CS230

Generating Yelp Reviews with GANs

Abhi Kulgod
Department of Computer Science
Stanford University
akulgod@stanford.edu

Veeral Patel
Department of Computer Science
Stanford University
veeral@stanford.edu

Jayen Ram
Department of Computer Science
Stanford University
jjram@stanford.edu

Abstract

Supervised methods such as n-gram based Language Models and Recurrent Neural Networks (RNNs) have often been applied to text generation tasks. For example, RNN models are trained to predict the most likely next word given a previous word and sequences are then generated by feeding these predictions back into the network and producing tokens one by one. This approach has been shown to have a few major drawbacks – namely, the problem of *exposure bias*[1]. When generating tokens at test time, the inputs are drawn from the distribution of the model itself, rather than the distribution of the training data, creating errors that can quickly accumulate along the way. As a result, models that directly optimize for the task of text generation have become increasingly popular. Generative Adversarial Networks (GANs), for example, have proven to be very successful in generating synthetic text for political speeches and Chinese poems[2]. In this paper, we take a similar GAN approach with a CNN discriminator and RNN generator, apply it to Yelp review data, and report our generated sentences as being comparable to a traditional LSTM RNN.

Code: https://github.com/veeralpatel/cs230-final-project

1 Introduction

Over 145 million reviews have been written on Yelp.com since its inception in 2005[3]. With its rise in popularity, many consumers now turn to it as the de facto site to help choose places like restaurants, barbershops, and hotels. People trust Yelp reviews because they are seemingly written by real people, but after learning about recent issues regarding the prevalence of fake reviews written by bots[4], we were interested in learning more about how to distinguish between real and fake reviews and the process behind generating such fake reviews. We think this problem is interesting because there is a monetary incentive for both good and bad actors to detect and generate bad reviews. A good actor like Yelp would like to minimize the number of fake reviews to preserve the integrity of their review site and retain users. On the other hand, a bad actor like a new restaurant business might generate fake reviews to show a false sense of quality and obtain more customers. A generative adversarial network helps up iteratively train models to handle both of these problems, but we focused our efforts on improving the generator since it's a more complex problem.

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

2 Related work

Prior research has been done in generating text without GANs, with GANs[5], and the task of specifically generating Yelp reviews.

Bowman et al. introduced an RNN-based variational autoencoder generative model that incorporates distributed latent representations of entire sentences[6]. This allowed them to better holistically model style, topic, and high-level syntactic features. Their results showed diverse, well-formed sentences that were effective at imputing missing words.

For generative adversarial network approaches to generating text, Yu et al. used a reinforcement learning approach to model the generator and performed policy gradient updates on the generator[2]. The reward was provided by the discriminator being judged on a complete sequence established via Monte Carlo rollouts. However, a majority of their experiments were on synthetic data generated by their own LSTM and their real data came from a single data source like political speeches or Chinese poems, so it was difficult to directly compare our results with theirs.

William et al. used a conditional GAN to fill in missing text conditioned on the surrounding context[7]. Their architecture consisted of an LSTM for both the generator and discriminator and they showed that it performed better qualitatively and quantitatively than an MLE approach. While infilling was an approach we could have taken to produce fake reviews, we worried that the reviews generated would be too similar syntactically to existing reviews and plagiarism detection software employed by such review sites could easily detect such copies.

Professor Forcing is a variation of training an RNN with teacher forcing (where ground-truth samples are fed back into the model to be conditioned on for the prediction of later outputs) by using a discriminator to discriminate between hidden states from sampling mode and teacher forcing mode[8]. This is different from traditional GAN classifiers that discriminate between real and generated samples. Because the discriminator only looks at the hidden states, gradients can be directly passed to the generator so that hidden state dynamics of sampling and teacher forcing mode are similar. Results showed that Professor Forcing acted as a regularizer and improved test likelihood on character level Penn Treebank.

