CS230

Unsupervised Text Generation Using Generative Adversarial Networks

Andrew E. Freeman
Department of Statistics
Stanford University
aefree@stanford.edu

Anna C. Shors
Department of Statistics
Stanford University
ashors@stanford.edu

Anastasios Sapalidis
Department of Statistics
Stanford University
tsap@stanford.edu

Abstract

Despite their success in other fields, Generative Adversarial Networks (GANs) have seen little success in the area of text generation due to the need to "pick" words to generate at each timestep. This discrete "picking" process is non-differentiable and makes it difficult to pass gradients from the discriminator to the generator during training. In this paper, we explore a method for using GANs for text generation, which consists of training a generator to output sentence encodings, and using a pretrained RNN decoder to decode the encodings into human-readable sentences. We compare some early stage results against a vanilla RNN baseline. We also demonstrate how this method can be expanded to generate coherent paragraphs.

1 Introduction

Generative Adversarial Networks (GANs) have risen in popularity in the field of computer vision due to their ability to generate novel realitic images. However, GANs have gained little traction in the field of text generation, as generating text requires models to choose from a set of vocabulary words a single word to generate at each timestep. The function that maps from the space of vocabulary words to a single word is non-differentiable, which poses a challenge during GAN training when the gradients need to pass from the discriminator to the generator. Recent research on using GANs for text generation has focused on using reinforcement learning and policy gradient methods to train the generator (19) (18). An alternative approach consists of training a generator to produce sentence encodings and using a pretrained RNN decoder to decode the encodings into human readable sentences (5). In this paper, we explore this method for text generation.

This technique consists pretraining an autoencoder, then training a GAN to generate sentence encodings that are indistinguishable from the encodings generated by the autoencoder from real text. When training the GAN, we use the autoencoder's encoder to generate "real" sentence encodings. These real encodings, as well as the encodings generated by the generator, are fed as input into the discriminator, whose goal is to distinguish the real encodings from the fake ones. Once the GAN training is complete, the generator can be used to generate sentence encodings which can then be decoded using the autoencoder's pretrained decoder.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Dataset and Features

We use the Wikipedia movie plot dataset from Kaggle (10) to train both the autoencoder and the GAN. This dataset contains approximately 35,000 short movie summaries, comprising over 600,000 sentences, scraped from Wikipedia. For training the auotencoder and GAN, we used standalone sentences rather than maintaining a structure of sequential sentences to simplify our initial model. We removed any parenthetical comments in the sentences in order to simplify the sentence structure. We then verified sentence splits were valid (i.e. titles such as Mr. or Mrs. do not indicate the end of a sentence) and excluded any sentences of length less than 5 characters to account for cleaning errors. From this cleaned set, we split our data into train, dev, and test sets. At train time, we removed any non-alphabetic characters aside from trailing punctuation and commas to further simplify the model. Finally, we used the Keras Tokenizer to tokenize the sentences.

3 Methods

We take inspiration from the methods provided by Donahue and Rumshisky (5). We first pre-train a simple RNN autoencoder on our movie summaries dataset. We train the encoder to generate a dense context vector from a given sentence and train the decoder to recover the original sentence from this context vector. Because we need the encoder and decoder to work independently of one another, we do not use attention for this model. We use categorical crossentropy loss for each word generated by the decoder and define the loss function for a given sentence to be the average crossentropy of each word in the generated output. We train our autoencoder using teacher forcing, so that the decoder is given the correct word from timestep t as input when generating the word at timestep $t + 1$.

