Evaluating Generative Models for Text Generation

Prasad Kawthekar
Department of Computer Science
Stanford University
pkawthek@stanford.edu

Raunaq Rewari
Department of Management Science and Engineering
Stanford University
raunaq@stanford.edu

Suvrat Bhooshan
Department of Computer Science
Stanford University
suvrat@stanford.edu

Abstract

Generating human quality text is a challenging problem because of ambiguity of meaning and difficulty in modeling long term semantic connections. Recurrent Neural Networks (RNNs) have shown promising results in this problem domain, with the most common approach to its training being to maximize the log predictive likelihood of each true token in the training sequence given the previously observed tokens. Scheduled Sampling, proposed by Bengio et al. (2015), was proposed as an improvement to the maximum likelihood approach by stochastically introducing inference steps during training steps. More recently, Generative Adversarial Nets (GAN) that use a discriminative model to guide the training of the generative model have become popular in vision domain, and also reinterpreted as a reinforcement learning problem to adapt it to text generation by Yu et al. (2016). Here we test and compare these three approaches and thus hope to extend the evaluation presented for the SeqGAN model in Yu et al. (2016) using two additional datasets and an additional perplexity evaluation metric.

1 Introduction

Being able to generate text that is close to the quality of human generated text has a lot of applications like translation from one language to another, chatbots, question answering etc. RNN extensions such as LSTMs or GRUs that retain long term memory of tokens have been shown to work well in practice using maximum likelihood estimation. However, training using maximum likelihood has its downsides like exposure bias. This refers to the situation where during training the prediction of the next word conditioned on the previous word becomes infeasible since the previous word may not have been seen in the training data. If the generator makes an error early on in the generation process, the generated sentence will keep diverging further away as more words are generated. To solve this issue, a few things have been tried in the past, like scheduled sampling (Bengio et al. (2015)) and having task specific sequence scores. Here we also analyze SeqGAN, a recently introduced Generative Adversarial Network model proposed by Yu et al. (2016). We delve into greater details for GANs in the upcoming sections, but the GAN training procedure essentially consists of a battle between a discriminator and a generator. The goal of the generator is set to fool a discriminator by generating text similar to the original dataset. In the best case scenario, we would reach Nash equilibrium and the discriminator will output a probability of 0.5 for each of the two classes, which means that the true and generated examples are indistinguishable. GANs have shown considerable success in the computer vision domain, but very little work has been done on text. One reason for this is that GANs work on real valued data, whereas text is represented atomically in terms of discrete tokens (like ”man”, ”girl” etc). The discriminator gets inputs from the output of the generator that

it tries to update using the gradient from the loss function. In computer vision, the output of the generator is an image (a matrix consisting of real valued numbers) which can be updated by small amounts to make it more difficult for the discriminator to differentiate between the real and fake image. A similar procedure is not possible in case of a generator generating text. To make GANs work for text, we need to work around this problem of non-differentiable outputs and one solution is using reinforcement learning. Another issue that we need to keep in mind is that the discriminator will only be able to give feedback (reward) after the generator has generated the complete sentence, and hence there is no way to judge the quality of sentence produced mid-generation.
The SeqGAN algorithm models the generator as a stochastic policy where the state is the generated tokens so far and the action is the next token to be generated. Having a stochastic policy lets us sample different actions during training and gives us a robust estimate of the policy. The algorithm uses gradient policy update to work around the generator differentiation problem along with using Monte Carlo Tree Search (MCTS), which is a known effective way to get the delayed reward at the end and use that to update the intermediate steps. Policy Gradients have to actually experience a positive reward, and experience it ever so often in order to eventually and slowly shift the policy parameters towards repeating actions that give high rewards. Policy gradient uses Monte Carlo (MC) search to approximate the state-action value. Hence, text generation is a good fit to apply policy gradients. In this paper, we have run experiments to compare this RL approach with other algorithms like Maximum Likelihood Estimation (MLE) and Scheduled Sampling (SS).

