Text Generation using Generative Adversarial Networks

Vincent-Pierre Berges vpberges@stanford.edu
Isabel Bush ibush@stanford.edu
Pol Rosello prosello@stanford.edu

Abstract

Recurrent neural networks have recently found success in statistical language modeling, the task of assigning probabilities to sequences of words according to how likely they are to appear in text. Language models can be used to generate synthetic text by sampling words one at a time. However, while text produced in this way has relatively good grammatical structure, it tends to lack long-term coherence. In particular, it is very easy for people to distinguish between text written by human authors and text generated by sampling from current state-of-the-art language models. In this paper, we quantitatively show the limitations of using language models in a generative manner by training discriminator models to distinguish between real and synthetic text produced by language models. Our best discriminative model reaches 90% accuracy on the test set, demonstrating the weaknesses of using language models in a generative manner. We propose the use of generative adversarial networks (GANs) to improve synthetic text generation. We describe architectural changes to recurrent language models that are necessary for their use in GANs, and pre-train these generative architectures using variational auto-encoders.

1 Introduction

Automatic text generation is a difficult challenge in artificial intelligence. While recent work has demonstrated considerable improvements in producing short responses to direct questions [1] or translating text [2], generating convincing, unprompted passages is an unsolved task in natural language processing. Perhaps the most successful approaches so far involve language modeling. Given a large corpus of text, language models are trained to estimate the probability that a sequence of words will appear together. While they are mostly used to improve other tasks such as machine translation and speech recognition, language models can also be used to generate synthetic text one word at a time by sampling from the probability distribution over words conditioned on all previously-generated words. Figure 8 in Appendix A shows example text generated in this manner from a state-of-the-art Recurrent Neural Network (RNN) language model. While the text generally preserves the grammatical structure of English, it is very easy to tell that it has not been written by a human.

In this work, we aim to improve the quality of synthetic text generation through the use of variational auto-encoders (VAEs) and generative adversarial networks (GANs). An auto-encoder encodes each input into a latent representation, and then attempts to reconstruct the input by decoding the latent representation. A VAE enforces that the latent representation forms a normal distribution, so that by feeding random samples from a normal distribution into the decoder, new synthetic samples can be produced. GANs take a different approach, training a discriminative model in tandem with the generative model. During training, the two models have opposing goals, with the discriminator attempting to distinguish the output of the generator from real samples from the dataset and the generator attempting to make the discriminator guess incorrectly by generating better synthetic samples.

In this paper, we evaluate the quality of generative text models by training discriminative models to distinguish between real passages and synthetic text produced by the models. We propose using GANs to improve generative text models, and describe architectural modifications to generative text models that are necessary to train them in a GAN. Before incorporating our generative models into the GAN framework, we pre-train them as decoders in sequence-to-sequence VAEs.

2 Related Work

GANs were first proposed for image generation by Goodfellow *et al.* and have shown promising success at generating synthetic images that resemble real photographs in a number of previous works [3][4][5][6]. In image generation, the input to the generator is a random vector sampled from a continuous space, and the output is continuous pixel values. Such an approach is much more difficult for text generation since words are discrete. Outputs of language models are generally probability distributions over the vocabulary for the probability of each word at each timestep, and the output word is sampled from this distribution. Unfortunately, it is not possible to back-propagate through this sampling step and thus such a language model may not be used directly as the generator in a GAN.

Although we are unaware of any previous work attempting to generate text from a continuous space using a GAN, there has been some previous work using modified GANs for text generation. One approach to avoid needing to back-propagate directly through the generator is to train the generator through reinforcement learning as was done in SeqGAN [7]. The generator outputs a sequence of text where at each step, the current state is the produced sentence so far, the available actions are the possible next words, and the rewards are provided by a discriminator. The discriminator is retrained periodically with a number of real and generated sentences, and this new discriminator is in turn used to provide new rewards to update the generator. Another approach used in a technique called Professor Forcing [8] is to back-propagate gradients through the continuous hidden states rather than the output words. In this work, a GAN model attempts to generate free-running RNN generators (where the output is fed back into the input) that are more similar to models trained with teacher forcing (where the true word is fed in at each time-step). Rather than discriminate between real and generated text, the discriminator attempts to discriminate between the hidden states of the free-running and teacher forcing RNN generators, thus encouraging the distribution of hidden states in the free-running model to be similar to those of the teacher forcing model.

