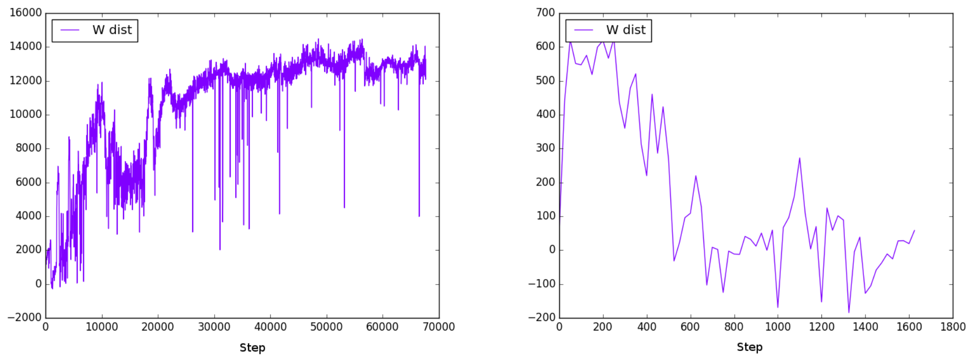


Figure 3: Assignments of costs and Wasserstein distance during training of the simple grammar task. **Left:** Without smoothed absolute costs. **Right:** With smoothed absolute costs.

Real	Generated
[3 4 5 6 0 4 0 5 6 7 8 9 0 9 0]	[3 4 5 6 7 8 9 0 1 2 3 0 4 5 6]
[6 7 8 9 0 1 2 3 4 5 0 4 5 6 7]	[1 2 3 4 5 8 0 3 4 0 5 6 7 0 7]
[6 7 8 9 0 8 9 0 8 9 0 2 3 4 5]	[3 4 5 6 0 9 0 4 5 6 7 0 5 6 7]
[8 0 9 0 2 3 0 8 9 0 9 0 5 6 7]	[6 8 9 0 8 9 0 5 6 7 0 7 0 4 5]
[3 0 5 6 7 8 9 0 6 7 8 9 0 8 0]	[2 3 0 1 0 3 4 0 8 0 7 8 9 0 1]

Table 2: Example generations for the simple grammar task.

5.2 Single Word Generation

Before generating sentences, we tried generating single words. We restricted words to the first 8 characters. If the word was shorter than 8 characters, an end-of-string token `<e>` is produced, followed by padding tokens `<p>`. We found that the model learns to produce the end-of-string token followed by padding tokens very early on, and then spends time learning actual words. The training set contained all the words from the PennTree Bank dataset, and the generator produced words at a character level. The vocabulary size for characters was 37 with embedding size 32. The recurrent hidden state size is 256. We used 10 most recent generators for experience replay, and found it useful to alternate between 20 steps of training the generator and 25 steps of training the discriminator. As the tasks get more complicated, taking more generator and discriminator steps at any given turn helped convergence.

Figure 4 shows how the costs for real and generated samples change over time and the Wasserstein distance converges to zero. Table 3 shows some example generations next to real samples. Although the model makes spelling mistakes, it succeeds in generating 'word-like' examples that fool the discriminator. Using a real cost multiplier of 10 aids in a much more stable learning curve, and the discriminator gradients start approaching zero as the model converges, indicating stability at convergence.

Real	Generated
they<e><p><p><p>	big<e><p><p><p><p>
american	profaid<e>
hong<e><p><p><p>	whoses<e><p>
research	without<e>
even<e><p><p><p>	mr.<e><p><p><p><p>
had<e><p><p><p><p>	june<e><p><p><p>
expected	businest
damage<e><p>	city<e><p><p><p>
time<e><p><p><p>	worline<e>
new<e><p><p><p><p>	must<e><p><p><p>

Table 3: Example generations for the single word generation task.

