Generating Gaussians, Pictures, and Stories with Generative Adversarial Networks

Adam Yala, Hayley Song

1 Introduction

In this paper, we explore the framework of Adversarial Training as introduced by [3] and explore its empirical properties in a variety of generation tasks. Generative Adversarial Networks have recently attracted a lot of interest as a semi-supervised training method and have shown promising results in tasks as varied as Image Generation [6], Transfer Learning [2], Imitation Learning [4], and most recently Text Generation [10] in small vocabulary domains. In this paper, we aim to expose the issues and successes of GANs, as illustrated through a diverse set of generation tasks. Namely, we work through generating a Gaussian (the toy example in [3]), generating images and generating text. All models and experiments were implemented using the Tensorflow library. Code is available upon request.

1.1 Background

The goal of GANs is to train a generator network $G(z;\theta^G)$ that transforms noise vectors z to produce samples from the real data distribution $p_{data}(x)$. The training signal for G comes from a Discriminator network $D(x;\theta^D)$, which is trained to distinguish samples between $G(z)$ and $p_{data}(x)$. D and G form a *minimax* game, with the following Value function, $V(G,D)$:

$$min_G\ max_D V(G,D) = E_{x\ p_{data}}\left[log(D(x)\right] + E_{z\ p_z}\left[log(1-D(G(z))\right]$$

In the global optimum, samples from G become indistinguishable from samples from p_{data}, and $D(x) = D(G(z)) = .5$. As both G and D can be complex highly-non-convex neural networks, we optimize the two numerically via gradient descent and switch between optimizing the two networks. It would be computationally prohibitive to fully optimize D at the inner loop (i.e. train D for thousands of iterations to convergence per optimization of G), and so in practice we train D for k steps for time we train G 1 step. k is often set to small values such as 1, which acts as extreme early stopping, which is a form of regularization.

2 Generating a Gaussian

To illustrate some of the overall properties of GANs, we begin by implementing the algorithm on a simple toy task, namely generating samples from a Gaussian distribution of mean 0 and variance 1. We build a feed-forward neural network G to transforms noise samples z drawn form a uniform distribution ranging from $[-5,5]$ to samples drawn from p_{data} and we build a similar neural network D to discriminate between generated and real samples. Figure 1 shows the graph we would expected given optimal convergence, as displayed in the original paper [3]. We note that samples towards the edges are more spread apart and samples towards the center cluster more, reflecting the Gaussian pdf.

2.1 Architecture

For both G and D, our network takes a single-dimensional input z and runs it through a 3 hidden-layer neural network with *tanh* non-linearities. The layer shapes are: $(1,6)$, $(6,5)$, $(5,1)$. For G, we use a *tanh* as our output node and multiply the result by 5 to match to range to x samples. For D, we use a *sigmoid* as our output node to capture the probability of the sample being from p_data. To optimize the networks, use SGD with momentum and we train for 4500 training iterations, with a 200 samples per mini-batch. We do not employ regularization techniques besides early-stopping. All weights were randomly initialized for symmetry breaking. Unless stated otherwise, we used $k = 1$ for all experiments.

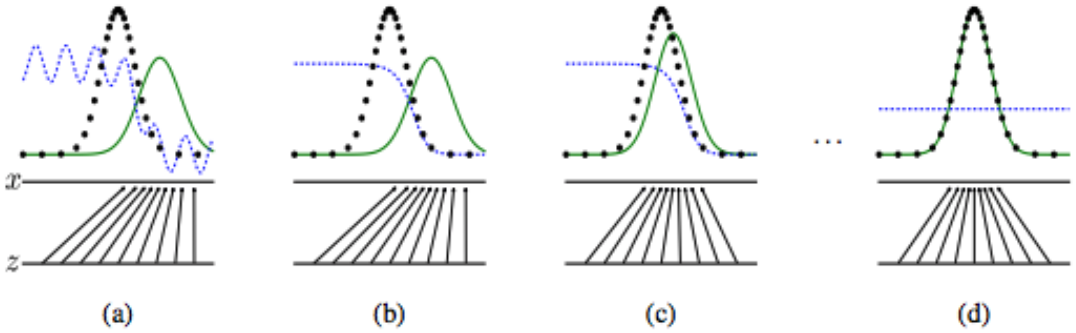


Figure 1: This figure is taken form [3], and shows the optimal behavior of generative adversarial training. The blue line represents D decision boundary, the green the pdf of G, and the dots represent samples from p_{data}. Note that the arrows at the bottom show the transformation from the z domain to x via G. (a) shows the initial parameters of D and G. (b) shows how the graph changes after we optimize D. (c) shows that given a better D, G can further approach p_{data}. (d) shows that after several iterations, we hope to hit an optimum where the Gs samples are indistinguishable from real samples and D is a flat line at .5. We note that this example, as it was in the original paper, is purely illustrative and is not reflective of the actual GAN behavior on this task.