In this paper, we explore using a GAN to generate text directly from a continuous space. Since this is a difficult problem, we explore pre-training the generator using a VAE, where the decoder from the VAE is used as a generator in the GAN. A similar model was used for image generation [9], although in that work the VAE and GAN were trained concurrently rather than using the VAE as pre-training for the generator. Previous work explored generating text from a continuous space using a VAE model [10]. However, since the VAE decoder in that work still uses sampling and teacher forcing, we are not able to use their VAE decoder model as a generator in our GAN.

3 Approach

Our approach is to train a GAN for text-generation from a continuous random space. We explore different models for the generators and discriminators including long-short-term memory (LSTM) RNNs, convolutional neural networks (CNNs), and fully-connected neural networks (FCNNs). We also try pre-training the generator by using a decoder from a VAE model.

3.1 Generative-Adversarial Network

The idea of a GAN is a two-player minimax game in which the discriminator attempts to classify an input x as real (with probability $D(x)$) or generated, and the generator produces generated data $G(z)$ from input noise z with the goal of fooling the discriminator into classifying incorrectly (Figure 1). If the generator can learn to mimic the distribution of the real data, it can fool the discriminator into predicting each class with 50% probability. The discriminator is trained to maximize the probability of predicting the correct labels for real and generated data (maximizing $\log D(x)$). We follow the suggestion of Goodfellow *et al.* for the generator to maximize the probability of the discriminator predicting incorrect labels ($\log D(G(z))$) rather than minimizing the probability of $\log(1 - D(G(z)))$ as in a minimax game to avoid diminishing generator gradients when the generator is poor at the start of training [3].

In a GAN model, loss is back-propagated through both the discriminator and generator so that the generator may learn from the discriminator's decisions. To allow for this back-propagation, we chose to use sequences of distributed word vectors as inputs to the discriminator rather than one-hot word encodings to avoid the sampling step in common language models. An alternative would have been to output probabilities over the vocabulary from the generator as is standard in language models and then input these probabilities to

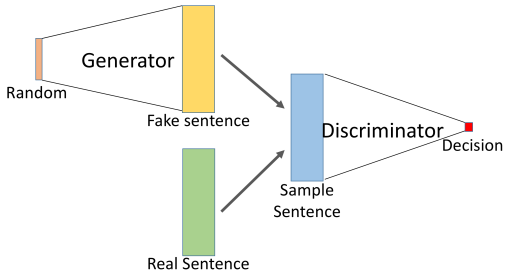


Figure 1: Representation of a GAN. The Generator produces text from a random vector and the Discriminator tries to differentiate the generated text from real text.

the discriminator before sampling, but it would be too easy for the discriminator to distinguish between probability distributions in the generated text and one-hot encodings in the real text.

Unfortunately, word-vector representations of words are not continuous, i.e. not every vector in the word-vector space represents a discrete word. GANs work very well to generate images since the pixel space is continuous; it is possible to transition from a red pixel to a blue pixel in a continuous manner as all the intermediate points between the color red and blue are valid colors. The same does not apply to words as there is no continuous way to transition from the word "red" to the word "blue" since not all the intermediate points are valid words. Since the outputs of the our generators are continuous word-vectors, to read the generated text, we pick the closest real word vectors to generator outputs. It is important to note that we do not sample the closest word vectors before we input the sentence into the discriminator as this would not allow for back-propagation.

3.2 Discriminator Models

For discriminators, we explore a number of neural-network binary classifier models. We try a Bag-of-Words (BoW) model, a CNN model, and an RNN model with LSTM cells. The BoW model takes the mean word vector of the sequence of word vectors, has a single fully-connected hidden layer with 100 neurons and a ReLU activation, a 50% dropout layer for regularization, and then a fully-connected layer with one output and a sigmoid activation. The CNN model we use is based on the setup of Yoon Kim [11] and consists of a series of convolutional layers with kernels as wide as the word vectors. Our kernels cover three words at a time, and our model has two convolutional layers with ReLU activations followed by a 50% dropout regularization layer and a fully-connected layer with one output sigmoid activation. The LSTM model takes the sequence of word vectors as its inputs and feeds the cell outputs to a fully-connected layer with one output and a sigmoid activation. The LSTM cells have a hidden dimension of 100. The discriminators are trained using binary cross-entropy loss as they output the probability the sequence is real text.

