What's good for the goose is good for the GANder
Comparing Generative Adversarial Networks for NLP

Christina Hung
Stanford University
cjh@cs.stanford.edu

Brendan Corcoran
Stanford University
bmc2016@stanford.edu

Abstract

Generative Adversarial Nets (GANs), which use discriminators to help train a generative model, have been successful particularly in computer vision for generating images. However, there are many restrictions in its applications to natural language tasks–mainly, it is difficult to back-propagate through discrete-value random variables. Yet recent publications have applied GAN with promising results. Sequence GAN (Yu et al. 2017) introduces a solution by modeling the data generator as a reinforcement learning (RL) policy to overcome the generator differentiation problem, with the RL reward signals produced by the discriminator after it judges complete sequences. However, problems with this model persist, as the GAN training objective is inherently unstable, producing a large variation of results that make it difficult to fool the discriminator. Maximum-Likelihood Augmented Discrete GAN (Che at al. 2017) suggests a new low-variance objective for the generator, using a normalized reward signal from the discriminator that corresponds to log-likelihood. Our project explores both proposed implementations: we produce experimental results on both synthetic and real-world discrete datasets to explore the effectiveness of GAN over strong baselines.

1 Introduction

Generating authentic human language is a significant and worthwhile challenge in Natural Language Processing. As Richard Feynman said, "What I cannot create, I do not understand." Furthermore, text generation has wide-ranging applications from machine translation to summarization to dialogue generation.

Recurrent neural networks (RNNs), specifically with long short-term memory (LSTM) cells, have given a useful architecture for statistical token generation. The standard methodology has been to maximize the conditional probability of the next token based on the training data (maximum likelihood estimation, or MLE). However, there are several limitations to this baseline approach. First, it suffers from exposure bias: during generation, the generator may see partial sequences that it has not seen in the training data. It also poorly handles when the generator samples the next word stochastically rather than choosing deterministically. Finally, it tends to produce boring and statistically-safe predictions, instead of text that a human would produce.

This last point, along with the famous Turing test, has inspired several projects aimed at generating text that is indistinguishable from what a human would generate. This naturally leads to the idea of adding a new model in addition to the generator: a discriminator model that simply judges whether a given sentence was generated by a human or machine.

This setup, known as Generative Adversarial Networks (GANs), has proven to be quite successful in image generation and other real-valued settings (Goodfellow et al., 2016; Radford et al., 2015; Yang et al., 2017).

However, applying GANs to text generation is quite difficult. In image generation, gradients can be back-propagated from the result of the discriminator through the start of the generator, updating both models at the same time. This is possible since the generated image is composed of continuous pixel values. In the case of text, the generated elements are discrete words: it is impossible to slightly adjust the value of a word like is possible with a pixel.

Instead, a reinforcement strategy must be implemented, using rewards from the discriminator to update the generator's parameters. Several projects have attempted at some variation of this idea. Yu et al. (2017) proposes a model named SeqGAN that assigns a unique reward for every word in the sequence, based on approximating its average discriminator reward. Li et al. (2017) elaborates on this technique and applies it to neural dialogue generation. Finally, Che et al. (2017) proposes some additional measures, namely a new update rule, that helped the generator-discriminator model be better able to learn.

In this project, we compare the effectiveness of vanilla MLE (our baseline) to two different GAN models from the afore-mentioned literature: SeqGAN (Yu et al., 2017) and MaLiGAN (Che et al., 2017).

2 Baseline

Given that RNNs are a good model to generate sequential data with, the next question is how to train them. This is done by slowly forcing the generator to produce sentences that are similar to the training data via a process known as Maximum Likelihood Estimation (MLE) or teacher forcing.

Specifically, the model works as follows: First, take some t-word sentence from the training data: $Y = y_1, ..., y_t$. The generator (with parameters θ) can make a prediction $G_\theta(y_t|Y_{1:t-1})$ for the next word y_t given the first $t-1$ words. Then the likelihood that the generator generates the training sentence is:

$$G_\theta(Y) = \prod_{t=1}^{T} G_\theta(y_t|Y_{1:t-1}) \tag{1}$$

We want to increase this likelihood or, equivalently, decrease the following (negative log likelihood) loss function:

$$J(\theta) = -\sum_{t=1}^{T} \log G_\theta(y_t|Y_{1:t-1}) \tag{2}$$

At a high level: for each real sentence, consider each partial sequence (that is, feed as inputs into generator). Then increase the probability that the model predicts the next token in the real sentence. Run stochastic gradient descent on this loss on training data minibatches until convergence.