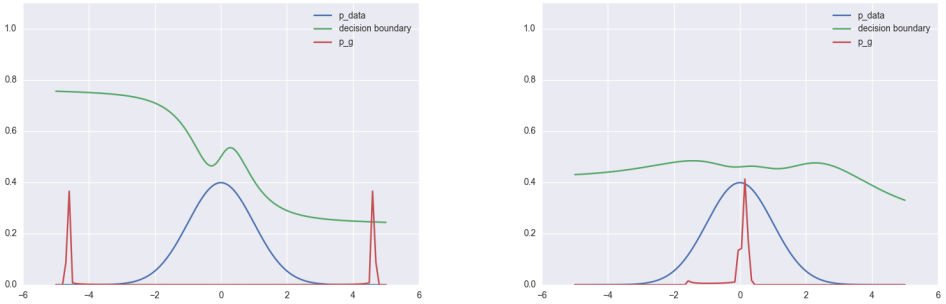


Figure 2: The real data distribution is shown in blue, the discriminator loss function in green, and a historgram representing the density of $G(z)$ is shown in ref. On the left, we show our model before any learning. On the right, we show the model after convergence. We note that G does not learn to draw from p_{data}, but instead learns to only draw the mode of p_{data}.

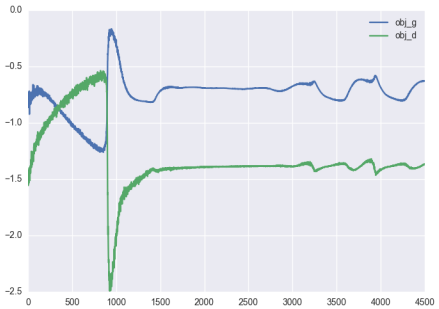
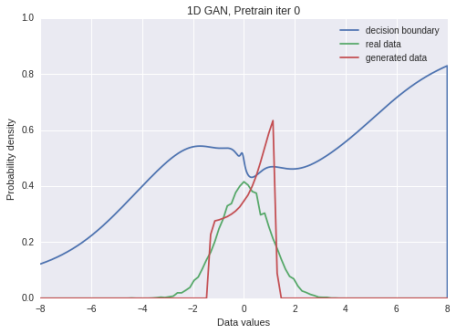
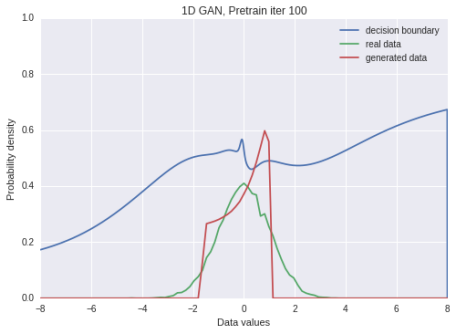


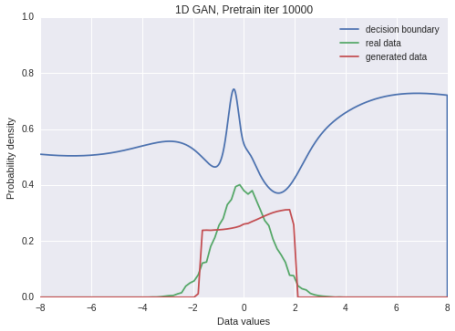
Figure 3: We plot out loss functions across training iterations for both D and G. D is represented in green, and G in blue.