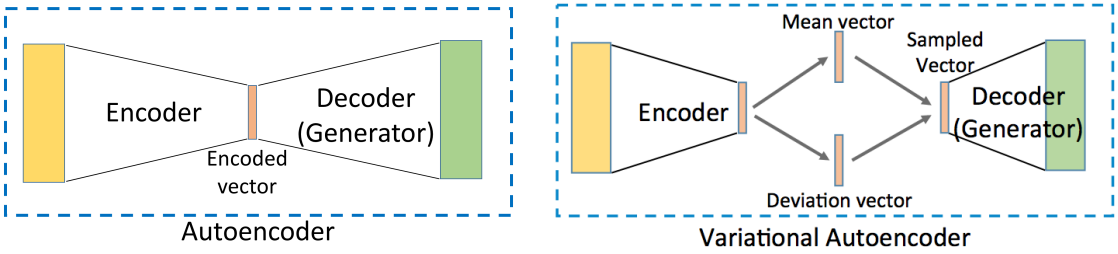
3.3 Generator Models

To improve generated GAN text, we explore starting the GAN with a pre-trained generator. In order to generate sentences from a random vector, we use the decoder from a pre-trained auto-encoder. An auto-encoder takes in a real sentence, encodes it into a much smaller latent representation, and then decodes the representation to reconstruct the input (Figure 2a).

Our auto-encoders take in sequences of word vectors (GloVe embeddings of real sentences [12]) and try to reconstruct them. We use word vectors of length 100 and sequences of length T. We use the mean squared error for reconstruction loss as we are interested in minimizing the error between the predicted word vectors y to the real word vectors x in the input:

$$\mathcal{L}_{MSE} = \frac{1}{100T} \sum_{t=1}^{T} \sum_{i=1}^{100} (x_t^{(i)} - y_t^{(i)})^2 \tag{1}$$

In most of our generation experiments, we let $T = 50$ since we are interested in improving generation of longer sequences. After some experimentation, we set the dimension H of the latent representation to 500.



(a) An auto-encoder.

(b) A variational auto-encoder.

Figure 2: Autoencoder architectures. Note that both the inputs and outputs are sequences of word-vectors. We pre-train our GAN generator as a decoder.

Since our goal is to use auto-encoders to generate text, we need to ensure that we can take the decoder part of the auto-encoder and use it with random inputs. With a classic auto-encoder, there is no guarantee on the distribution of the encoded representation, so we instead use VAEs to train the generators. The VAE model relies on the same principle as a standard auto-encoder, except that the latent representation is a normal distribution sampled using mean and standard deviation representations trained in the encoder (Figure 2b) [13].

If we call the mean vector h_μ and the deviation vector $h_{\log\sigma}$ (note that we have the encoder generate the logarithm of the deviation to make calculations simpler), the sampled vector will be as follows:

$$h_{sampled} = h_\mu + \mathcal{N}(0,1) \times \exp(h_{\log\sigma}) \tag{2}$$

We need to avoid having the VAE make $h_{\log\sigma}$ as small as possible and revert to the classic auto-encoder, so we add the KL divergence $\mathcal{L}_{KL}$ to our mean square error reconstruction loss:

$$\mathcal{L}_{KL} = \frac{1}{2H}\sum_{i=1}^{100} h_\mu^{(i)2} + \exp(h_{\log\sigma}^{(i)}) - h_{\log\sigma}^{(i)} - 1 \tag{3}$$