For the Yelp specific work, Yao et al. used an LSTM RNN for generating new text in existing Yelp reviews after identifying entities that could be replaced as identified by WordNet[9] lexical similarity[10]. For example, a review about the delicious sushi and ramen at a Japanese restaurant would be replaced with a review about the delicious pizza and pasta at an Italian restaurant. In addition, an important parameter they added called "temperature" allowed them to control the "novelty" of their generated text compared to training data. Through these methods, Yao et al. were able to generate highly convincing fake reviews as measured by their user surveys.

3 Dataset and Features

The dataset used in pre-training and adversarial training was sourced from Professor Shebuti Rayan of Stony Brook University[11]; it contains 427,000 reviews from Chicago and NYC covering restaurants and hotels. In addition, the dataset is labelled real or marked fake (by Yelp), where 11% are marked as fake by the Yelp spam filter. In our pre-training for the generator, we use 60,000 training examples and leave the rest for adversarial training. In our pre-training for the discriminator, we train with 10,000 training examples as generated by the previously trained generator. In our adversarial training, we further train the discriminator with 50% real from the Shebuti dataset and 50% generated examples from the generator.

For the sake of ease in generation, we limit the input and output sentences to be of size 30, and all reviews in the dataset are clipped as such. In addition, we pre-process our sentences by passing them into a word2vec model from Tensorflow.

The dataset involved in word2vec training was derived from a dataset of Yelp reviews from Kaggle[12]. The dataset contains 5 million reviews about 174,000 different businesses in 11 metropolitan areas; the dataset was not separated based on stars given by review or topic of the review. In our word2vec training, due to memory constraints, 200,000 of these reviews are used. We experimented heavily with our hyperparamater of embedding size, and settled on a value of 100 for the size of our embedded vectors.

4 Methods

In the adversarial network, we have a generator, G, that produces samples to be evaluated by a discriminator, D, whose output is then used to update and improve G. While some GANs begin this process (adversarial training) with G and D initialized randomly, pre-training both networks has been shown to greatly improve their success[2]. The details of these networks' architectures and loss functions for both pre- and adversarial training are described below.

4.1 Discriminator

Given a mix of real and synthetic samples, D's output is the probability that a given sample is real. We use a Convolutional Neural Network (CNN), whose architecture is based on previous work by Yu. The network has a single convolutional layer with parallel filters of lengths $[1, 2, 3, 4, 5, 8, 10, 15, 20]$.

As shown in the figure to the right, this is analogous to having separate filters for unigrams, bigrams, etc. Each of these filters are followed by ReLU activation and then a max pooling layer. Finally, the parallel filter outputs are stacked and passed through two fully connected layers to produce an output z. D's final prediction is given by sigmoid activation, $\hat{y} = \sigma(z)$. We can then optimize D's parameters, ϕ, to minimize the cross-entropy loss with respect to the input x and their true labels, y.

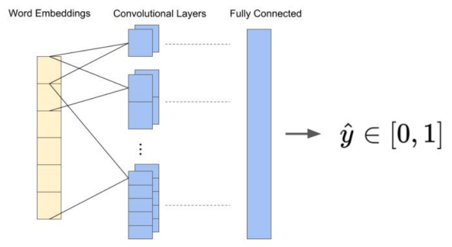


Figure 1: Parallel filters of D

Since we know we want D to correctly label real data with 1 and fake data with 0, we can write the loss in terms of the real and fake samples. Given m_d real samples drawn from the Yelp dataset, $\{x^{(1)}, ..., x^{(m_d)}\}$, and m_g fake examples produced by the generator, $\{z^{(1)}, ..., z^{(m_g)}\}$, the goal is to minimize the cost defined below:

$$J(\phi) = -\frac{1}{m_d}\sum_{i=1}^{m_d} \log D_\phi(x^{(i)}) - \frac{1}{m_g}\sum_{i=1}^{m_g} \log(1 - D_\phi(z^{(i)})) \tag{1}$$

We pre-train D using samples drawn from the real data and from G after its respective pretraining. The same method is used during adversarial training, as updates are being made to G.