3 Models

3.1 Discrete GAN outline

In order to improve on the MLE baseline, we evaluate the effectiveness of applying GANs to help train a generator. Both GAN models (SeqGAN and MaLiGAN) that we evaluate take a similar approach to implementing GANs for discrete data.

The discriminator is a standard 2-class logistic regression problem, can be either a convolutional neural network (CNN) or a bidirectional RNN, and can be trained with the usual supervised learning techniques. It answers the following question: Is a sentence real or generated? We define $D_\phi(Y)$ as the probability that the sentence Y is real, according to the discriminator (with parameters ϕ).

On the other hand, the generator is trained to generate sentences that are able to fool the discriminator. It is updated as follows: generate a sample, get rewards from discriminator, and use those rewards to determine which probabilities should increase (using reinforcement learning).

Algorithm 1: Discrete GAN algorithm

```
pretrain generator
pretrain discriminator
for training iterations do
    for discriminator iterations do
        sample minibatch from real data
        generate minibatch using generator
        update discriminator parameters
    end
    generate minibatch using generator
    get rewards from discriminator
    use rewards to update generator parameters
end
```

The difficult part is *how* to use the reward from the discriminator to update the generator. The most naive gradient update (where Y is now a generated sentence) might look like:

$$D_\phi(Y) \sum_{t=1}^{T} \nabla_\theta \log G_\theta(y_i|Y_{1:t-1}) \tag{3}$$

That is, if $D_\phi(Y)$ is large (i.e. the discriminator thinks the sentence is real), increase the probabilities of each token in that sentence. However, this update does not prove to be effective enough, and both SeqGAN (Yu et al., 2017) and MaLiGAN (Che et al., 2017) improved on this update rule.

3.2 SeqGAN

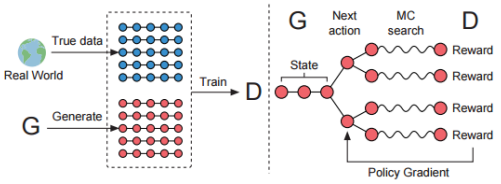

Figure 1: Diagram demonstrating discriminator training (left) and generator training (with Monte Carlo sampling) (right)

Based on our observations above, we note that regardless of how well an entire sentence fools the discriminator, each time step in that sentence should receive different rewards. For example, consider the sentences *It is windy today.* and *It is book today.* The discriminator will likely rate the second sentence fairly low since it does not make grammatical sense. However, *It* and *is* should not be penalized for this. To address this, SeqGAN (Yu et al., 2017) introduces a method to assign rewards to every generated step (REGS) by approximating the ultimate discriminator reward for each partial sequence.

Define $Q(Y_{1:t-1}, y_t)$ to be the expected discriminator reward given partial sequence $Y_{1:t-1}$ and choosing y_t as the next token. Then the update the parameters of the generator with the following gradient update:

$$\sum_{t=1}^{T} E_{y_t \sim G_\theta} \left[G_\theta(y_t|Y_{1:t-1}) \cdot Q(Y_{1:t-1}, y_t) \right] \tag{4}$$

At a high level, this states that if the Q score of some next token is high (that next token tends to generate full sentences that fool the discriminator), then the probability of that token should increase.

The next concern is that calculating Q exactly would be prohibitively expensive, growing exponential in the length of the sentence. Instead, Q can be approximated by sampling some sentences with a Monte Carlo tree search and averaging the resulting discriminator rewards. To be precise, let $MC(Y_{1:t})$ be a set of N sentences all starting with $Y_{1:t}$ and completed by freely running the generator. Then

$$Q(Y_{1:t-1}, y_t) = \frac{1}{N} \sum_{n=1}^{N} D(Y^n_{1:T})$$
$$\text{where } Y^n_{1:T} \in MC(Y_{1:t})$$

3.3 MaLiGAN

MaLiGAN handles the generator update in a different way. The basic version of the algorithm does not given unique rewards for every step (it can be extended to do so, but without much gain). Instead it introduces three tricks into the reward reinforcement: importance sampling, normalization, and baseline. In the end, the gradient update is:

$$\left(\frac{r_D(Y)}{\sum_i^m r_D(Y^i)} - b \right) \sum_{t=1}^{T} \nabla_\theta \log G_\theta(y_i|Y_{1:t-1}) \tag{5}$$

Importance sampling means the function $r_D(Y) = D(Y)/(1 - D(Y))$ is used instead of $D(Y)$ itself which weights high discriminator scores even higher. Normalization serves to determine the realtive strongest sentence in the minibatch. Effectively, these combine to identify the strongest one or two sentences and boosts their probabilities.