$$\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{KL} \tag{4}$$

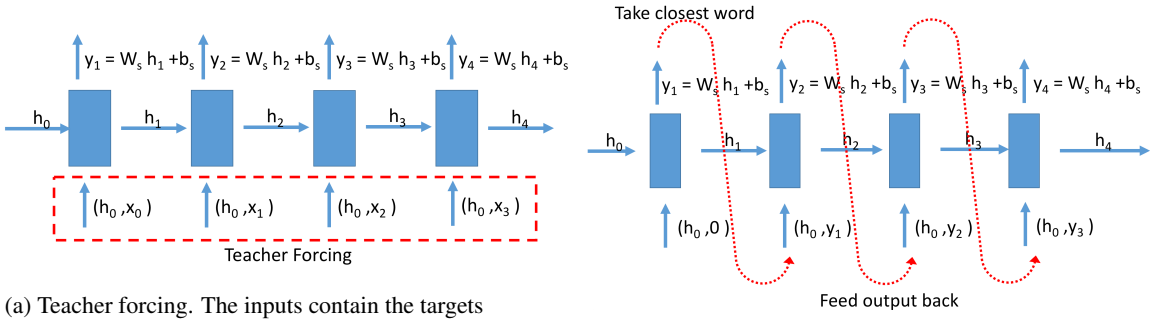
After training the VAE with this loss, the decoder should be able to generate random sequences from a random normal input since the KL term of the loss forces the latent vector to look like it is sampled from the normal distribution.

Note that this is a sequence-to-sequence problem just like in machine translation, except that in this case the input and the target are the same. However, there are additional challenges compared to standard sequence-to-sequence language models. Usually, when training an RNN, teacher forcing is used to pass the true previous word as input to the RNN cell (Figure 3a). Teacher forcing makes training easier and more stable, but since there is no true previous word in the GAN setup and we wish to back-propagate through our generator, we instead feed the last predicted word to the RNN cell (Figure 3b). Existing sequence-to-sequence models do this free-running mode at test time, however they adjust the output at the previous time-step before feeding it into the input by sampling from the probability distribution at the output. This adjustment step helps prevent errors from accumulating over time. However, it is not possible to sample in a GAN setup because this sampling step is not differentiable. Bowman *et al.* show that avoiding teacher forcing during training ("0% word-keep" in their paper) leads to much lower-quality sentences [10]. Additionally, attention, which usually helps sequence-to-sequence decoders focus on particular parts of the input during decoding, cannot be used since there is no true input to attend to in the GAN setup.

In this paper, we explore using both FCNNs and RNNs for the VAE encoder and decoder. Both the encoder and decoder have a single hidden layer with 500 neurons for our FCNN model and a single layer of LSTM cells also with 500 neurons for our RNN model.

4 Experiments

We experimented with our discriminator and generator models separately before combining them into a GAN setup. We built all models using TensorFlow [14] and Keras [15] libraries.



(a) Teacher forcing. The inputs contain the targets at the previous time-step.

(b) Free-running. The inputs contain the predictions at the previous time-step.

Figure 3: Teacher forcing versus free-running setup. We cannot use teacher forcing during generation due to the lack of target labels.

4.1 Dataset

Since we are interested in modeling longer-term text generation, we use the WikiText-2 Long Term Dependency Language Modeling Dataset as our source of human-written text [16]. This dataset consists of tokens extracted from the set of verified "Good" and "Featured" articles on Wikipedia. We chose this dataset over the more common Penn Treebank [17] (PTB) because it is over 2 times larger and because PTB is concerned with only a single sentence at a time (i.e. consecutive sentences are unrelated). We use 32,000 sequences from WikiText-2 and truncate each sequence to 50 tokens long.

Since we require the use of word vectors, we embed sequences using pre-trained GloVe word representations [12]. We use the publicly-available, 100-dimensional vectors trained on 6 billion tokens from Wikipedia and Gigaword 5. We do not update the embeddings during training. We map tokens that are not in the top 10,000 most frequent tokens and tokens that are not found in GloVe to the `<unk>` token, which we represent as the zero vector.

4.2 Discriminator

We evaluated discriminator models by training them to classify real sequences and synthetic sequences generated from a state-of-the-art language model. We first trained the published implementation of the language model from Zaremba *et al.* (large version) [18] on our WikiText-2 dataset for 55 epochs. The final perplexity of the language model is 49.92 on the training set and 89.28 on the validation set (compare to benchmarks of 37.87 and 82.62, respectively, for the PTB dataset). Samples from the trained language model are shown in Figure 8 in Appendix A. Like results from most state-of-the-art language models, sentences have decent grammatical structure but lack long-term semantic coherence.