2 Related Work

This section starts by talking about generative models and then transitions to talking specifically about the various approaches taken towards text generation.
One of the most popular generative models is the Fully Visible Belief Network (FVBN), which forms the basis for sophisticated generative models from DeepMind, such as WaveNet. Although WaveNet is able to generate realistic human speech, the main drawback of FVBNs is that samples must be generated one entry at a time.

$$\text{p}_{model}(x) = \prod_{i=1}^{n} p_{model}(x|x_1, x_2, x_3....., x_n)$$

Hence, the cost of generating a sample is O(n). Since these steps cannot be parallelized, WaveNet thus requires two minutes of computation time to generate one second of audio, and cannot yet be used for interactive conversations. Recently, variational autoencoder (VAE) that combines deep learning with statistical inference intended to represent a data instance in a latent hidden space, have also become popular. GANs do not have this parallelization problem and do not need a variational bound (like in VAE) and seems to be the algorithm of choice.

Another interesting strategy proposed by Yoshua Bengio, called *Professor Forcing*[2] works on accomplishing a similar goal as Scheduled Sampling but instead uses adversarial domain adaptation to encourage the dynamics of the recurrent network to be the same when training the network and when sampling from the network over multiple time steps. Here, the discriminator is trained as a probabilistic classifier that takes as input a behavior sequence b derived from the generative RNNs activity (hidden and outputs) when it either generates or is constrained by a sequence y. Hence, the behavior sequence b is either the result of running the generative RNN in teacher forcing mode (with y from a training sequence with input x), or in free-running mode (with y self-generated according to P_θ(y|x), with x from the training sequence). Hence, this approach essentially tries to make the hidden state representations are each time point in the RNN be very similar to each other such that after the training, the RNN is able to output tokens which makes the sentence appear real. Let the function B(x, y, θ) output the behavior sequence (chosen hidden states and output values) given the appropriate data (where x always comes from the training data but y either comes from the data or is self-generated). Let D(b) be the output of the discriminator, estimating the probability that b was produced in teacher-forcing mode, given that half of the examples seen by the discriminator are generated in teacher forcing mode and half are generated in the free-running mode. The model was trained using the standard loss functions for discriminator and generator, which we will see in the next section.

As discussed, we can also model the text generation problem as reinforcement learning task, where the state is the currently generated tokens and the action corresponds to selecting the next token. This strategy involves using the generator as a stochastic policy and using MCTS to approximate state-action pairs in the intermediate nodes once the fully generated sentence receives a reward. Such a strategy was successfully used by DeepMind to make AlphaGo and the REINFORCE algorithm used in our model was also used for *dialogues generation*[3] by Jurafsky et. al.

3 Approach

This section first introduces GANs and then talks about the specific details of the models analyzed in this paper.

3.1 Generative Adverserial Networks (GANs)

The basic idea of GANs is to set up a game between two players i.e. a generator and a discriminator. The generator's job is to create samples that make it seem that they come from the real distribution (approximated by the training data distribution). The discriminator usually does supervised learning using a deep network to learn to differentiate the true samples from the fake ones. The players have cost functions that are defined in terms of both players parameters i.e.$\theta^{(D)}$ and $\theta^{(G)}$ for the parameters of the discriminator and generator respectively. The discriminator wishes to minimize its loss, J$^{(D)}(\theta^{(D)}, \theta^{(G)})$ and must do so while controlling only $\theta^{(D)}$. The generator wishes to minimize its loss, J$^{(G)}(\theta^{(D)}, \theta^{(G)})$ and must do so while controlling only $\theta^{(G)}$. During GAN training, the discriminator and the generator are trained one after the other such that both keep improving simultaneously and we are able to get human quality samples from the generator. The cost functions used for the two players are as follows:

$$\textit{Discriminator}\text{: J}^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\tfrac{1}{2}\,\mathbb{E}_{x\sim p_{data}}\, logD(x) - \tfrac{1}{2}\,\mathbb{E}_z\, log(1 - D(G(z))$$

$$\textit{Generator}\text{:}-\tfrac{1}{2}\,\mathbb{E}_z\, logD(G(z)$$

3.2 Maximum Likelihood Estimation (MLE)

We have used cross-entropy loss to train the model using MLE. During training, we feed in the words of the given sentence at each time point and the expected output at each step is the next word in the sentence. Hence, the training protocol tries to find the best parameters that explain the data (text):

$$\theta^* = \textstyle\sum_{X^i, Y^i} logP(Y^i|X^i, \theta)$$

During inference, instead of feeding in the true words, the output of the current time step becomes the input to the next one.