Finally, we subtract a baseline, so that not only are the strongest sentences rewarded, but the weakest sentences are also penalized.

4 Synthetic Data Experiments

To demonstrate a simple proof of concept, we conducted a simulated test with synthetic data, similar to the method described by Yu et al. (2017). Generally speaking, a language can be defined in terms of its underlying statistical model, which is then encoded in the parameters of a generator RNN. That is, when we try to generate natural English, we are actually trying to find the parameters of this oracle RNN. Thus, for our synthetic language model, we create a randomly-initialized LSTM as our "true" (oracle) model to generate a synthetic data distribution with a vocabulary of 5000 tokens. We will describe our method of creating this oracle more in the following section.

4.1 Evaluation Metric

One problem with the task of generation is that it is fairly difficult to evaluate. Some possible methods of evaluating how good the generated sentences include the use of BLEU scores, accuracy of the discriminator, cross entropy loss (the loss from MLE), or manual human evaluation.

Cross entropy loss is attractive because it is precise and easy to evaluate. Given a real sentence, the generator tells you how likely it was to have generated that sentence. But we actually want

to measure the opposite: given a generated sentence, how likely is it for that to be a real "human" sentence or a machine-generated sentence? This is what human evaluation can tell us; however, due to the variation and nature of human judgement, sole reliance on human evaluation is both impractical and unfeasible during training. Fortunately, using synthetic data can give us the best of both worlds.

As mentioned earlier, language can be defined in terms of its underlying statistical model, encoded in the parameters of a generator "oracle" RNN. With synthetic data, since we would already know the parameters of our oracle RNN, we can evaluate the probability that an arbitrary sentence was produced by the oracle–i.e. whether it is a real sentence (that came from the underlying statistical model).

As there is no available oracle for English (that's what we want to find!), we can instead create one for a synthetic language as follows: Arbitrarily set the parameters of an RNN. Use this RNN to generate some samples (real data for the synthetic language). Train the generator on this synthetic training data. To evaluate the performance of the generator, use the loss function:

$$J_{\text{oracle}} = -\sum_{t=1}^{T} \log G_{\text{oracle}}(y_t|Y_{1:t-1}) \quad (6)$$

While the usual loss compares real sentences against our generator, this compares generated sentences against the oracle RNN and thus directly measures the probability that matters.

In fact, during MLE training (while minimizing generator loss on the train set) we plot both generator loss (on the dev set) and oracle loss at each epoch and compare.

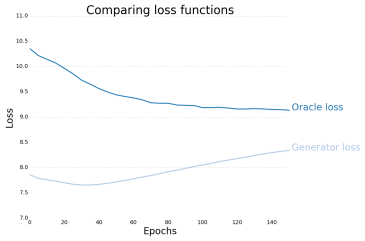

Figure 2:

As this shows, the oracle loss steadily decreases while the generator loss decreases at first and then steadily increases. This is somewhat expected and demonstrates an important point: when the goal is to create human-sounding text, overfitting to the training set is not necessarily to be avoided. Overfitting means the generator would tend not to generate the exact sentences of the evaluation set. But what matters is that what the generator *does* generates could pass for being human-generated.

4.2 Results

After establishing an oracle RNN and generating 10,000 20-word synthetic sentences of training data, we ran MLE, SeqGAN, and MaLiGAN and conducted the following measurements.

First, both SeqGAN and MaLiGAN require pretraining using MLE. We pretrain for 150 epochs with batch size 64 and observe a drop and convergence in the oracle loss, as desired.

Next, we pretrain the discriminator, which is again the same for both SeqGAN and MaLiGAN and observe its accuracy improve from 50% to $> 99\%$.

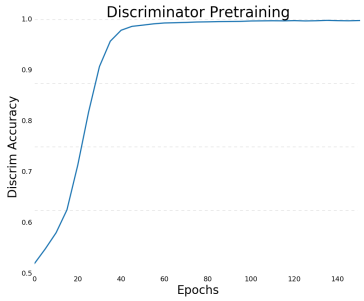

Figure 3:

Next, as a sanity check, we freeze the discriminator after pretraining and run SeqGAN with only the generator updating. We expect the accuracy of the discriminator to drop since a better generator should be harder to distinguish from real data.