We generated 32,000 synthetic sequences of length 50 by sampling from this trained language model. Then, we trained each discriminator on the binary classification task of separating the 32,000 sequences from WikiText from these 32,000 synthetic sequences. We trained each discriminator for 10 epochs and evaluated them on a held-out test set of 20% of the sequences. To evaluate our hypothesis that generated samples from current language models are worse for longer text sequences, we evaluated the performance of the discriminators by training them from scratch on subsequences of different lengths and assessing their classification accuracy (Figure 4). As expected, the performance of the discriminators improves as the subsequences increase in length. All discriminators were able to correctly classify at least 85% of the 50-token sequences, with the CNN discriminator slightly outperforming the BoW discriminator and the LSTM discriminator consistently performing best.

4.3 Generator

Our VAE models are able to reconstruct text when we overfit them on a small dataset (Figure 5). This result is encouraging as it shows that a VAE model is able to reproduce sentences despite the shift to outputting word vectors. We are also able to validate that the inclusion of the KL divergence in the loss function encourages the sequence encodings to form a unit normal distribution (Figure 6). However, when using the

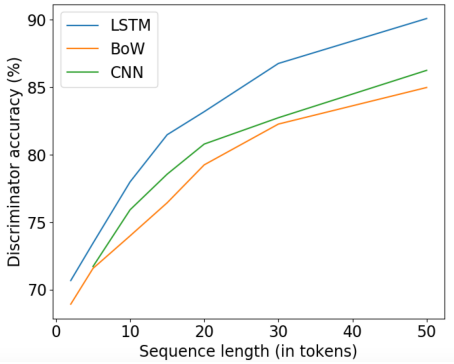
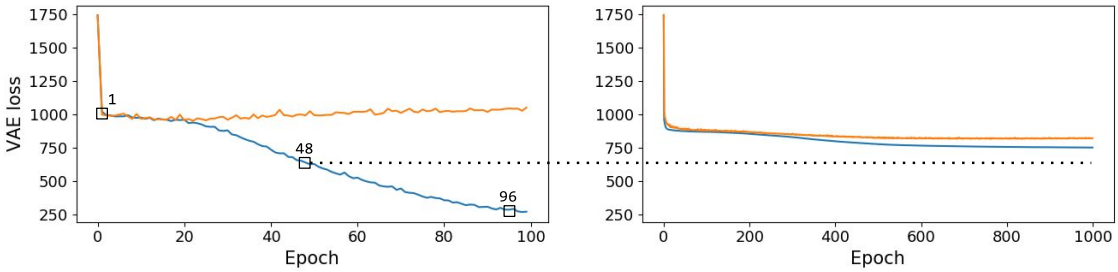


Figure 4: Test set accuracy of BoW, CNN, and LSTM RNN models on discriminating between real and generated text sequences of varying length after 10 epochs of training.




```
INPUT: failed due to the lack of electricity . with the exception of emergency radio <unk>
1.     same likewise well same well likewise instance instead well well addition well
48.    failed due to the lack of current . with the <unk> of emergency radio addition
96.    failed due to the lack of electricity . with the exception of emergency radio <unk>
```

Figure 5: Training and validation loss for a FCNN VAE model on a small dataset of 50 sequences (left) and on the full dataset (right). Below the graphs is the reconstruction of an example sentence at various epochs during training. On the full dataset, neither the training nor validation loss decreases to the threshold where sentences begin to be plausible English in the overfit model.

whole dataset, neither the training nor validation VAE loss decreases below the threshold needed to generate plausible English text (Figure 5).

For the LSTM VAE, we had little success reconstructing text using our differentiable decoder that outputs word vectors and feeds outputs to inputs without teacher forcing or sampling. To explore these issues, we tried overfitting an LSTM VAE model on 100 sequences using word probabilities versus word vectors and with and without teacher forcing. Using word probabilities allows for successful reconstruction, but removing teacher forcing means later words in the sequence are more likely to be incorrect as the errors accumulate over time.With word vectors, reconstruction using the same LSTM model is unsuccessful with or without teacher forcing (Figure 9 in Appendix A). Outputting words with the closest euclidian distance to the output word vector rather than sampling from a probability distribution appears to cause the same words to be output repeatedly. When using word probabilites along with sampling, if two output words are equally likely, the sampling step will choose one of them. However, when using word vectors, if two output words are equally likely, the model may minimize MSE by outputting the mean vector of the two words. In general, the nearest word to this vector may be unrelated to the two possible output words. The commonly-output words such as “though” and “well” fall near the mean of the word-vector space, supporting this hypothesis.