3.3 Scheduled Sampling (SS)

Scheduled Sampling[1] is one of the methods that aims to avoid the problem of discrepancy between training and generating mode that was introduced early on in the paper. The main difference between training and generating for sequence prediction tasks when predicting the next token is whether we use the true previous token or an estimate coming from the model itself. Scheduled Sampling suggests a sampling mechanism that randomly makes this decision such that the training and generating mode become closer to each other. A curriculum learning strategy is used where at the beginning of training, true tokens are used because the model at this point is not trained at all. As the model gradually keeps improving, we start sampling more from the model, which is closer to how it is done in the generating mode. In this paper, we used a linearly decaying curriculum.

3.4 SeqGAN

Like mentioned earlier, we needed to design a solution that lets us work around the non-differentiability of generator outputs and the fact that our discriminator can send the reward only after generation of the complete sentence, which is where SeqGAN comes to the rescue. There is a discriminative model that gets true examples from real training text and negative examples from the generated sequence and learns to differentiate the two classes. The generator uses policy gradient and Monte Carlo search along with the rewards from the discriminator to update its parameters. Figure 1 shows the overall training procedure (from [4])

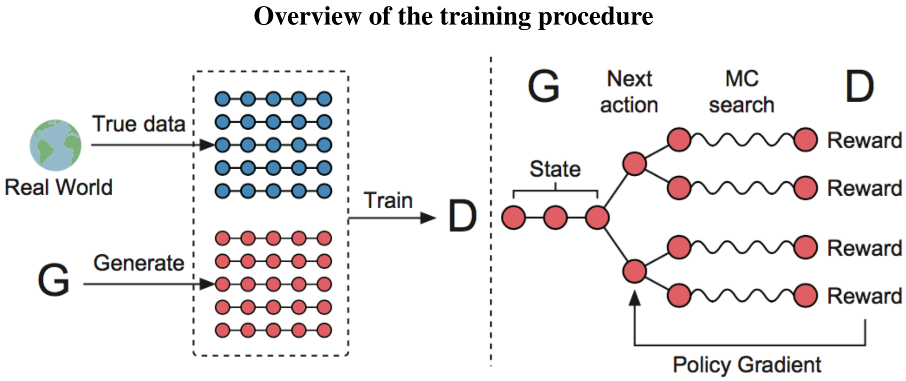

Figure 1: Left: Discriminator gets real and fake examples for training. Right: Shows the current state and next action representation. Also shows how MC search is used to reach the reward and policy gradient is used to update the policy.

Switching back to the specifics of the model used in this paper. Let, G_θ be the stochastic generator policy and D_ϕ be the discriminator. MC search uses rollout policy, G_β to generate further states and actions till it finishes generating and gets a reward from the discriminator. We have used our generator policy as the rollout policy to get accurate estimates of the action value function. The role of the discriminator is relatively straightforward; it is used to provide the reward function and updates itself using the following cost function that continuously helps generate even better examples:

$$\mathrm{J}^{(D)} = -\tfrac{1}{2}\,\mathbb{E}_{Y\sim p_{data}}\, logD_\phi(Y) - \tfrac{1}{2}\,\mathbb{E}_{Y\sim G_\theta}\, log(1 - D(Y))$$

