DLI Teaching Kit

Lab 3, Sample Solution

1 Generative Adversarial Networks

1. Generative models attempt to model how data is generated, based on the training examples supplied. In other words, a generative model looks at examples of real data and attempts to learn that which would have generated that data. These models are evaluated based on how realistic their generated data is. If a model can create a realistic piece of data, we extrapolate to conclude that it understands fundamental aspects of the data. Generative modeling fundamentally is very useful for evaluating representation learning, pattern recognition, and summarization of data.

2. Unsupervised learning requires one to come up with a useful and interesting learning objective, as there are no labels, and thus there is no obvious learning objective minimizing the negative log-likelihood of the correct label, like in supervised learning.

   Some approaches to unsupervised learning that have been used previously include clustering methods and autoencoders. Clustering methods use the given features of data to cluster similar examples together. Clustering is fundamentally different from Generative Adversarial Network (GAN) as clustering is not a generative model and generally uses features that are already given, or nascent in the data.

   Autoencoders are similar to GANs in that they also involve generation. Autoencoders are generally trained to compress data and then reconstruct the original data. Compression is obtained by either forcing the data into a lower dimensional representation or by enforcing a sparsity constraint on the representation created. Autoencoders can also be built to denoise data and generate the original, clean data.

   Autoencoders can better encode variational information in the data by introducing a latent variable constraint in autoencoder. In variational autoencoders (VAE), the model is constrained to make its representation defined by a high dimensional distribution over a latent variable. Thus,

variations in generations can be created by sampling from the latent variable.

Autoencoders are fundamentally different from GANs. Autoencoders are trained to reconstruct the data examples. GANs, on the other hand, have two networks that together form the model: one network is trained to generate data and the another is trained to distinguish between generated and true data. Thus, the generator is not trained to reconstruct data directly, but to simply trick the discriminator into thinking that the data it generates is real. The generator ideally learns the distribution of features that makes the data real, thus enabling it to distinguish generated and true data.

The objective function for a GAN is given as follows,

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\log D(x,y)] + \mathop{\mathbb{E}}_{z \sim p_{model}(z)} [\log(1 - D(G(z,y),y))]$$

Some of the key differences between GANs and other generative models are,

(a) The design of the generator has very few restrictions. This compares favorably in comparison to Boltzmann machines, in which there are very few density distributions that allow tractable Markov chain sampling. For Independent Component Analysis (ICA), the generator must be invertible and the latent space Z must have the same dimensionality as the input x.

(b) No Markov chains are needed, which is an advantage over Boltzmann machines and GANs

(c) No variational bounds are needed as in VAEs. VAEs are conjectured to be asymptotically consistent, but this hasn't yet been proven.

(d) GANs are subjectively considered to produce better quality samples than other methods.

(e) GANs have the disadvantage that they aim to find the Nash equilibrium which is a much more difficult problem than optimizing an objective function. This can make training effectively challenging.

If we restrict ourselves to generative models that utilize a maximum likelihood principle, then they can all be seen as attempting to minimize the **KL divergence** between the data and the model:

$$\theta^* = \arg\min_\theta D_{KL}(p_{data}(x) || p_{model}(x;\theta))$$

Following is an enumeration of other generative models and their specific differences to GANs:

a Tractable explicit density models

i. Fully visible belief nets: they use the chain-rule to decompose the probability of an n-dimensional vector x into product of one dimensional probability distributions:

$$p_{model}(x) = \prod_{i=1}^{n} p_{model}(x_i|x_1, ..., x_{i-1})$$

So the computation of each x is now composed of non-parallelizable $O(n)$ steps. This leads to large computational cost (time). GANs ,on the other hand, do not have this constraint.

ii. Nonlinear independent components analysis (ICA): they assume that latent vectors, z, can be non-linearly transformed to arrive at x. And so,

$$p_x(x) = p_z(g^{-1}(x))\| \det(\frac{\partial g^{-1}(x)}{\partial x})\|$$

However this requires that z be of the same dimensionality as x and that function g be continuous, differentiable, and invertible. GANs do not have these constraints.

b Explicit distribution models requiring approximation: these include models that work with tractable approximations to intractable probability distribution functions representing maximum log-likelihood:

$$\mathcal{L}(x;\theta) \leq logp_{model}(x;\theta)$$

i. Variational approximations: Variational autoencoder is a member of this family of models. The main drawback of the variational method is that with a weak prior distribution approximation, or a posterior distribution approximation, the gap between the approximated $\mathcal{L}$ and the true likelihood can result in a poor p_{model}, which is different from p_{data}. Whereas GANs, given sufficiently large model and infinite data, are designed to settle at p_{data}. Furthermore, on the qualitative side of things, it has been observed that GANs tend to generate better quality images (when working with image datasets) than variational autoencoders.