4.2 Generator

We begin by training a generative model on real-world sequences of Yelp reviews. Given the tokens up to a point in a sequence, its task is to predict the most likely next token. We do this using an RNN with a long short-term memory cell (LSTM). The LSTM cell iterates through an input sequence of word-embeddings, $x_1, x_2, ..., x_T$, to produce hidden states, $a_1, a_2, ..., a_T$. These hidden states are then passed through a fully connected layer to produce the output layer, z. A final softmax activation is applied to z to produce a distribution over all of the possible next tokens, $\hat{y}$. We refer to the parameters of the LSTM cell and output layers jointly as θ. Thus we can define the output of G, parameterized by θ, at any "step" of the sequence, t:

$$\hat{y}_t = p(y_t|x_1, ..., x_{t-1}) = G_\theta(y_t|x_1, ..., x_{t-1}) = softmax(z)$$

In traditional GAN applications, adversarial training would involve generating samples according to G_θ, evaluating them according to D_ϕ then deriving updates for θ in order to confuse the discriminator. In other words, each step would attempt to maximize equation (1) by back-propagating through ϕ

and the samples to find the derivative of $J(\phi)$ w.r.t θ. Unfortunately, the output space of G_θ (word embeddings) is highly discontinuous and thus non-differentiable.

Instead, we model G_θ as a Reinforcement Learning (RL) agent and estimate its updates during adversarial training using Policy Gradient algorithms, as explained in the next section.

4.3 Policy Gradient Algorithm

Estimating generator updates using Policy Gradients has previously been applied to text generation by Yu et al[2]. The policy in this RL approach is represented by G_θ, whose task is to produce sequences of length T, $Y_{1:T}$, in order to receive rewards. The state at a given time step t is represented by the tokens produced so far, $Y_{1:t-1}$, and potential actions are represented by potential tokens to add to the sequence. More formally,

$$\pi_\theta(s = Y_{1:t-1}, a = y_t) = G_\theta(y_t|Y_{1:t-1})$$

The objective for G is to maximize the expected value of its reward, which is the probability of a completed sequence being real as determined by the discriminator, $R_T = D_\phi(Y_{1:T})$. This gives us the loss function for adversarial training as[2]:

$$J(\theta) = -\mathbb{E}[R_T|s_0] = -\sum_{y_1\in\gamma} G_\theta(y_1|s_0) \cdot Q_D^G(y_1, s_0) \tag{2}$$

Where Q_D^G is the expected reward for completed sequence from a given state (since there is no reward for a partial sequence). Maximizing equation (2) yields the following gradient[2], which aligns closely with updates in the REINFORCE algorithm:

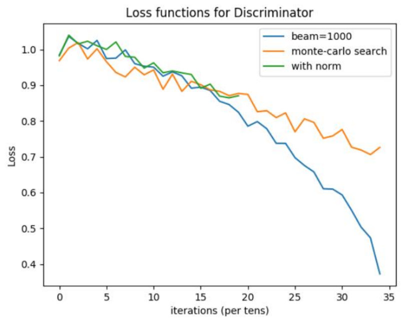
$$\nabla_\theta J(\theta) \approx \sum_{t=1}^{T} \mathbb{E}_{y_t\sim G_\theta}[\nabla_\theta \log G_\theta(y_t|Y_{1:t-1}) \cdot Q_D^G(y_t, Y_{1:t-1})] \tag{3}$$

The expectation in equation (3) can be estimated through various sampling methods. We explore both Monte Carlo (MCS) and Beam Search (BS). At each time step in the generation process, G_θ outputs a probability distribution over the vocabulary for how likely words are to come next. MCS samples from this entire distribution, while BS would sample from the top-k most likely next-words. We hypothesize an exploration-exploitation tradeoff between these search policies; BS may help G learn to prioritize the best words, at the expense of missing or "forgetting" potentially high reward tokens. During testing, we compare MCS with BS using an annealed schedule for beam width – starting out with a beam width covering the entire vocabulary, and decreasing linearly to small target width.