It has been shown in previous studies that a GAN trains much better if the discriminator and generator are pre-trained. We can train the discriminator to minimize its cross entropy and the generator to perform Maximum Likelihood Estimation (MLE).
We would ideally like to maximize our expected end reward, i.e.:

$$J(\theta) = \mathbb{E}[R_T|s_0,\theta] = \textstyle\sum_{y_1\in Y} G_\theta(y_1|s_0)\ .\ \mathrm{Q}^{G_\theta}_{D_\phi}(\mathrm{s}_0, \mathrm{y}_1)$$

where, R_T is the reward for the completed sequence and $\mathrm{Q}^{G_\theta}_{D_\phi}$ is the action-value function which we need to estimate for the end of sentence as well as at each intermediate token. We already know that the action-value function for the completed sentence is the output of the discriminator, but for the intermediate tokens, we use Monte Carlo search with a rollout policy that is equal to the generator policy. The more ”episodes” of rollout we run, the better our estimates will become. Once you have these action-value pairs for each token in the generated sentence, you can update the parameters of the generator using policy gradient as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{Y_{1:t-1}\sim G_\theta}\Big[\sum_{y_t\in\mathcal{Y}} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q^{G_\theta}_{D_\phi}(Y_{1:t-1}, y_t)\Big]$$

Then we can use these gradients to update the parameters using gradient descent. In our experiment we first pretrain an LSTM generator for 10 epochs on MLE. Then we continue training the generator for 10 more epochs using a CNN discriminator for text. We use Adam optimizer for gradient updates.

4 Datasets and Experiments

4.1 Datasets

We ran all of our experiments on two datasets: Cornell Movie–Dialogs Corpus (CMC), and Penn - TreeBank (PTB) dataset from Tomas Mikolov's webpage (http://www.fit.vutbr.cz/ imikolov/rnnlm/simple-examples.tgz).
The movie corpus contains a large metadata-rich collection of fictional conversations extracted from raw movie scripts [6]. There are 220,579 conversational exchanges between 10,292 pairs of movie characters. We preprocess the dataset to extract only the dialogues, and limit the vocabulary to the 10,000 most common words int the corpus. The rest of the words are replaced by the $< unk >$ token.

Similarly, we use the PTB dataset from Tomas Mikolov's webpage. The dataset is already preprocessed and contains overall 10000 different words, with the same $< unk >$ token for unknown words[6].

4.2 Experiments

We divide our datasets into three parts: 70% of the dataset is used for training, 15% is used for validation and 15% is reserved for testing.

We tested 3 different models i.e. MLE, SS, SeqGAN, which we have explained in previous sections in this paper.
We hyper parameterized the embedded length and hidden dimension of the RNN corresponding to the LSTM generator. We tried various values for embedded length (64, 128, 192, 256) and hidden dimension (50, 100, 150, 200) and found that the largest model (embedded length=256, hidden dimension=200) achieves the minimal validation loss. After the best hyper parameters are chosen for each dataset, each model is trained using these hyper parameters and the various evaluation metrics are computed, which we will talk about in the next subsection. The implementation for the models is obtained from (https://github.com/LantaoYu/SeqGAN). The code is adapted to work for natural language text (as opposed to the artificial synthetic experiment described in the original paper), to hyper-parametrize the models and to evaluate the additional metrics described below.

4.3 Evaluation Metrics

Since evaluation is one of the main focus of this paper with the hope of extending the results presented in the SeqGAN paper by Yu et al. (2016), we evaluate the models on the two new datasets above and also report the loss, perplexity and example sentences for qualitative judgment. These metrics are described below-

1. Loss: We have calculated cross-entropy loss for training, validation and test sets for both datasets. SeqGAN is guided by the discriminator loss during training, but we have reported the generator loss to keep things consistent.

2. Perplexity: Perplexity is a measurement of how well a probability distribution predicts a sample. It is usually calculated as 2^H, where H is the cross-entropy loss.

3. Qualitative judgment: Since Generative Adversarial Network are trained to generate more ”realistic” human-like samples, they do not optimize for traditional cross-entropy loss directly used in Maximum likelihood methods. Therefore, we decided to qualitative evaluate the samples generated by all three models and see from a human's perspective if GAN's generate better samples, and what the distribution of the perplexities on these samples are.