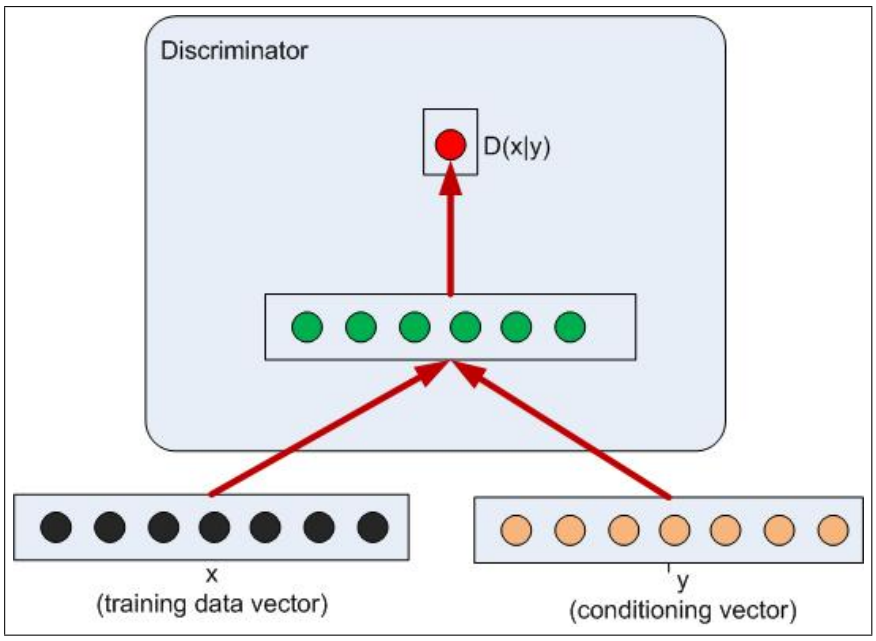
ii. Markov Chain Approximations: they can be used to sample from a training set as well as generate samples, in a generative model. For sampling from the training set, using a Markov chain can sometimes yield very good representative distributions, however this is not guaranteed and cannot always be measured for convergence to the true data distribution. Furthermore, for generative steps, using a Markov chain method is inefficient, because it works with long sequential chains. Models like Boltzmann machines that are based on Markov chains have not performed well in high dimensional datasets, such as ImageNet. GANs are specifically designed to avoid these constraints.

c Implicit Density Models: they do not need to explicitly define a density function, but offer a way to train the model by indirectly interacting with p_{model}, usually by sampling from it.

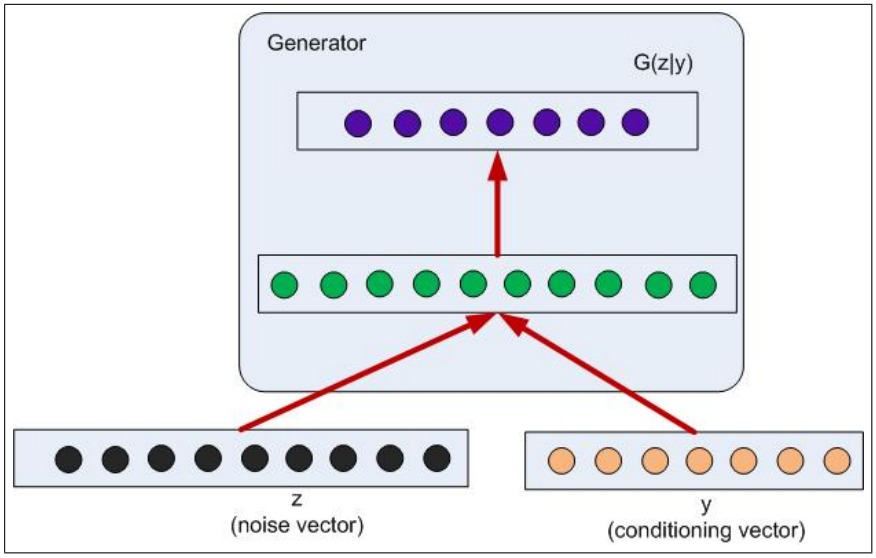
i. Markov Chain: based generative stochastic network (GSN) draws samples from p_{model}. These sample draws define a Markov chain transition operator. As mentioned previously, Markov chains often fail to scale to high dimensional spaces, and impose increased computational costs for generation.