4.4 GAN

Although we were unable to train VAE models with sufficient generalization on the large dataset to enable generation of sequences that make semantic sense using the decoder portion of the VAE, we still tried incorporating these decoders/generators into GAN models to validate that the models could work in the GAN

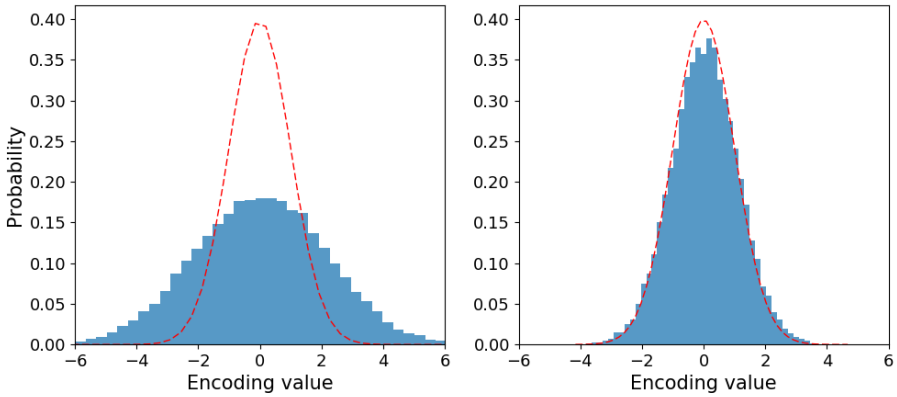


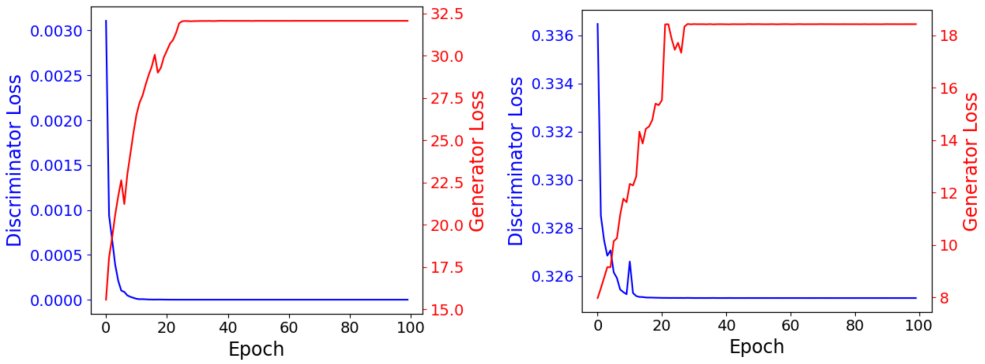
Figure 6: Distribution of the sequence encoding values on the training set using a vanilla FCNN autoencoder (left) and a FCNN VAE (right) after 500 epochs of training. In general, the distribution of the vanilla autoencoder sequence encodings is unknown. However, the distribution of the VAE encodings closely approaches the unit normal distribution (in red).

context with back-propagation through all steps. We used the keras-adversarial library [19] for the GAN framework. In all GAN experiments, we used the LSTM model as the discriminator as it performed best in our discriminator experiments. For the generator, we tried FCNN and LSTM models, both with and without pre-training these generators as decoders in VAE models. We trained all models for 100 epochs.

In all cases, the loss curves over the GAN training were similar to those in Figure 7a. The discriminator loss quickly drops to near-zero while the generator loss increases until reaching a plateau. We did not see a significant difference in loss curves for generators starting from scratch versus pre-trained on the VAE or for LSTM RNN generators versus FCNN models. Since the GAN loss is set up such that the generator is correct when the discriminator is wrong and vice versa and the discriminator task is much simpler, it is expected that the discriminator loss would drop while the generator loss rises early in training. However when GANs have been successful in image generation, the generator loss eventually decreases later in training rather than flattening out.

Since the GAN discriminator was classifying the generated text with extremely high probability, we experimented with a technique used in image GANs called one-sided label smoothing to encourage the discriminator to estimate soft probabilities rather than be overly confident in the classification [20]. In one-sided label smoothing, we keep the target labels for the generator at one for generated text and zero for real text, but decrease the discriminator target labels for the real samples from one to 0.9. Unfortunately, this lowered the peak generator loss but did not help it to decrease later in training and fool the discriminator (Figure 7b).