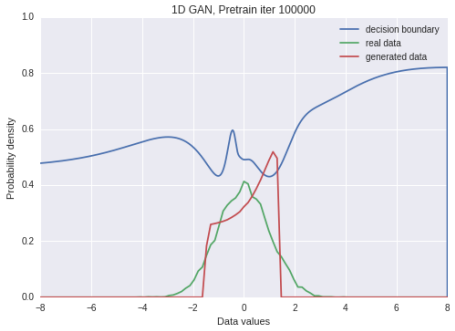
(a) No pre-train



(b) 100 iterations



(c) 10,000 iterations



(d) 100,000 iterations

Figure 4: Effect of pre-train on the final discriminator and generator. Notice the Discriminator's decision boundary becomes more and more flat at 0.5, i.e. random guess

2.2 Convergence and, Observed behavior and the Helvetica Problem

Convergence Figure 3, we plot G and D's loss functions in Blue and Green respectively. We note that as as G error is very high, as around step 1000, D's error is correspondingly low,and that the two networks quickly reach equilibrium. We note that we expect the error to converge to a non-zero value as the global optima for G is at

$$1 - D(G(z)) = log(.5) = -.69$$

Similarly, the global optima for D is at -1.38, which approximately fits the values we see the loss functions. Overall, we found that convergence was relatively robust to random initialization, and hyper-parameters on network structure and learning rate. We experimented with both making the network 10x wider and 3x deeper, and convergence remained relatively quick and stable.

Helvetica Problem Figure 2 shows the distribution of $G(z)$ and D's decision boundary before and training. We note that the learned D decision boundary is approximately at .5 for all samples, as we hoped, and G does learn to approach p_{data}. However, we find G's pdf produces one large spike at $G(z) = 0$. G learn maps all z to the mode of the Gaussian, and ignores other values. This constitutes what is commonly referred to as the *Helvetica Problem* in training GANs, where instead of learning to sample from full probability distribution, G can learns to sample one very good example, such as the mode. We were surprised to encounter this problem even in the toy task. Avoiding the helvetica problem is requires reformulating the problem all together to remove the local optima. In our exploration, we discovered Batch Discrimination [8] , where we give the Discriminator features about several samples to make a prediction instead of one sample alone. In this setup, D can learn to discriminate between the spike we saw in Figure 2 and a real Gaussian. We do not explore this issue further in this project in order to focus on other interesting generation tasks.

2.3 Improvements: Pretraining the Discriminator

We also experimented with the number of pre-train iterations on the discriminator before jointly training the discriminator and the generator. During our experiments, we observed that the discriminator often learned much faster than the generator. One work-around for this problem is to train the discriminator a single batch while training the generator on two batches. Another solution is to pre-train the discriminator. Figure 4 shows how the increase in the number of pre-train

iteration improves the generator (G)'s ability to 'fool' the discriminator (D). If we start the join training of D and G after 100,000 iterations of pre-train on D, the final D's decision boundary becomes flatter around 0.5.

3 Generating Images: CelabA dataset

We trained adversarial nets on CelabA dataset[11] . As in the original paper, we designed the discriminator net(D) with maxout activations and the generator nets (G) with a mixture of rectifier linear activations and sigmoid activations. We settled on the cross entropy as the loss function of both D and G. As mentioned before, D wants its predictions on the real images to be ones, and the predictions on the 'fake' data from G to be zeros. G's main goal is to fool the discriminator, and thus wants D's predictions to be ones.

The original images from the CelabA dataset are cropped to the size of 64 by 64, and the output images are set to the same size.

3.1 Results

Comparison with the baseline Figure 5 shows the generated images after training our model on CelebA dataset, and Figure 5a shows the average image of the training samples, which serves as our baseline. Note that the average image is the Maximum Likelihood Estimator (MLE) under square distance loss function, without the adversarial setting.

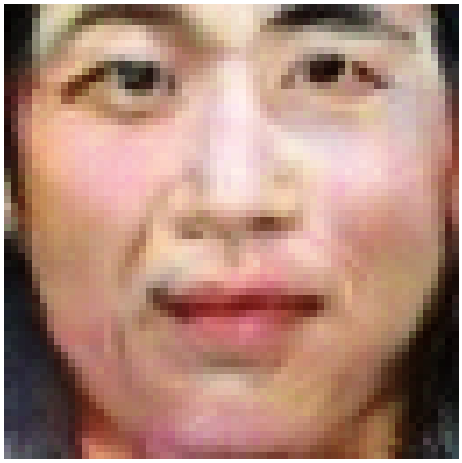
Images generated after intermediate epochs Figure 5b to 5e show the images generated after Epoch 2, 5, 10, and 17. The images start to look more and more 'realistic' as the training progresses. After 2 epochs, the samples look artificial, as if they were generated by computer graphics. Most of the images can be easily detected to be unreal. As the training progresses, we see the improvements of the generator. In particular, we noticed the increase in the variations in the tones in the generated images after Epoch 17 (Figure 5e in comparison to the ones after Epoch 5 (Figure 5c. As the original paper noted, however, even the final images still have defects. In particular, we noticed the lack of three dimensional perception, and confusion caused by the textures in the hairs and backgrounds.