ii. Direct: GANs are the notable member of this family, in addition to kernelized moment matching models.

3. Conditional GANs: these can be represented using the following simplified diagrams (from Mirza et al. [1]):



In such a model, the discriminator output is conditioned on the conditioning vector (or context y). This could be a word embedding or sentence embedding, as in the case of a word or sentence description of an image. The discriminator output is a boolean that claims the input x is real or not, i.e. if it comes from the original data or was generated by the Generator.



The generator, on the other hand, generates a vector $G(z|y)$ that is of the same dimensionality as the supplied training example x. The input to the generator is a vector z, which is a noisy version of x, and the conditioning vector y. Note that it is the same conditioning vector that is used with the discriminator.

The objective function is given by

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\log D(x,y)] + \mathop{\mathbb{E}}_{z \sim p_{model}(z)} [\log(1 - D(G(z,y),y))]$$

2 Project: Sequential GANs

2.1 Overview

For our project we decided to explore sequential GANs on text data. We will describe several model architectures we tried, and explain our final model design in detail.

The overall structure of GANs for sequences involves a recurrent network cells (LSTM or GRU) for the generator and one for the discriminator. The discriminator also includes an MLP layer that takes the final hidden state of the discriminator and outputs a scalar probability of the sequence being real.

2.2 Problem Setting

The main challenge with generating text with GANs is that the generator needs to make a series of discrete choices in picking words from the vocabulary. This is a difficult task to train effectively. If the generator is trained to generate a

sequence of words, and the sequence is then fed to the discriminator, the error from the discriminator cannot then be passed to the generator through regular back-propagation, since we cannot differentiate through discrete choices.

This leaves us with two main routes to take,

- Finding a reasonable way to make text generation differentiable. We explore this method using the Gumbel-Softmax.
- Using a sampling method to estimate the error gradient like REINFORCE.

2.3 Differentiable Approach: Gumbel-Softmax

Our first approach was to try to make text generation differentiable. There are a few papers that have done this somewhat successfully like Alex Lamb et al. [2] and Ankit Vani [3]. Both models make the signal from the discriminator to the generator differentiable by having the discriminator directly read the hidden states of the generator instead of the word embedding sequence created by the generator.

For our approach, we were inspired by recent papers about the Gumbel-Softmax (Eric Jang et al. [4]), which allows us to easily "turn-up the temperature" on softmax outputs and helps to better approximate sampling from a distribution in a differentiable manner. In other words, the Gumbel-softmax has a temperature parameter that allows us to increase or decrease the entropy, or "discreteness", of the model's decision over the softmax options.

Figure 1: The following figure is taken from Eric Jang et al. [4]