Once the autoencoder is trained, we can train our GAN. Following the methods proposed in Donahue and Rumshisky, we use a Wasserstein GAN, rather than a standard GAN, for more stable training (2). Both the GAN generator and discriminator (called the "critic" in the case of WGANs) consist of simple series of ResNet blocks. The GAN critic takes as input a sentence encoding (either a real encoding generated by the encoder, or a fake encoding generated by the generator) and outputs a real number that describes the "level of realness" of the encoding. The goal of the critic is to output very low numbers when given fake sentence encodings and output very high numbers when given real sentence encodings. The critic cost function is described in Equation 1, where f_w refers to the critic, and g refers to the generator.

$$\mathcal{J}^{(D)} = -\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) + \frac{1}{m}\sum_{i=1}^{m} f_w(g(z^{(i)})). \tag{1}$$

The goal of the generator is to produce sentence encodings that are indistinguishable from real sentence encodings. The generator is thus working to make the critic predict its encodings as real. The generator loss is described in Equation 2.

$$\mathcal{J}^{(D)} = -\frac{1}{m}\sum_{i=1}^{m} f_w(g(z^{(i)})). \tag{2}$$

After the GAN has been trained, we pass the encodings generated by the generator into the decoder to create human-readable sentences. [1]

We can expand this idea to generate a series of coherent sentences as follows. Instead of training a generator to output sentences unconditionally, we can encourage the generator to output a sentence conditioned on the previously generated context vectors. Thus, at timestep t, the generator takes as input the series of $t - 1$ previously generated sentence vectors. Once the generator has finished generating a full sequence of sentences (signaled by a special "end of sequence" token), the series of context vectors is fed into the critic. The critic thus takes as input a series of sentence encodings, where each series is either from a real sequence of sentences or from the generator, and predicts whether the sequence of sentences is real or fake. The generator and discriminator loss functions are identical to those in Equations 1 and 2, except $g(z^{(i)})$ now uses a recurrent structure to output a series of sentence encodings, and f_w now takes as input series of sentence vectors, rather than a single sentence vector.

[1] Model code: github.com/tassossapalidis/latextgan

4 Results and Discussion

We train the autoencoder on a random sample of 150,000 sentences from our dataset of 600,000. This autoencoder consists of a bidirectional GRU encoder and a standard GRU decoder. We use 512 hidden units for the encoder and concatenate the forward and backward encoder hidden states to create a 1024-dimensional vector that is used as input into the decoder. We train using the Adam optimizer with a learning rate of 0.001. Table 1 summarizes the performance of this autoencoder on 2,000 sentences in the validation set.

Autoencoder Performance	
BLEU Metric	Score
BLEU-1	78.7
BLEU-2	70.2
BLEU-3	63.6
BLEU-4	58.6

Table 1: Averaged BLEU scores for 2,000 encoded/decoded sentences from the validation set

We use a GAN with a generator and critic both consisting of a series of 5 ResNet blocks with a block dimension of 1024. Following the results from (8), we add a gradient penalty to the discriminator's loss function to enforce that the function learned by the generator be 1-Lipshitz and train using Adam with a learning rate of 0.0001, $\beta_1 = 0.5$ and $\beta_2 = 0.9$. We include a sample of sentences created by the generator in Table 2. In addition, we include a sample of sentences generated by the baseline, which is a simple word-level RNN with attention. [2]

RNN Baseline	GAN
He is then attacked by indians, and they attempt to escape.	Wolfgang's break the <unk> for copper meets father by surrendering.
8 years later, the young boy, is a simple and western person.	Speed <unk>, listening the film with todd as <unk> follow.
Jerry helps quacker and sally, but are getting arrested.	A murderous <unk>, both honor the resident does die.

Table 2: Sentences generated from a simple word-level RNN with attention (left) vs. sentences generated by our GAN (right)

Currently, the GAN is able to produce somewhat coherent sentences. However, they lag in quality relative to the RNN baseline. BLEU score calculations for both models are included in Table 3.

RNN Baseline vs. GAN Performance		
BLEU Metric	RNN Baseline Score	GAN Score
BLEU-1	94.6	76.5
BLEU-2	84.4	56.1
BLEU-3	66.9	22.1
BLEU-4	40.5	1.1

Table 3: Averaged BLEU scores for 100 generated sentences from the RNN Baseline and our GAN. Each sentence was evaluated against the corpus comprising the training set.

In the coming weeks, we plan to continue tuning the hyperparameters of our GAN to improve the quality of the generated sentences. We also hope to continue developing our autoencoder (which can be done independently of the GAN), to create higher quality decodings of the encodings generated by the decoder. If, after tuning our hyperparameters, we are able to generate fluent English sentences, we hope to begin exploring the recurrent architecture described in section 3.