3.2 Discussion

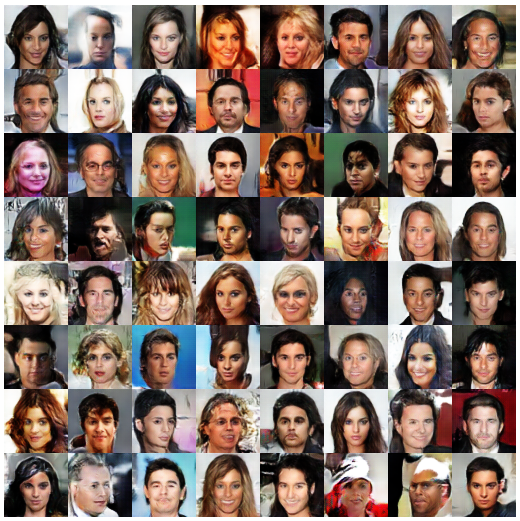
In this experiment, we trained our GAN model on CelabA dataset, which was not done in the original paper. Our results generally agreed with the results from the paper in that the generated samples look surprisingly realistic after sufficient training. Through investigating several checkpoints during the training, we learned that the samples after a few epochs still looked unrealistic; they sometime look alien or completely deformed. As the model goes through more training, however, it learns the variations of the skin colors, and different face shapes. Still yet, the final samples were not perfect and demonstrated weakness in reconstructing details such as hair textures and depth. However, the GAN approach shows great promise in generating high dimensional real value vectors, like images.

4 Generating Text

In this section, we apply Generative adversarial networks to text generation. Text generation, and more generally sequence generation, is an important task in the Natural Language Processing community and spans tasks as varied as Machine Translation, Semantic Parsing but we will specifically focus on Language Modeling, where the task is to learn a probability distribution p_data over sequences of words given a large corpus of text. Today, researcher typically employ recurrent neural networks (RNNs) for these tasks and train their model to maximize the log-likelihood of the observed data. However, these techniques suffer from *exposure bias* [7]. Namely, at train-time, the model learns to predict a tokens given only correct tokens. At test-time, the model does not have correct tokens to predict from, and errors quickly compound. The mismatch between the train and test setup is a fundamental one. GANs provide an attractive framework to attack this tackle exposure bias with. With adversarial training, the generator never sees correct tokens to start with and instead only gets supervision from discriminator. To perform our analysis, We use the Penn-Tree-Bank for experiments, and utilize the same preprocessing as in [5]. We use the most frequent $10,000$ as our our vocabulary, and replace all other words with *unk* tokens.

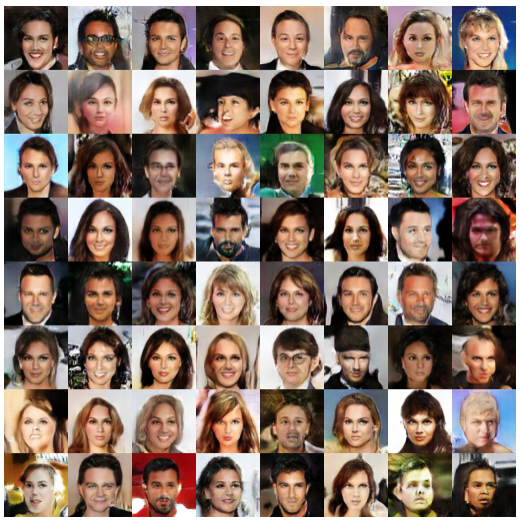


(a) Average image of the training samples; baseline MLE.



(b) Epoch 2.

(c) Epoch 5.



(d) Epoch 10.

(e) Epoch 17.

Figure 5: Samples generated after Epoch 2,5,10 and 17.

4.1 Method

Following the prior work, we apply a recurrent neural network for language modeling. RNNs are commonly applied to sequence tasks as they the hidden state h_t depends on all previous inputs, and conceptually capture arbitrarily long dependencies. At a high-level, we use to build an RNN G, that given a $<start>$ token, and a noise vector z as it's first hidden state, will generate sentences $G(z)$. We also use an RNN to represent D, which given a sequence of tokens of length t, reads through them, and predicts whether or not the sequence came from p_data given h_t. We use the same architecture for both G and D to ensure that both players in the mini-max game have roughly equal expressive power, as we could reach a very poor local optima otherwise. In this work, we employed Gated Recurrent Units (GRUs)[1] to represent both G and D.