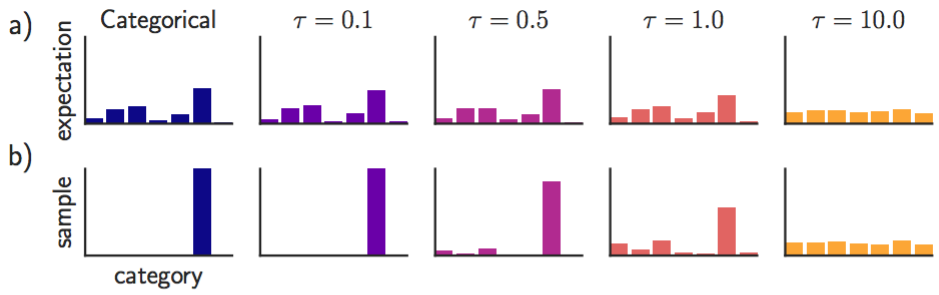


Figure 1: The Gumbel-Softmax distribution interpolates between discrete one-hot-encoded categorical distributions and continuous categorical densities. (a) For low temperatures ($\tau = 0.1, \tau = 0.5$), the expected value of a Gumbel-Softmax random variable approaches the expected value of a categorical random variable with the same logits. As the temperature increases ($\tau = 1.0, \tau = 10.0$), the expected value converges to a uniform distribution over the categories. (b) Samples from Gumbel-Softmax distributions are identical to samples from a categorical distribution as $\tau \to 0$. At higher temperatures, Gumbel-Softmax samples are no longer one-hot, and become uniform as $\tau \to \infty$.

For our model, we used the generator's current time step's hidden state to output a Gumbel-Softmax probability distribution over the vocabulary thus predicting what the next word to generate should be. We then used these probability weights and took a linear combination of the vocabulary to make a mixed-word embedding to feed in for the next timestep.

We tried our first model on simple character level RNNs reading words. The goal of the generator was to learn to generate strings of characters that looked like plausible words (which we took to mean pronounceable and/or having vowels intermixed with consonants).

Our initial simplified model, for which we used a vocabulary of words with constant length, did not work well and seemed to generate garbled sequences, which were rarely words. We even tried initializing the vocabulary matrix with pretrained character embeddings[1], but this did not help our rudimentary model learn. We tried hyperparameter tuning and changed the learning rate and the temperature of the Gumbel-Softmax. The tuning did not seem to significantly improve performance either.

We think that the main problem with the character generation decisions made in a "soft" manner is that the sequence of characters the RNN reads in are always a mixtures of multiple characters. This makes it difficult for the RNN to learn since the inputs can be quite inconsistent.

Ultimately, we decided not to continue exploring this approach of making generation decisions soft—it is very possible though that this route to generation can be made to work well, or even better than the policy gradient route that we settled on.

2.4 Policy Gradient Approach: Discrete GANs

It is common practice to use policy gradients in text problems to help train parts of the model that make discrete choices (Danny Yogatama et al.[5] and Caglar Gulcehre et al.[6]). These models generally use the log-likelihood of the probability placed on the correct decision by the downstream model as the reward for the policy gradient trained portion of the model. In the case of sequential GANs this would be done by assigning the negative log-likelihood of the discriminator as the reward to the generator.

For our policy gradient model, we were primarily inspired by Maximum-Likelihood Augmented Discrete Generative Adversarial Networks, by Tong Che et al.[7] and Boundary-Seeking Generative Adversarial Networks, by R Devon Hjelm et al. [8]. In these papers, they found a more stable reward function to assign to the generator.

In the standard negative log-likelihood reward, the reward the generator would get for choosing a word would be the negative log-likelihood of the whole sequence generated being determined real by the discriminator, $\ln D(G(z))$.

[1] We used the generated text file from this repository: https://github.com/minimaxir/char-embeddings. It contains 300D character embeddings derived from the GloVe 840B/300D dataset.

Instead, these papers found that the following reward works better,

$$\text{Reward} = \frac{r_{Di}}{\sum_i r_{Di}}$$

where i is minibatch dimension and $r_{Di} = \frac{D(G(z_i))}{D(G(z_i))-1}$.

The motivation behind their reformulation of the reward is that optimally, the discriminator will make predictions such that

$$D(x) = \frac{p_{data}}{p_{data} + p_{generated}}$$

Which means that,

$$p_{data} = p_{generated}\frac{D(x)}{1 - D(x)}$$

They found in the paper that this novel formulation of reward provided a signal to the generator that had lower variance. In our initial experiments with both reward formulations, we also found that the new formulation of the reward provided a more stable and lower variance signal. Therefore, we performed the majority of our experiments with the new formulation of the reward.

2.5 Dataset and Preprocessing Procedure

For all our experiments, we made clear sentence delimitations between examples, unlike with the language model training procedure. Thus each real example in our batch was a real sentence. We did this because we wanted to have our generator learn to generate complete sentences from start to end. We thus dealt with the variable length of sentences by padding (and truncating) sentences to a standard maximum sequence length.