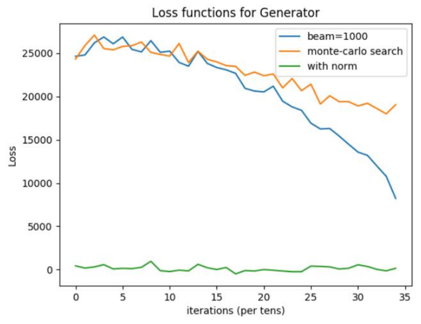
Finally, we explored methods of stabilizing gradient updates during adversarial training. Policy Gradients with Advantage have been shown to improve stability and training by subtracting a baseline value from rewards given to actions[13]. In our case, no state has a true "value" because rewards are not given until sequences are completed. Instead, we simply normalize the rewards, as a means of estimating advantage[14].

5 Experiments, Results, and Discussion

For our tests, we experimented with different hyperparameters and rollout policies. In the table below, "Random" represents the outputted text before training our generator, "LSTM" represents the outputted text before starting adversarial training, GAN - BS represents using Beam Search (beam width of 1000) for selecting next tokens, GAN - BS represents using Monte Carlo Search for selecting next tokens, and GAN - NORM represents normalizing our discriminator rewards that are passed to the generator. In all these experiments, we stayed with a consistent learning rate of 1e-2 for our generator and 1e-5 for our discriminator. We ran adversarial training for 35 total epochs since we had already done pretraining before (more hyperparameter specifics can be found in the gan.py file of the project repository). The n-gram score is calculated similarly to the MaskGAN[7] paper and represents the number of unique n-grams produced by the generator that occur in the generator training data. For example, an n-gram score of 1.0 represents that the generator perfectly represents the training data and outputs the same combination of words. In addition, we list the top 10 sentences (out of 50 total sentences) as qualitatively evaluated by a group of 12 people.



(a) Discriminator Training



(b) Generator Training

n-gram Scores			
Experiment	n = 2	n = 3	n = 4
Random	0.03977	0.00001	0.0000
LSTM	0.37874	0.17603	0.06958
GAN - BS	0.26772	0.11074	0.03579
GAN - MCS	0.29007	0.13991	0.05451
GAN - NORM	0.35383	0.15388	0.05518

Top Sentences Generated by GAN - NORM
the atmosphere is nice for a date lovers, there a bit of wait for <UNK> cocktails, and <UNK> cute and affordable spot for the price.
i've tried the bbq pork tacos but the best chicken gyro i've had (and if you who have to mention for lamb).
wonderful italian food - inexpensive, understated, yet delicious pizza. loved how it is perfection.
authentic puerto rican hole located with a trendy <UNK> joint. dark, nice atmosphere as well.
my husband and i came for a sunday brunch here. they've got such <UNK> from the most simple brunch order.
this delicious good place for a date and drinks. i had a great experience and pretty happy hosts for sushi. great vibe, enough wait.
amazing pasta and a large, tasty champagne. the service was excellent, well priced really benedict and jambalaya was tasty.
i'm going to be honest, i wasn't blown away by a bit overbearing all of the little reviews led me to before the <UNK> come for all things they.
amazing and cozy too. good wine list, service, and fast service, but not impressed. we were slightly <UNK> this place is very good for delicious dinner.
we thought spinach appetizer was good. had found at least 8 months since it opened, there were not people in a small window.

As shown above, while the generated sentences are comprehensible, they often contain incorrect grammar, contradict themselves, and contain multiple <UNK> tokens. The GAN with the normalized rewards performed the best, but as seen in the graph of our generator loss function, training was highly unstable with the modified rewards compared to the other methods. Overall, because of such instability of our GAN, more research and hyperparameter tuning needs to be done before concluding that using adversarial training leads to better LSTM RNN performance.

6 Conclusion and Future Work

With respect to our model, we found that just training the LSTM provided the highest n-gram score, but found that a lot of the sentences qualitatively seemed to link together common bigrams and trigrams. When training the GAN, we found that Monte Carlo Search with reward normalization provided the most stable network.