GRUs are a popular RNN architecture, that employs neural gates to control the flow of information:

$$i_t = \sigma(W^i x_t + U^i h_{t-1} + b^i)$$
$$r_t = \sigma(W^r x_t + U^r h_{t-1} + b^r)$$
$$c_t = tanh(W x_t + U(r_t \cdot h_{t-1}) + b)$$
$$h_t = i_t \cdot c_t + (1 - i_t) \cdot h_{t-1}$$

Here, i controls how much read in of the new cell and r how much to forget of the old state h. The presence of gates has been shown useful and fighting against vanishing gradients as we unroll RNNs through time, which allows GRUs to capture longer term dependencies.

Challenges Applying GANs to generating sequences poses to two fundamental challenges. Firstly, G starts with randomly sampling and receives supervision from D to gradually get closer to the true distribution p_{data}. Given real value outputs such as our Toy Gaussian case or image generation, a nudge in the direction of more realistic sample has a relatively straightforward interpretation. Given a sequence of discrete tokens, a slight nudge in any given direction does not have as clear of an interpretation to another output, as there is not a token at $token + \epsilon$. Embeddings do not provide an answer to this problem either, as there is not a token for every point in the embedding space. Secondly, D gives a loss for the entire sequence and one cannot the assess the value any individual token choice without considering future tokens. We note that this is fundamentally different from the maximum likelihood case, were we have a clearly defined loss at every time step that does not depend on anything except the current state. The two problems, coupled with a large vocabulary, result in a much more difficult learning problem. To address these challenges, we cast text generation as a Reinforcement Learning problem, where G forms a stochastic policy, and D evaluate our policy and assigns reward. We apply Policy gradient methods to train the G at each iteration, and train D as before.

Casting the problem as Reinforcement Learning The objective our of policy G, is to generate a sequence from a start state $s_0 = z$ to maximize its expected reward:

$$J(\Theta) = E(R_T | s_0, \Theta) = \Sigma_{y \in V} G(y|s_0) \cdot Q(s, y)$$

Where T is the the time horizon (we use fixed length sequences for convenience) , V is our vocabulary, and Q is the expected cumulative reward if we choose token y at s_0 and then follow the policy G. Our first task is to estimate the action-value function Q. Given a full sequence Y_T, we have:

$$Q(y = y_T, s = Y_{T-1}) = D(Y_T)$$

However, this the reward at the last time step is insufficient, we wish to also estimate the action-value function at previous time-steps. Consider a time-step $t < T$, where we have produced tokens $Y_{0:t}$ of a full sentence, or trajectory Y_T. We can view this full sentence as a single monte-carlo roll out from t, and in principle, we can average several rollouts for the following results. We model the Q function for intermediate states as follows:

$$Q_{Y_T}(y = y_t | s_t) = D(Y_{0:t}) + \lambda D(Y_{0:t+1}) + \cdots + \lambda^{T-t} D(Y_{0:T})$$

Let's motivate this Q function. At D processes a given sentence Y, it does so token by token, and maintains a probability of estimate of the sentence being real as it unrolls. We can interpret the intermediate log-probability as a reward for a single choice. For example, if we choose a token *"6.867"* given a sentence, *"I really liked taking"*, D will assign *"6.867"* a much higher intermediate

probability (reward) than choosing *"taking"* as the next token, and this fits what we'd like from our reward function. Given reward at state s_t, we can estimate the value (expected cumulative) at s_t by performing monte-carlo rollouts for the remaining $T-t$ tokens and averaging the cummalative rewards. In practice, we use a single rollout, namely a single generated sentence, for both speed and convenience. We note that the effect of λ here is to balance how myopic to make our policy G. If λ is 0, the policy will learn to pick a token will get it immediately high reward, and be a act greedily in respect to D. If $\lambda = 1$, the policy can learn to take poor immediate tokens if they open the path for a better sentence in the long run.

Policy Gradients Following the REINFORCE[9] derivation, the gradient of the objective function $J(\Theta)$ in respect to our policy parameters Θ_G, can be expressed as follows:

$$\nabla_{\Theta_G} J(\Theta_G) = E_{Y\sim G}[\Sigma_{y\in V} \nabla_{\Theta_G} log\ G(y_t|Y_{t-1}) \cdot Q(y = y_t, s = Y_{1:t-1})]$$

We approximate this expectation by sampling several sentences, and averaging them as a mini-batch.