We did not limit the vocabulary size and simply initialized all word embeddings with a uniform distribution between -0.05 and 0.05. We also included a start of sentence, $\langle$SOS$\rangle$, and end of sentence, $\langle$EOS$\rangle$, symbols. We allowed the generator to unroll to either a maximum length, or to the earliest instance of an $\langle$EOS$\rangle$ symbol. We did not feed $\langle$EOS$\rangle$ symbols to the discriminator.

In terms of data, we tried our model on the Penn Treebank and Gutenberg datasets, but focused most of our energy on the Stanford Natural Language Inference (SNLI) dataset. The main reasons for this decision was that when we trained our model on the Penn Treebank data, our model put a very high likelihood on $\langle$UNK$\rangle$ token and would often generate long strings of just UNKs when we generated greedily. Moreover, the SNLI sentences were much shorter (we used the hypothesis sentences which have an median length of 7 tokens), which made it easier to train with large batches on NVIDIA GPUs (as we found that using larger batch sizes lead to much more stable error signals for the generator and better learning overall).

In order to utilize larger batch sizes when training with the adversarial objective, we truncated sentences that were longer than a set max. length. For SNLI, we truncated sentences to a maximum of 50 tokens (on the training set

only; we did not truncate the validation and test set sentences). Moreover, we truncated the Gutenberg dataset sentences to 150 tokens on the training set as well.

2.6 Model Architecture and Training Procedure

We pretrained our generator with a language model objective, cross-entropy of predicting the next word, with teacher forcing. Then we trained the discriminator for K iterations (5 to 50 steps) on the real and generated data, without updating the generator to ensure that the gradients from the discriminator that would be passed to the generator would be reasonable. We then iterated between training the discriminator for a step and then training the generator for a step. When training the generator, we sampled from the distribution that the generator generated (over the vocabulary), to choose the next word. The sequence of chosen words were also used as input into the discriminator.

We used 50 dimensional, one-layer LSTM recurrent neural networks for both the generator and discriminator. We also shared a 50 dimensional embedding matrices ($[V \times 50]$) between both models. For the generator, we also had one linear layer that took each time step's hidden state and fed it into a softmax over the full vocabulary of words. The discriminator also had one linear layer that took the final time-step's hidden state and transformed it to a scalar. This scalar, when passed to a sigmoid function, represented the confidence that the discriminator thought that the example was real.

We experimented with both SGD and ADAM optimizers, and found that the SGD optimizer lead to better learning. The ADAM optimizer for the discriminator generally lead to a discriminator that was too strong too early to be tricked by the generator. We experimented with learning rates between 0.1 and 0.00001.

We generally stopped training either when the discriminator got too strong or when the performance of the generator on the original language model task began to go down significantly.

We also found that changing the temperature on the softmax for the generator over the vocabulary (what words to generate next) also significantly changed performance; we found that a temperature of 0.5 tended to lead to better performance, in comparison to temperatures of 1.0 or 0.75. We discuss this in our results and findings section.