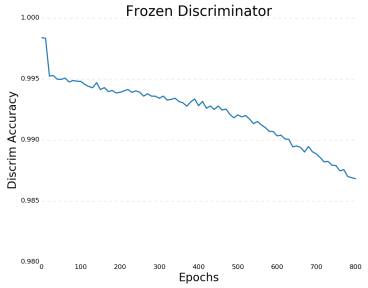

Figure 4:

Finally, we ran each algorithm under normal settings to compare the results.

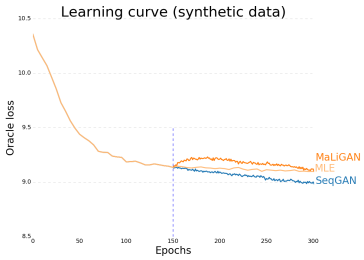

Figure 5:

4.3 Discussion

There are a couple of notable results from these plots:

First, even with a frozen discriminator, the generator made surprisingly little progress in reducing the discriminator accuracy. Over 800 epochs, the accuracy fell from around 99.5% to around 98.5%. This suggests that the discriminator is always ahead in the game. If even by the end, the discriminator can almost perfectly pick out the generated sentences, then the generator could stand to be improved.

On the other hand, we do observe modest gains for both algorithms compared to the MLE baseline. SeqGAN immediately improves its oracle loss, while MaLiGAN has a setback at first but seems to catch up and potentially surpass MLE.

5 Real Data Experiments

In addition to using synthetic data for a proof of concept, we further explored the limitations of our models on a more challenging but fundamental task: sentence-level language modeling. For simplicity and efficiency, we only compared the results of the MLE baseline and MaLiGAN models. We used the standard Pen Treebank (PTB) dataset (Marcus et al., 1993), in particular the pre-defined training and evaluation sets from the corpus.

5.1 Evaluation Metric

As mentioned earlier, there are many ways to evaluate text generation: BLEU scores, accuracy of the discriminator, cross-entropy loss, and manual human evaluation; and of these, cross entropy loss is the most attractive as it is precise and easy to evaluate. Therefore, we report sentence-level perplexity, which is averaged over all the sentences in our evaluation set.

Another attractive evaluation metric is BLEU-scoring, which measures the similarity between the quality of machine translation and human texts. In our context, BLEU would evaluate the sentences generated by our generator, compared to the original 'gold' sentences in PTB. Thus, following the example of previous authors Yu et al. (2017) and Che at al. (2017), we also report the BLEU-3 scores of our generated sentences.

Finally, since evaluation of natural language is incomplete without some aspect of human evaluation, we performed human evaluation on a few generated sentences as well. We took 5 sentences generated from MLE and MaLiGAN, and judged whether each of the sentences generated by our model(s) were likely to have been generated by a human.

5.2 Training Setting

In order to reduce the variation of sentence length, we removed sentences longer than 35 words, and padded sentences shorter than 35 words. This left us with 32584 sentences in the training set, and 1411 sentences in the evaluation set. To reduce the percentage of unknown words, we additionally brought down the size of our vocabulary to contain only the 10k most commonly used words in the training set; and used GloVe vectors (Pennington et al., 2014), pretrained on Common Crawl (having around 42B tokens, 1.9M vocab, and 300d vectors) for our word embeddings. Finally, we also used <START> and <END> tokens, as well as <UNK> tokens for words in the corpus that are either not in the top 10k most common words, or not in the pretrained GloVe vectors.

Like the synthetic data experiments, we pretrained MaLiGAN on around 150 epochs of MLE using a batch size of 64. To reduce overfitting, which more-readily occurs in real-word data since we used pretrained GloVe vectors, we introduced a dropout keep probability of 0.6 to our discriminator.

5.3 Results

We note the sentence-level perplexities achieved by MLE and MaLiGAN from both our experiments and those directly reported by Che et al. (2017) in Table 1 and the best BLEU-3 scores of generated sentences from the models in Table 2.