5 Results

5.1 Loss Curves

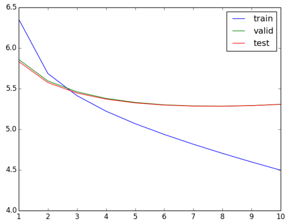

Figure 2: MLE

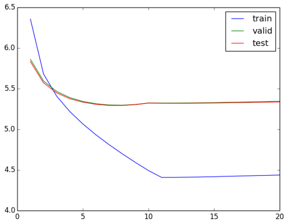

Figure 3: SeqGAN

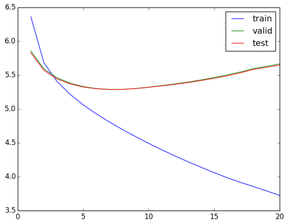

Figure 4: SS

Learning Curves: PTB Dataset

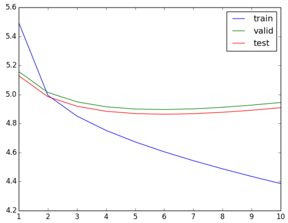

Figure 5: MLE

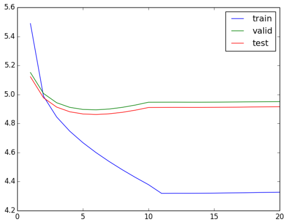

Figure 6: SeqGAN

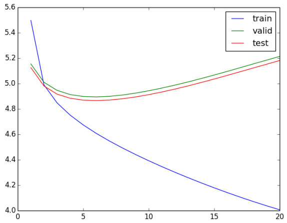

Figure 7: SS

Learning Curves: CMC Dataset

The training loss for the MLE model keeps decreasing, however, the test and the validation loss converge in around 5-7 epochs. The model shows signs of over fitting as the test accuracy increases slightly afterwards on further training.

Scheduled Sampling model has the best learning loss out of all the three models. This is expected as the model tries to maximize the likelihood the data similarly to MLE models. However, it is prone to overfitting and after converging in 5-7 epochs, the test loss starts increasing. The point to note is that after 20 epochs, the train loss decreases heavily, and even though the test loss is high, the quality of new sentences generated by the model is very good, as is shown in the section below.

5.2 Test Perplexity

Table 1: Test Perplexities

Dataset	MLE	SeqGAN	SS
PTB	38.93	39.12	39.05
CNC	29.10	29.09	29.16

We hypothesized that however the SeqGAN model does not optimize for the cross entropy loss function, it should predict sentences which are more human like, and should have multi-modal Gaussian distribution where it performs really well on some subsets of sentences in the test set, but really poorly on some others. We hoped that this would explain the poor test loss and perplexity of the SeqGAN algorithm, whereas the quality of the text generated by the model would still be good.

However, on plotting the distribution of the perplexities on sentences in the test set, it was found that all three models have a Gaussian distribution.

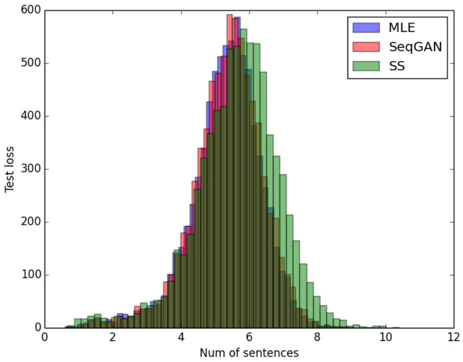

Figure 8: PTB

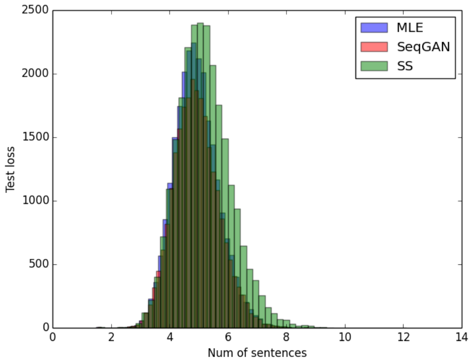

Figure 9: CMC

Test perplexity distributions.