2.7 Results and Findings

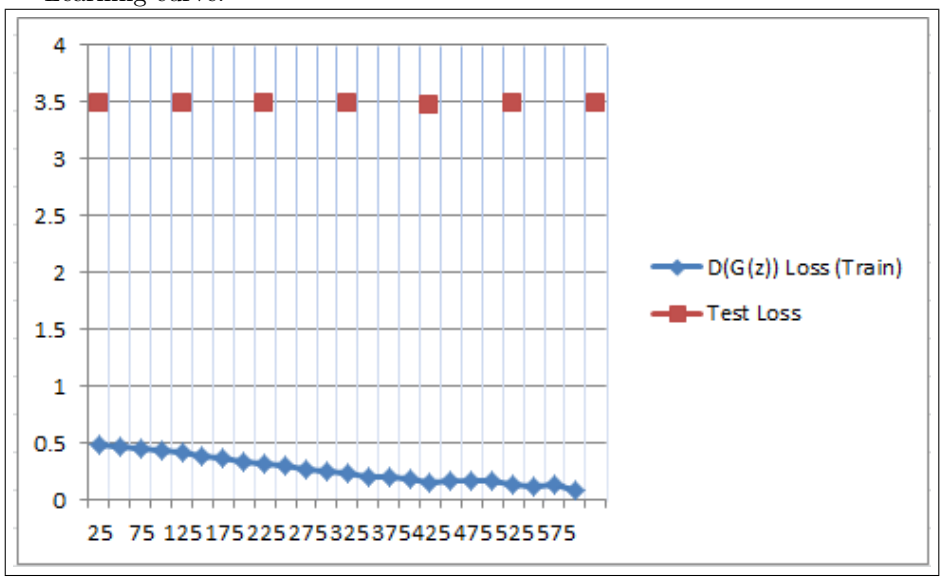
2.7.1 Performance

Overall, we were not able to reproduce the results in Tong Che et al. [7], which found that through adversarial training they were able to improve the performance of their 200 dimensional GRU generator on the language modeling task from a perplexity of 141.9 down to 131.6.

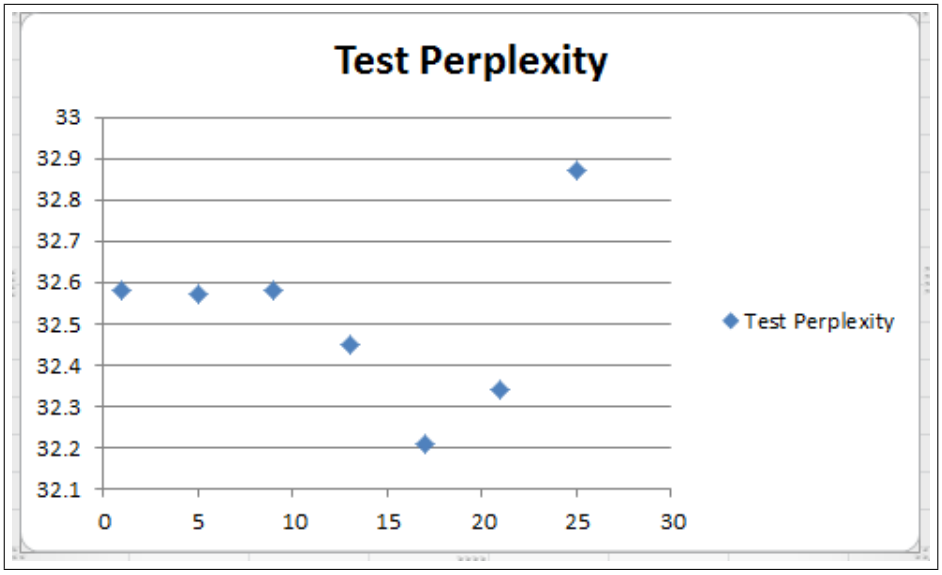
When we trained out 50 dimensional LSTM language model with weight-tying on the Penn Treebank, we got a much lower initial perplexity of 90.12 and found that with adversarial training, we were not able to improve the perplexity. It could be that Che et al. had pre-trained their generator to a lower perplexity, they might not have seen such improvements through training with the adversarial objective.

However with SNLI, we saw some minor improvement with adversarial training. When training on the SNLI dataset, we had an initial perplexity of 32.58 after pre-training the generator with a language model objective for several epochs. After training adversarial training, we were able to improve the perplexity only slightly to 32.21.

Learning curve:



Test Perplexities:



2.7.2 Observational Results

We found that our model was, on some occasions, able to learn to generate more reasonable and realistic sounding sentences through adversarial training.

Since language models have exposure bias, they are often not good at recovering after they predict an incorrect word during the sequence. Thus there were often some extremely degenerate examples that didn't produce an end symbol and continued to repeat a word or a phrase. We found that through adversarial training, the generator learned to truncate these sentences.

We're providing some examples of of sequences generated by our generator. The first line is that generated by the language model, and each successive line is generated after 500 steps of training, up to 2500 training steps. The words are generated by choosing the most likely next word one at a time (greedy decoding).

- a street is a crowd .
  a street corner is playing a guitar .
  a street corner of a building .
  a man is wearing a hat .
  a street is a crowd .
  a street vendor is playing a guitar .

- there 's a man in a blue shirt and a white shirt and a blue shirt is playing with a child .
  there is a man in a blue shirt .
  there are people .
  there are a man in a blue shirt .

there are a lot of people .
there is a man in a blue shirt .