The best means of evaluating GANs is still an open question. In their original GAN paper, Goodfellow *et al.* evaluate samples from their generator by fitting a Gaussian Parzen window to the samples and reporting the log-likelihood of the test set data under this distribution [3]. However, Theis *et al.* argue against using this metric as they demonstrate that these Parzen window estimates are not a good estimate of the model's true log-likelihood for high-dimensional data and the metric often favors models with poor-quality samples [21]. Instead, we chose to evaluate our GAN models by training a discriminator from scratch on generated samples vs. real text samples. However, after only a single epoch, the discriminator was able to achieve 100% accuracy on the validation set for all generator models. While this is unfortunate as it does not allow us to compare our various models, it is not surprising as the text generated by the GANs at the end of training is very repetitive and still does not make grammatical or semantic sense (Figure 10 in Appendix A). Even if the sentences had better grammatical structure, the repetition between generated sequences allows the discriminator to simply learn the small subset of vocabulary used in these sequences to achieve 100% accuracy. This repetition indicates that the hidden states of the generator are holding the majority of the information used to generate the sequences and that the model is mostly ignoring the random input vector. Looking at the qualitative results in Figure 10, we also notice that the LSTM model has much more repetition within the sequences than the FCNN model, although it is able to decrease this repetition somewhat during training.



(a) Without one-sided label smoothing

(b) With one-sided label smoothing

Figure 7: Loss curves for an LSTM RNN generator and LSTM discriminator during GAN training. While one-sided label smoothing increases discriminator loss and decreases generator loss magnitudes, it does not help the generator to improve significantly as training progresses.

5 Conclusion

In this paper we explored using a GAN for text generation. We experimented with a variety of models for generators and discriminators including BoW, FCNN, CNN, and LSTM RNNs. All discriminator models were successfully able to distinguish between real and synthetic text sequences with high accuracy even when the synthetic text was generated with state-of-the-art text generation models. Creating an adequate generator that allows for back-propagation proved to be the more challenging task as it requires the following changes to a standard language model or sequence-to-sequence setup:

- Output words cannot be sampled from a probability distribution as this sampling step is not differentiable. Since using probability distributions without sampling would mean the discriminator would only need to discriminate between one-hot vectors and spread probability distributions (a task that is too simple), we output predicted word vectors instead of probability distributions.
- In RNN generators, teacher forcing cannot be used during GAN training since there is no ground-truth, so the outputs must be passed as inputs to the next timestep.
- In RNN generators trained as decoders, attention cannot be used.

These changes lead to a number of issues:

- Since the embedding function from words to word vectors is not continuous, small update steps during training may output the same word or a completely different, potentially unrelated, word.
- Since we cannot snap word vectors to the nearest word before inputting to the next timestep or passing to the discriminator, but we must snap to the nearest word to output generated text, the generated text may not match what is seen by the discriminator.
- Without teacher training and word sampling/snapping, errors propagate over time.

Due to these issues, we were unable to create an improved model for text generation using a GAN framework. Future work could attempt to mitigate the errors we observed from removing the sampling step by sampling in the forward pass but using an approximation of the backwards pass such as the straight-through estimator, which computes the derivative with respect to the expected loss rather than with respect to the specific outcome of the sampling step [22], or the straight-through Gumbel-Softmax estimator, which performs well on categorical outputs [23]. To mitigate errors from removing teacher training, future work could use Professor Forcing to pre-train a generator that is similar to one trained with teacher forcing, and then use this pre-trained generator in the GAN. Alternatively, since FCNN and de-convolutional neural network generators do not have the time-step error propagation issues, improving such generators may be another potential direction for future work. Since we were able to overfit a FCNN VAE model on a small dataset but not on the full dataset, this model may improve with more hidden layers and longer training time to eventually produce a decoder capable of generating text from a continuous space in a fully differentiable manner.

References

[1] Ankit Kumar, Ozan Irsoy, Jonathan Su, James Bradbury, Robert English, Brian Pierce, Peter Ondruska, Ishaan Gulrajani, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, 2015.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *arXiv:1406.2661 [stat.ML]*, 2014.