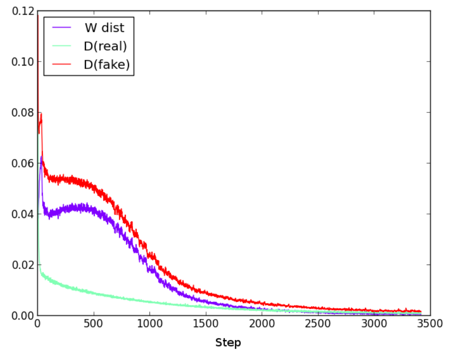


Figure 4: Assignments of costs and Wasserstein distance during training of the single word generation task with smoothed absolute costs and real cost multiplier.

5.3 Sentence Generation

Finally, we tried getting our model to generate sentences using training samples from the PennTree Bank dataset. To make the task easier, we preprocessed the data to include only the 1000 most frequent words, replacing all other words with the unknown token `<u>`. The generative model is character-based, with a vocabulary size of 38, adding period ('.') to the list of characters used by the single word generation model. The embedding size is 32 and the recurrent hidden size is 512. We use experience replay to consider 35 most recent versions of the generator. We alternate between 35 generator training steps and 40 discriminator training steps. We also used a real cost multiplier of 10 to help stabilize training.

Learning language through a adversarial loss alone is hard. The model is able to produce a basic structure of word-like sequences separated by spaces, but does not produce legible sentences. Figure 5 shows the convergence of the model during training, and Table 4 shows example generations.

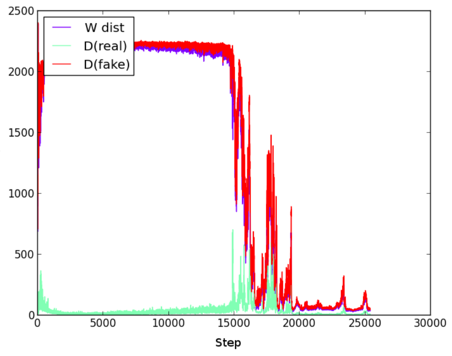


Figure 5: Assignments of costs and Wasserstein distance during training of the sentence generation task with smoothed absolute costs and real cost multiplier.

6 Negative Results

Initial experiments used the original WGAN formulation that clamps weights to a range to enforce the Lipschitz criterion. Although this converged for simple toy tasks, it failed to get close to converging for any of the more complex tasks or the toy tasks with a larger vocabulary.

To help optimization with smoother rewards, we also tried entropy regularization on costs assigned by the discriminator. However, this only slowed convergence and did not help solve any unsolved tasks.

For the sentence GAN, we also tried a word level model, however it failed to learn anything meaningful with the large vocabulary. We find that it is easier for the model to learn with a smaller generation

Real	Generated
however stateswest is n't abandoning its pursuit o	paaesreasusm slsoyrsediens will shiopet udex thovt
in remarks at a white house <u> marking polish herit	frliidaiirin c a Nellserd honst wo g iatines fl cr
uptick an expression <u> that a transaction in a lis	webaisannleiaye oec vra d orpes ary lectr qs fsin
he noted that some minimum margin requirements hav	e Nen—aporoupya ll Nowot seredimuipads miny ki ste
<u> initial offering of N common shares of which N s	pidefthad buyng w itaoals pscecide <u> ofey cireisp
for the nine months ended july N <u> technologies re	sesiiissotretaond deestmr hssel <u> hh sailitwa ion
it is the presence of internal and external enemie	shre uslalgiwilltdary oncuioner cnybev aoorup shh
the austin <u> company which specializes in the dire	a cyictriaeh comay ourgeat priniton ack wioaretins
some analysts believe the company has the ability	dntaygs lailsubrflah styngron wean e hincantpliolt
now many airline companies might become a lot less	abuennyghinlis ibitepiotaenn f hace tslofico p igd

Table 4: Example generations for the sentence generation task.

vocabulary, making character-level tasks more suitable for this model. In other words, the more discreteness there is, the harder it is for the model to converge.