Here are 5 example generated sentences from each model as well.
MLE:

1. <START> they 're sell the emerging york 's chief economic jeep , ... <END> ...<END>

Table 1: Eval-perplexities

Algorithm	Eval-Perplexity
MLE	128.2069
MLE (Che et al.)	141.9
MaLiGAN	123.5861
MaLiGAN (Che et al.)	131.6

Table 2: BLEU-3 scores of generated sentences

Algorithm	BLEU-2
MLE	0.55186
MaLiGAN	0.74332

2. <START> monday in the president , right stock prices of s&p at 10 million ... <END> ...<END>
3. <START> the earlier the achieved was back to be next neck by shield interest ... <END> ...<END>
4. <START> <START> " wilbur will determine any noting , makes rule or the california and come to the number than practically admission ...<END> ...<END>
5. <START>but the crops in 101 % rallied out to program need , which said the company were relieved ..., the agency hastings fled sidhpur ...<END> ...<END>

MaLiGAN:

1. <START> the house included the indicated segment group except pearce barrett ...<END> ...<END>
2. <START> the investment games is for houses , " said the department ... <END> ...<END>
3. <START> many economists warburg privately urge adding nissan and demand moving soup . <END> ...<END>
4. <START> wall pepsi responded intensely points decidedly at its impeachment . <END> ...<END>
5. <START> the government says barely $ 37 million ...<END> ...<END>

5.4 Discussion

As seen in the results from Table 1 above, both of our reported perplexities appear to be slightly lower than those reported by Che et al. (2017). We attribute these low perplexities to our use of pretrained GloVe vectors. However, the key takeaway from these reported perplexities is that the perplexity produced by MaLiGAN performs slightly better than MLE, albeit by a lesser margin than the perplexities reported by Che et al. (2017). This may be because GANs are fairly difficult to train, and require exact hyperparameter tuning–the specific hyperparameters used by Che et al. (2017) to achieve such results are unknown.

In examining the BLEU scores, however, we note that MaLiGAN outperforms vanilla MLE by a much higher margin. This seems to imply that while the perplexities of MLE and MaliGAN are fairly similar, MaLiGAN is more equipped to produce much more "human"-like text, than MLE. Similarly, in examining the sample sentences generated by each of the models, although the sentences do not appear to carry much meaning, it appears that MaLiGAN prodcues sentences that follow the constructs of the English language more closely than those produced by vanilla MLE. Thus, from these results, we have reason to believe that MaLiGAN may be beneficial and produce fruitful results in other similar natural language applications, such as machine translation and neural dialogue generation.

6 Conclusions and Future Work

In our project, we explored the limitations of 3 different text generation models: vanilla MLE, SeqGAN (Yu et al., 2017), and MaLiGAN (Che et al., 2017). Using synthetic data, we were able to show that in general, GANs are modestly more successful at generating text than vanilla MLE. Using real data comparisons between MLE and MaLiGAN, we were able to see a significant improvement in terms of 'human-likeness' between sentences generated by the two models.

As for future work, it would be interesting to implement input clamping to further reduce the variance of input sequence lengths and prevent overfitting. Additionally, the results of our MaLiGAN model point to the potential for promising results in applications to areas of text generation where the end-result is expected to sound more 'human.' Neural dialogue generation is an application where it is important to produce human-sounding sentences, and therefore would be an interesting one to explore.

Acknowledgments

We would like to thank our project adviser Kevin Clark, instructors Chris Manning and Richard Socher, and the entire teaching staff of CS224N.

References

Che, T.; Li, Y.; Zhang, R.; et al. 2017. Maximum-Likelihood Augmented Discrete Generative Adversarial Network. In arXiv: 1702.07983.

Goodfellow, I.; et al. 2014. Generative adversarial nets. In NIPS, 26722680.

Goodfellow, I.; et al. 2016. NIPS 2016 Tutorial: Generative Adversarial Networks. In arXiv: 1701.00160.

Lamb, A.; et al. 2016. Professor Forcing: A New Algorithm for Training Recurrent Networks. In arXiv: 1610.09038.

Li J; et al. 2017. Adversarial Learning for Neural Dialogue Generation. In arXiv: 1701.06547.

Marcus, M., Marcinkiewsicz, M., and Santorini, B. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. In Computational Linguistics, 19(2):313-330.

Pennington J., Socher R., and Manning C. 2014. In GloVe: Global Vectors for Word Representation.

Radford, A.; et al. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In arXiv: 1511.06434.

Yang J et al. 2017. LR-GAN: Layered Recursive Generative Adversarial Networks for Image Generation. In arXiv: 1703.01560.

Yu, L.; Zhang, W.; et al. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17).