Generator Architecture To construct G, we employ a a single GRU (no stacking) with a hidden state and cell size of 200. We represent each token as length 200 embeddings, and do not pre-train embeddings. At each time step, as we unroll out GRU, we apply a softmax given our h_t, over the 10000 possible tokens. We sample the next token given the probability distribution, and feed it back into the GRU. For our Q function, we use a decay of .9 and use Adam for all gradient optimizations with the default settings on tensorflow. To improve stability, we utilize gradient clipping with a max gradient norm of 5.

Discriminator Architecture To construct D, we also use a single GRU with 200 length embeddings and length 200 cells and states. As we iterate through x or $G(z)$, where z is the noise vector for h_0, we apply a $sigmoid(W \cdot h_t + b)$, to estimate the probability of the sentence being real at time step t. As with G, we use Adam for all gradient optimizations with default settings, and use the usual the Discriminator loss function as shown in previous sections.

4.2 Evaluation and Baselines

Metric One common evaluation metric in language modeling is perplexity,

$$perplexity(D) = 2^{H(D)}$$

where $H(D)$ is the entropy of D given our language model, which specifies the probability of the text sequence. Perplexity can be interpreted as the average branching factor our language model, and lower is generally better. We use the same number of epochs for each methods and report on the perplexity on the test set.

Baselines To evaluate our new model, we compare the following approaches

1. **Maximum Likelihood Baseline (MLE)**
   In this baseline, we use the G to maximize the likelihood of training data. At each step, the model begins with the correct tokens and learns to predict the next token. We note that this is a much easier learning task.

2. **PolicyGan**
   In this baseline, we use the architecture described in our Methods section, with a G trained via policy gradients and D trained as to distinguish samples from G and D.

3. **PretrainGan**
   In this baseline, we begin by pre-training G for 1 epoch an MLE model, and pre-training D independently to distinguish between real samples and random samples. After pre-training, we apply the policy gradients as before.

4. **MixedGan**
   In this baseline, G take both gradients from MLE and gradients from policy gradients. D is trained as usual.

Method	No train	Epoch 1	Epoch 10
MLE	10,000	1200	250
PolicyGan	10,000	9900	8700
PretrainGan	10,000	1200	700
MixedGan	10,000	nan	nan

Table 1: Perplexities for different GAN baselines for text generation. We note that MixedGan diverged within the first epoch, as it assigns some token in the test set a probability of 0.

4.3 Results and Discussion

Table 1 shows the perplexity of the different baselines. We note that MLE baseline by far out performs the competitors in perplexity and the gains from pretrainGan mostly come from MLE preTraining. This is by more than an order of magnitude. We believe that this is because the adversarial framework faces a significantly harder learning task, as it has to learn language without ever seeing language itself. The search space in this task is incredibly large and though the PolicyGan methods do learn, they do so extremely slowly. In contrast, MLE can quickly move in the exact direction towards given tokens, which constitutes a much simpler learning task.

We can confirm that a helvetica type scenario did not occur in our setup, as the the discriminator maintains about 95-100% accuracy in discriminating between generated and real sentences on all GAN models.

Room for Improvement We believe there is still much room to explore further hyper-parameter choices in order and mixed annotation schemes to improve the stability and the training speed of the algorithm. Another possible experiment is to implement a curriculum type training approach, where we anneal the amount of supervision over time. Another possibility is to employ CNNs instead of RNNs as they can learn much more quickly, simplifying an already difficult learning task or attempting a domain with a smaller vocabulary. It would also be interesting to explore human evaluation of generated sentences, but we leave these thoughts to future work over Christmas.

5 Conclusion

In this work, we explored the empirical trade-offs in using GANs across a variety of generation tasks. We conclude that GANs are a widely applicable tool and the interest in them is justified but there is still much future work to be done in making them robust. As shown in the toy task, the helvetica problem arises in even trivial tasks. We showed that image generation task performs, and GANs have the ability to produce impressive results, but those results have not been shown on larger images due to difficulty in achieving coherence. We also discovered the challenges in applying GANs text generation, and more generally the challenges when applying PolicyGradients to very large search spaces.

5.1 Division of Labor

Adam contributed to the Gaussian and text-generation experiments, and wrote the corresponding sections.

Hayley contributed to the Gaussian and image-generation experiments and wrote the corresponding sections.

References

[1] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[2] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. *arXiv preprint arXiv:1409.7495*, 2014.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[4] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances In Neural Information Processing Systems*, pages 4565–4573, 2016.

[5] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[7] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[8] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[9] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[10] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.

[11] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.