In our findings, the GAN's training on a wide array of topics in the Rayana dataset may be beyond the scope of our architecture in that the RNN generator has several different types of reviews to train with; reviews range from negative to positive and discuss various foods, so reducing the scope of the project may provide better results. In our review of other related works, we found that most papers trying to solve the same text generation problem used a dataset of similar texts that had less variation than that of our Yelp dataset.

In the future we would like to incorporate new evaluation metrics that would augment our current n-gram metric we are currently using. We would like to crowdsource the results from our various models compared with some of Rayana's dataset of fake/real reviews that would allow us to see if our generated reviews can be differentiated from the fake reviews that the Yelp filter has caught. In addition, we would like to experiment and tune different loss functions that would help our GAN achieve higher stability; we found that with the amount of epochs we ran and data we had, it was difficult to actually converge and find stability.

Finally, we would like to explore Conditional GANs[15] that would allow us to map unique start tokens to various generation tasks (e.g. ask the network for a 5-star review about ice cream). This would require us to either use sentiment analysis when parsing through our dataset or hand-labeling the reviews in our dataset. This may allow us to create more concrete sentences that won't talk about a wide array of foods, places, and topics.

7 Contributions

All three team members coded the project together. Veeral worked on data processing and hyper-parameter tuning. JJ worked on building word2vec models and the generator. Abhi worked on the policy gradient methods and optimizations. All three worked on the discriminator and wrote the paper together.

References

[1] Ranzato, et al. "Sequence Level Training with Recurrent Neural Networks." [1511.06732] Sequence Level Training with Recurrent Neural Networks, 6 May 2016, arxiv.org/abs/1511.06732.

[2] Yu, et al. "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient." [1609.05473] SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient, 25 Aug. 2017, arxiv.org/abs/1609.05473.

[3] "An Introduction to Yelp Metrics as of December 31, 2017." *Yelp*, www.yelp.com/factsheet.

[4] Grover, Joel, and Amy Corral. "Don't Fall for Fake Reviews Online." *NBC Southern California*, NBC Southern California, 30 Nov. 2017, www.nbclosangeles.com/news/local/Fake-Reviews-on-Yelp-Facebook-Google-447796103.html.

[5] Goodfellow, et al. "Generative Adversarial Networks." Generative Adversarial Networks, 10 June 2014, arxiv.org/abs/1406.2661.

[6] Bowman, et al. "Generating Sentences from a Continuous Space." Generating Sentences from a Continuous Space, 12 May 2016, arxiv.org/abs/1511.06349.

[7] William, et al. "MaskGAN: Better Text Generation via Filling in the______." MaskGAN: Better Text Generation via Filling in the______, 1 Mar. 2018, arxiv.org/abs/1801.07736.

[8] Alex, et al. "Professor Forcing: A New Algorithm for Training Recurrent Networks." Professor Forcing: A New Algorithm for Training Recurrent Networks, 27 Oct. 2016, arxiv.org/abs/1610.09038.

[9] George A. Miller. 1995. WordNet: A lexical database for English. Commun. ACM 38, 11 (1995), 39–41.

[10] Yao, et al. "Automated Crowdturfing Attacks and Defenses in Online Review Systems." [1708.08151] Automated Crowdturfing Attacks and Defenses in Online Review Systems, 8 Sept. 2017, arxiv.org/abs/1708.08151.

[11] YelpZip Dataset. odds.cs.stonybrook.edu/yelpzip-dataset/.

[12] Yelp, Inc. Yelp Dataset | Kaggle, 6 Feb. 2018, www.kaggle.com/yelp-dataset/yelp-dataset.

[13] Karpathy, Andrej. "Deep Reinforcement Learning: Pong from Pixels." Deep Reinforcement Learning: Pong from Pixels, 31 May 2016, karpathy.github.io/2016/05/31/rl/.

[14] Schulman, et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation." [1506.02438] High-Dimensional Continuous Control Using Generalized Advantage Estimation, 9 Sept. 2016, arxiv.org/abs/1506.02438.

[15] Mirza, Mehdi, and Simon Osindero. "Conditional Generative Adversarial Nets." Nov. 2014.