[4] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *arXiv:1606.03498 [cs.LG]*, 2016.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434 [cs.LG]*, 2016.

[6] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv:1605.05396 [cs.NE]*, 2016.

[7] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *arXiv:1609.05473v5 [cs.LG]*, 2016.

[8] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *arXiv:1610.09038v1 [stat.ML]*, 2016.

[9] Anders Larsen, Soren Sonderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv:1512.09300v2 [cs.LG]*, 2016.

[10] Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv:1511.06349v4 [cs.LG]*, 2016.

[11] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882v2 [cs.CL]*, 2014.

[12] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv:1312.6114 [stat.ML]*, 2013.

[14] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[15] François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[16] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv:1609.07843 [cs.CL]*, 2016.

[17] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The Penn Treebank: An overview, 2003.

[18] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv:1409.2329v5 [cs.NE]*, 2015.

[19] Ilya Mnill. keras-adversarial. `https://github.com/bstriner/keras-adversarial`, 2016.

[20] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv:1701.00160v3 [cs.LG]*, 2017.

[21] Lucas Theis, Aaron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv:1511.01844v3 [stat.ML]*, 2016.

[22] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432 [cs.LG]*, 2013.

[23] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv:1611.01144v1 [stat.ML]*, 2016.

A Text Generation Examples

repeated gunnery colour means to take the change . anonymous said , " what [i don 't want about the aberdeen congregation] of high quality , and which i go off in time to my low growing average , " give the <unk> of people .

leading his alleged successful funding for 10 months , helping an ongoing call for a recent more understanding . <unk> and his wife <unk> ( ) and the second commissioned in rome . two of her experiences are now organised by jeremy <unk> and some <unk> enhance her to and

at a record point and was completed to an element in the state of wales (<unk> in yellow) . the experience after the battle date straight for the story , <unk> and the anti-<unk> penalty attempted to reject their <unk> through a policy of proposals that

Figure 8: Example sequences from a language model from Zaremba *et al.*[18] trained on the WikiText-2 dataset [16]. Although the sentences follow general English grammatical structure, they lack long-term semantic coherence.

Outputting probability densities

Input text: the revised 2010 forest plan recognized the need to develop plans to manage <unk> at the <unk> urban <unk>, use <unk> fire as a tool to manage ecosystem health , and meet air quality requirements set by the clean air act . the forest operates a fire management plan

Reconstructed text with teacher forcing: the revised 2010 forest plan recognized <unk> need <unk> develop plans of manage the manage <unk> album urban <unk> , the <unk> , , the tool , the ecosystem , . and meet air quality requirements , the the clean air act and the forest operates . fire management plan

Reconstructed text feeding output to inputs at next timestep: the revised 2010 forest plan recognized <unk> , the ceo of the <unk> urban <unk> , the the generation of the forest plan recognized the dna , the the the forest plan recognized the dna , the the generation of the forest plan , the forest , was <unk> plan

Outputting word vectors

Input text: from other <unk> by a long , wide , <unk> process of the <unk> , teeth in the <unk> with a very large <unk> , and large ridges on the tooth...

Reconstructed text with teacher forcing: though same though well same well same well well same . well though same well though well same same addition...

Reconstructed text feeding output to inputs at next time-step: though though though though though though though though though though though though though though though...

Figure 9: Example reconstruction sequences from an LSTM RNN VAE model. The sequences demonstrate the varying levels of reconstruction when outputting word probabilities versus word vectors and when using teacher training versus feeding outputs from one time-step as inputs at the next.

LSTM RNN

After 1st epoch: `theater theater theater theater theater theatre theatre...`
`relations relations political political role role role role role role role...`

After last epoch: `with france quercus pinus atherstone roadside roadside roadside an evidence archeological...`
`it cockpit tearing atherstone atherstone roadside roadside roadside an evidence archeological...`

FCNN

After 1st epoch: `every applied mumbai board arrest to drivers driver driver persons belfast bradstreet village jobs taxi dodge...`
`every applied mumbai share arrest to drivers driver driver persons belfast bradstreet village jobs town...`

After last epoch: `asteroid density magnitude district magnitude county county storms northern midfielder frequency goals...`
`asteroid density magnitude district magnitude county county storms killed injuries frequency goals...`

Figure 10: Example sequences from LSTM RNN and FCNN generators trained using a GAN.