- there are people in the field .
  there is a man in a blue shirt and a blue shirt and a blue shirt and a blue shirt and a blue shirt and a blue shirt and a
  nobody is playing with the ball .
  there are people in the park .
  there are people in the street .
  nobody is playing a guitar .

- a man in a blue shirt and a white shirt and a white shirt is playing with a ball .
  a street corner .
  a man in a blue shirt and a blue shirt and a blue shirt is playing with a ball .
  a man in the blue shirt is playing a game .
  a street in a park .
  a street in a park .

2.7.3 Softmax Temperature

We found that the temperature of the softmax used to generate words was very important in the performance of the model. Below we give examples of sequences generated at different temperatures of softmax.

All of the following sentences were generated with the same random noise, but with different temperatures.

- **Take most likely next word; No sampling**
  the man is playing with a ball .
  the man is playing with a ball .
  the man is sitting on the floor .
  there is a woman in a white shirt is sitting on the floor .
  a woman is sitting on the couch .
  the man is playing with a ball .
  a woman is sitting on the couch .
  there is a woman in a white shirt is sitting on the floor .
  the man is playing with a ball .
  there is a woman in a white shirt is sitting on the floor .
  the man is playing with a ball .
  the man is playing with a ball .
  a woman is sitting on the couch .
  the man is playing with a ball .
  the man is playing with a ball .

the man is playing with a ball .
the man is playing with a ball .
the man is playing with a ball .
the man is playing with a ball .
the man is playing with a ball .

- **Temperature is 1.0; sampling**
  a boy is riding another ricker as watch .
  the balcony is swimming .
  the son thrown street .
  four children are standing near the park in the street kitchen .
  this picture is waiting for a crowd .
  eight men are on apples
  a young woman with numbered wings , blue riding his bike
  the man is seeing sync
  the men are mid-throw in an elder .
  two girls are standing on in .
  the blond woman is broken on the beach
  two women are sitting on the side of the beach , and one behind
  senior children watch their play cement .
  the bright team in a red shirt smiles on free is at laundry .
  the lovers are asleep in the grass .
  an african girls is playing light to there is a bartender practicing violently
  romney is blur at the cat with hull obama .
  nobody does n't be not other some with his dog .
  three dogs are on their bike .
  a woman and a woman elmo under behind them 's .
  children are resting on one of the beach at a cover

- **Temperature is 0.5; sampling**
  a man is standing in front of a woman in a white shirt and white shirt is walking in the street .
  the child is in the pool .
  the man in the street .
  there is a woman wearing a black shirt .
  three men are playing soccer
  the man is on the beach .
  a woman is standing with the other people .
  the man is sleeping .
  the woman is sleeping on the couch .
  two men are walking on the beach .
  the man is wearing a green shirt .
  the man is going to be on the street .
  one person is being played .
  the man is holding a red shirt .

the girl is sitting on the floor .
the man is in a pool .
three girls are playing with a toy on the beach .
two men are on the beach .
the man is on the shore .
a man is waiting for a woman

2.7.4 Adversarial Learning

We found that adversarial training changed the sentences generated in very interesting ways. Below we show examples of how sentences changed as they were trained with the adversarial objective.

Each list represents how one fixed noise vector ended up generating different sequences as the generator was trained with an adversarial objective. The intervals between each example is 25 update steps of the generator and discriminator.

- a man is standing in front of a woman in a white shirt and white shirt is walking in the street .
  an old man is walking
  two men are outside .
  a woman is holding a white and black shirt , with a black shirt and is wearing blue .
  the man is playing the guitar .
  the girl is sleeping

- the child is in the pool .
  the man is a man
  the man is sitting on the floor .
  the man is going to the train .
  the people are playing with each other .
  the man is doing a trick on the beach .

- the man in the street .
  men are outside .
  the people are outside .
  the man is standing in the middle of the street .
  the man is standing in a blue shirt
  a woman is sitting on the floor .

- there is a woman wearing a black shirt .
  there is a dog is riding a bike .
  there is a woman wearing black jeans is standing on the beach .

there is no one of the woman
there is no one of the street .
the woman is riding his bicycle .

- three men are playing soccer
  a woman is standing on a dock
  a man is on the beach .
  the man is going to work .
  two dogs are playing with a frisbee .
  two boys are playing in the water .