5.3 Example Generated Text

MLE

PTB (Epoch 0). *a cotton appropriations financial prospects an threat to have benefits tremors that they say the decade the carpet nekoosa he*
CMC (Epoch 0). *You deserve away... maybe. I would let you see a* $< unk >$ *No, I take this! I don't let me it.*

PTB (Epoch 5). *wall street reports for activities of responsibility for latin america 's aircraft station in N investment banking and commerce department*
CMC (Epoch 5). *a good mother! Nothing. I know where it is? Yeah, we're what I mean about. Be sure the actual business*

PTB (Epoch 9). *our national ability to prevent nicaragua from the british trade*
CMC (Epoch 9). *Only call me and listen to me. Do you even know what to do the same to think of*

At the beginning, the algorithm is just predicted random words, as no learning has happened. Halfway through the training process, the model is generated pretty good sentences. It learns possessive nature of "'s" which is a word in itself in the model and correctly links it to america. Similarly, we can see the model learning to predict capitalized words at the beginning of new sentences. At the end of the training process, the model is able to generate sentences with a complicated phrase structure.

Scheduled Sampling

PTB (Epoch 0). *relating on a paid for offer the security area are the tax saying said some would looked they is one-day*
CMC (Epoch 0). *Everything's coming out here. Because you've tons*

PTB (Epoch 5). *the junk-bond market in financial center has made the merger bidding war about her*
CMC (Epoch 5). *You can lose this machine woman.*

PTB (Epoch 20). *japanese companies also have routine business jobs in the u.s. virgin islands*
CMC (Epoch 20). *Wouldn't be much surprised if they put that in the ground.*
You can't prove much otherwise.
I won't tell you the facts.
At the beginning of the training process, the outputs are just random words without any meaning, because no learning has happened. Halfway through the learning process, the model is already producing high quality sentence outputs. At the end of the learning process, the model is generating

sentences which can be compared to those written by humans in quality.

SeqGAN

PTB (Epoch 0). *employees may cure they were considering the agency that 's two congress cases ms. johnson clearly noted that began growth*
CMC (Epoch 0). *Why didn't you get you this on her? It doesn't need to be safely bad on me. He told me*

PTB (Epoch 5). *can end of its criminal office charges to remove the pacific law which is all the* $< unk >$ *response to the*
CMC (Epoch 5). *long as we have to find something true. I guess that was something. My brother's* $< unk > < unk >$ *Shit, they're gonna*

PTB (Epoch 9). *capital offers flat the debt carrier to imports from* $< unk >$ *heritage mr. nadeau said it expects net sales to reduce*
CMC (Epoch 9). *you're a very lucky woman. Why don't you say yes or you can do it. Next time, I don't know.*
At epoch 0, the samples are already decent because MLE pretraining has already occurred. Some example sequences generated after SeqGAN training are also shown above.

6 Conclusion

We calculated the Test Perplexities for the three models discussed in the SeqGAN paper on two new datasets.
The proposed SeqGAN algorithm does not significantly affect the training loss used in the MLE algorithm because it is not optimizing for likelihood. Instead, it is focused on fooling the discriminator. In our experiments, we found that the discriminator becomes really good in differentiating the SeqGAN's sentences from the true ones. However, since it uses reinforcement learning to give rewards only after a full sentence has been generated, it is not that effective in training.

In general, training Generative Adversarial Networks (GANs) is a hard problem for textual input, because text is inherently discrete, and therefore we cannot take perform mini updates in any direction for the model to learn, (as opposed to the continuous domain of images where they perform really well). We can argue that words can we represented as word vectors which are continuous. However, the vocabulary space is very small when compared to the overall continuous space of vectors, and no learning happens when we update the word vector in any direction, as the nearest neighbor in the vocabulary still remains the same. We need to make large updates to reach different words in the vocabulary, and at that stage, there is no meaningful information being learnt by the model.

References

[1] Bengio, S.; Vinyals, O.; Jaitly, N.; & Shazeer, N. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. *In NIPS*, 11711179
[2] Lamb, Alex M., et al. Professor forcing: A new algorithm for training recurrent networks. *Advances In Neural Information Processing Systems*. 2016.
[3] Li, Jiwei, et al. Adversarial Learning for Neural Dialogue Generation. *arXiv preprint arXiv:1701.06547* (2017).
[4] Yu, Lantao, et al. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473* (2016).
[5] https://www.tensorflow.org/tutorials/recurrent
[6] http://www.cs.cornell.edu/cristian/Cornell-Movie-Dialogs-Corpus.html