We tried ϵ sampling with importance sampling to get signal from rarer tokens. However, we did not notice any significant difference in the convergence of any task.

We also tried different ways of sharing weights. For example, we tried sharing embeddings between the recurrent networks. We also tried sharing entire recurrent weights, frozen for the actor and critic and the discriminator responsible for learning state representations. The discriminator can learn state representations by actually consuming both real and generated data. The individual components still had their own projections of the hidden states. Although the weight sharing strategies worked for simpler tasks, they could not converge on the word and sentence generation tasks.

Instead of using a fixed number of past generators for experience replay, we considered using an exponential distribution to sample more frequently from recent generators and rarely from old generators. We did not find this strategy to help more than the simpler replay buffer.

7 Discussion

We discussed a way to train a discrete generative model using adversarial objectives using an actor-critic framework to train a generator and a discriminator that learns to differentiate between real and generated sequences. It is important to note that we are able to converge on tasks without using a carefully constrained optimization strategy such as Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] or Proximal Policy Optimization (PPO) [Schulman et al., 2017]. Although our model can solve toy tasks and easier sequence modeling tasks like generating single words, we have yet to produce meaningful natural language sentences.

Future work involves using the methods in this work with better optimizers for reinforcement learning like TRPO or PPO. We would also like to try various curriculum learning strategies. Furthermore, 1D convolutional architectures, such as those using dilated convolutions, have found success in recent sequence modeling work [Yu and Koltun, 2015, van den Oord et al., 2016, Dauphin et al., 2016, Gehring et al., 2017]. We would also like to try dilated convolutions instead of recurrent networks for the recurrent components of our model.

Acknowledgments

I wish to express my gratitude to Emily Denton, for her invaluable suggestions throughout this project. I would also like to thank Prof. Rob Fergus for giving me the opportunity to work in his group.

References

P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. In *NIPS 2016 Workshop on Adversarial Training. In review for ICLR*, volume 2016, 2017.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An Actor-Critic Algorithm for Sequence Prediction. *ArXiv e-prints*, July 2016.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier. Language Modeling with Gated Convolutional Networks. *ArXiv e-prints*, Dec. 2016.

E. L. Denton, S. Chintala, R. Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional Sequence to Sequence Learning. *ArXiv e-prints*, May 2017.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

R. D. Hjelm, A. P. Jacob, T. Che, K. Cho, and Y. Bengio. Boundary-seeking generative adversarial networks. *arXiv preprint arXiv:1702.08431*, 2017.

J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, pages 4565–4573, 2016.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *ArXiv e-prints*, Dec. 2014.

X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley. Least squares generative adversarial networks. *arXiv preprint ArXiv:1611.04076*, 2016.

O. Press, A. Bar, B. Bogin, J. Berant, and L. Wolf. Language Generation with Recurrent Generative Adversarial Networks without Pre-training. *ArXiv e-prints*, June 2017.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

S. Rajeswar, S. Subramanian, F. Dutil, C. Pal, and A. Courville. Adversarial Generation of Natural Language. *ArXiv e-prints*, May 2017.

S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Generative adversarial text to image synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 3, 2016.

J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust Region Policy Optimization. *ArXiv e-prints*, Feb. 2015.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *ArXiv e-prints*, July 2017.

U. Syed and R. E. Schapire. A game-theoretic approach to apprenticeship learning. In *NIPS*, pages 1449–1456, 2007.

U. Syed, M. Bowling, and R. E. Schapire. Apprenticeship learning using linear programming. In *Proceedings of the 25th international conference on Machine learning*, pages 1032–1039. ACM, 2008.

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *ArXiv e-prints*, Sept. 2016.

F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *ArXiv e-prints*, Nov. 2015.

L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

J. Zhao, M. Mathieu, and Y. LeCun. Energy-based generative adversarial network. *arXiv preprint arXiv:1609.03126*, 2016.