- the man is on the beach .
  the man is in the middle of the street .
  the man is doing a trick .
  the boy is in the blue shirt .
  a woman is sitting on the floor on the side of a mountain .
  a man is sitting on the beach .

- a woman is standing with the other people .
  three dogs are playing .
  a woman is playing in the water .
  a woman wearing a black shirt is sitting on the couch .
  a woman is playing with an old man .
  the women are playing on the street

- the man is sleeping .
  there is a man on the beach
  the woman is playing with an old child
  there is a man wearing a pink shirt and blue pants is sitting on the sidewalk
  .
  people are playing in the snow .
  there is a woman in a pool .

- the woman is sleeping on the couch .
  the man is trying to be a woman
  the woman is walking in the sun .
  the man is in the snow .
  the dog is playing on the beach .
  an old man is walking .

- two men are walking on the beach .
  the man is standing in the snow .

the man is white
two children are playing in the snow .
the man is at the beach .
two women are playing in the snow .

2.8 How is a Sequential GAN on Text Different from a Language Model?

The primary addition that unconditional GANs bring over simply generating sequences with a language model is that language models are trained with teacher forcing, which means that even if language models predict the "wrong" word at a given timestep, at the next timestep the "correct" word is still fed in as the input. Teacher forcing leads to the issue of "exposure bias" as described by Marc'Aurelio Ranzato et al. in [9]. Exposure bias is the bias that sequence generators have when trained with teacher forcing and leads to sequence generators that are poor at "recovering" from making a "wrong" generation decision in a previous time-step, as they were never trained to do so.

2.9 Further Work

One general way to improve our model would be to adopt more techniques generally used in reinforcement learning. This would include adding a baseline estimate to the reward to encourage more stable learning. We could also add Monte Carlo Tree/beam search method to our model to improve the way that sample sequences are selected as was done in several of the papers exploring text generation with GANs. In addition, we could employ a value network to estimate the value of choosing a given word at the next time step (instead of just using the policy gradient, estimate the value of states).

Other improvements include utilizing a hierarchical softmax over the vocabulary, as a softmax over the whole vocabulary is an expensive operation.

Lastly, it would be very interesting to explore conditional generation of sequences, like conditioned on a sentence, generate a sentence that paraphrases that sentence or contradicts that sentence.

Overall, we found that training the LSTM generator with policy gradient alone to be very difficult. We think that conditional generation of sequences could be explored further with variational autoencoders and perhaps adding a secondary loss function for the generator as it trains against the discriminator, like a language model loss. Perhaps the generator could also get a reward based on the log-likelihood that a language model puts on the next word.

References

[1] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014). *https://arxiv.org/abs/1411.1784*

[2] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, Yoshua Bengio. "Professor Forcing: A New Algorithm for Training Recurrent Networks". (Oct 2016). *https://arxiv.org/abs/1610.09038*

[3] Ankit Vani. "Adversarial Objectives for Text Generation". (2016). *http://cs.nyu.edu/ akv245/advtext/report.pdf*

[4] Eric Jang, Shixiang Gu, Ben Poole. "Categorical Reparameterization with Gumbel-Softmax". *https://arxiv.org/abs/1611.01144*

[5] Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, Wang Ling. "Learning to Compose Words into Sentences with Reinforcement Learning". (2017). *https://arxiv.org/pdf/1611.09100.pdf*

[6] Caglar Gulcehre, Sarath Chandar, Yoshua Bengio. "Memory Augmented Neural Networks with Wormhole Connections". (Jan 2017). *https://arxiv.org/abs/1701.08718*

[7] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, Yoshua Bengio. "Maximum-Likelihood Augmented Discrete Generative Adversarial Networks". (Feb 2017). *https://arxiv.org/abs/1702.07983*

[8] R Devon Hjelm, Athul Paul Jacob, Tong Che, Kyunghyun Cho, Yoshua Bengio. "Boundary-Seeking Generative Adversarial Networks". (Feb 2017). *https://arxiv.org/abs/1702.08431*

[9] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, Wojciech Zaremba. "Sequence Level Training with Recurrent Neural Networks". (Nov 2015). *https://arxiv.org/abs/1511.06732*