[2] Baseline code: https://github.com/minimaxir/textgenrnn

References

[1] AKMAL HAIDAR, M., REZAGHOLIZADEH, M., DO-OMRI, A., AND RASHID, A. Latent Code and Text-based Generative Adversarial Networks for Soft-text Generation. *arXiv e-prints* (Apr. 2019), arXiv:1904.07293.

[2] ARJOVSKY, M., CHINTALA, S., AND BOTTOU, L. Wasserstein GAN. *arXiv e-prints* (Jan. 2017), arXiv:1701.07875.

[3] BAMMAN, D., O'CONNOR, B., AND SMITH, N. A. Learning Latent Personas of Film Characters. *ACL 2013, Sofia, Bulgaria* (Aug. 2013).

[4] CHINTAPALLI, K. Generative adversarial networks for text generation — part 3: non-rl methods, Jun 2019.

[5] DONAHUE, D., AND RUMSHISKY, A. Adversarial Text Generation Without Reinforcement Learning. *arXiv e-prints* (Oct. 2018), arXiv:1810.06640.

[6] FAN, A., LEWIS, M., AND DAUPHIN, Y. Hierarchical Neural Story Generation. *arXiv e-prints* (May 2018), arXiv:1805.04833.

[7] FANG, L., ZENG, T., LIU, C., BO, L., DONG, W., AND CHEN, C. Outline to Story: Fine-grained Controllable Story Generation from Cascaded Events. *arXiv e-prints* (Jan. 2021), arXiv:2101.00822.

[8] GULRAJANI, I., AHMED, F., ARJOVSKY, M., DUMOULIN, V., AND COURVILLE, A. Improved Training of Wasserstein GANs. *arXiv e-prints* (Mar. 2017), arXiv:1704.00028.

[9] HERRERA-GONZÁLEZ, B., GELBUKH, A., AND CALVO, H. Automatic Story Generation: State of the Art and Recent Trends. *Advances in Computational Intelligence 19th Mexican International Conference on Artificial Intelligence* (Oct. 2020).

[10] JUSTINR. Wikipedia movie plots. `kaggle.com/jrobischon/wikipedia-movie-plots/metadata`, 10 2018.

[11] LI, Y., GAN, Z., SHEN, Y., LIU, J., CHENG, Y., WU, Y., CARIN, L., CARLSON, D., AND GAO, J. StoryGAN: A Sequential Conditional GAN for Story Visualization. *arXiv e-prints* (Dec. 2018), arXiv:1812.02784.

[12] LUONG, M.-T., PHAM, H., AND MANNING, C. D. Effective Approaches to Attention-based Neural Machine Translation. *arXiv e-prints* (Aug. 2015), arXiv:1508.04025.

[13] MINIMAXIR. textgenrnn. `github.com/minimaxir/textgenrnn`, 2020.

[14] MIRZA, M., AND OSINDERO, S. Conditional Generative Adversarial Nets. *arXiv e-prints* (Nov. 2014), arXiv:1411.1784.

[15] PRANAVPSV. Gpt2 genre-based story generator. `github.com/pranavpsv/Genre-Based-Story-Generator`, 2021.

[16] SAEED, A., ILIĆ, S., AND ZANGERLE, E. Creative GANs for generating poems, lyrics, and metaphors. *arXiv e-prints* (Sept. 2019), arXiv:1909.09534.

[17] SHREYDESAI. Implementation of adversarial text generation without reinforcement learning. `github.com/shreydesai/latext-gan`, 2019.

[18] SUTTON, R. S., MCALLESTER, D., SINGH, S., AND MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems* (2000), S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press.

[19] YU, L., ZHANG, W., WANG, J., AND YU, Y. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *arXiv e-prints* (Sept. 2016), arXiv:1609.05473.

[20] ZHU, Y., LU, S., ZHENG, L., GUO, J., ZHANG, W., WANG, J., AND YU, Y. Texygen: A Benchmarking Platform for Text Generation Models. *arXiv e-prints* (Feb. 2018), arXiv:1802.01886.