Generating Jokes with SeqGAN

Junyang Jin **Maggie Yin**

Abstract

Generating and evaluating jokes are two challenging tasks for the current deep learning models. SeqGAN (Yu et al., 2016) is an approach to generating sequences adversarially by modeling the data generator as a stochastic policy in reinforcement learning and directly performing gradient policy update. In this project, we apply the methods of SeqGAN to joke generation, that is, to teach a GAN to tell jokes. While the SeqGAN synthetic data experiments showed good results and convergence using a ground truth "oracle" model to compute the exact loss of the samples generated, to actually apply the approach to real life tasks like joke generation, we supplied the model with jokes from a dataset and used the generator itself to monitor training progress. We develop a comprehensive set of evaluation metrics to evaluate the performance of SeqGAN on generating jokes, and perform several experiments in varying the algorithm parameters to observe the effect on training and convergence.

1 Background Work

Computational humor is still a developing field, with more advancements in recent years thanks to improvements in artificial intelligence and linguistic formulations of humor. The existing work on the use of GANs to generate humour is limited to Pun-GAN (Luo et al., 2019), which uses a GAN to generate puns. Our approach applies the methods of SeqGAN to the task of joke generation, and we use specifically two different implementations in PyTorch (Nair, 2018) (ZiJianZhao, 2019). We also look at a state-of-the-art humor detection model called ColBERT (Annamoradnejad and Zoghi, 2021) as part of our evaluation.

2 Datasets

We use a collection of three joke datasets (cite) that are scraped from three different websites: Reddit, StupidStuff, and Wocka. The jokes are primarily short jokes or one-liners. Our train set includes 21000 jokes with a vocabulary size of 20912. One problem that exists with joke datasets available to us in general is that they contain a significant amount of vulgarity and/or offensive language and stereotyping that will inevitably transfer over to the jokes that are generated by the model.

3 Methods

3.1 SeqGAN Approach

The basis of SeqGAN is that it models text generation as a sequential decision making process in reinforcement learning, where the reward signal is provided by the discriminator at the end of each episode, and the generator picks the action and learns the policy using estimated overall rewards.

Thus we train a θ-parameterized generative model G_θ to produce a sequence $Y_{1:T} = (y_1, ..., y_t, ..., y_T), y_t \in \gamma$, where γ is the vocabulary. Additionally, we train a ϕ-paramaterized discriminator D_ϕ to produce a probability indicating how likely a sequence $Y_{1:T}$ is from real sequence data. Thus, as in Figure 1, the discriminator is trained on both real data and ”fake” data generated by the generator. The generator is then updated via policy gradient and Monte Carlo search on the basis of expected end reward from the discriminator.

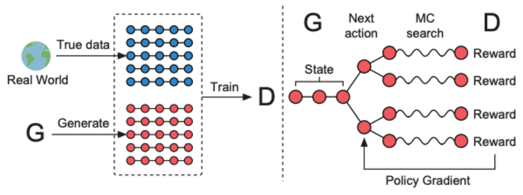


Figure 1: SeqGAN diagram

Algorithm 1 Sequence Generative Adversarial Nets

```
Require: generator policy $G_\theta$; roll-out policy $G_\beta$; discriminator
    $D_\phi$; a sequence dataset $\mathcal{S} = \{X_{1:T}\}$
 1: Initialize $G_\theta$, $D_\phi$ with random weights $\theta, \phi$.
 2: Pre-train $G_\theta$ using MLE on $\mathcal{S}$
 3: $\beta \leftarrow \theta$
 4: Generate negative samples using $G_\theta$ for training $D_\phi$
 5: Pre-train $D_\phi$ via minimizing the cross entropy
 6: repeat
 7:     for g-steps do
 8:         Generate a sequence $Y_{1:T} = (y_1, \dots, y_T) \sim G_\theta$
 9:         for $t$ in $1 : T$ do
10:             Compute $Q(a = y_t; s = Y_{1:t-1})$ by Eq. (4)
11:         end for
12:         Update generator parameters via policy gradient Eq. (8)
13:     end for
14:     for d-steps do
15:         Use current $G_\theta$ to generate negative examples and com-
            bine with given positive examples $\mathcal{S}$
16:         Train discriminator $D_\phi$ for $k$ epochs by Eq. (5)
17:     end for
18:     $\beta \leftarrow \theta$
19: until SeqGAN converges
```

3.2 Architecture

The generator in SeqGAN is an LSTM while the discriminator is a CNN with max-over-time pooling and a highway architecture before the final fully connected sigmoid activation layer. In the two implementations we use, the first one with roll-out follows the architecture of the paper exactly. The second, simplified implementation uses a GRU RNN for both the generator and discriminator, and no rollout is used (a single reward is used for the entire sentence).

3.3 Procedure

In SeqGAN, synthetic training data is generated by an ”oracle” model (a randomly initialized LSTM) that can compute the likelihood of any sentence generated by the generator, thus providing a ”perfect” evaluation metric of NLL oracle loss to monitor training. We extend this approach to use real data; using our joke dataset, we pretrain the generator on this data and use the generator itself as the oracle. As the generator is trained on the real data, it can compute the loss of its own generated sentences and give us an instructional learning curve. We pretrain the discriminator after the generator on real and generated data. Then we begin adversarial training, where the generator and discriminator are trained alternately until convergence. Algorithm 1 shows the general procedure.

4 Training

We use a batch size of 32, max sequence length of 20, and learning rates of 1e-2 trained on NVIDIA Tesla V100 and P100. We run 5 different experiments (shown in Table 1) varying the adversarial training hyperparameters shown in the algorithm (g-steps, d-steps, and k) as well as pretraining hyperparameters.

5 Evaluation Metrics and Results

We used two types of metrics to evaluate the generated sentences in this project: ColBERT and BLEU. ColBERT(Annamoradnejad and Zoghi, 2021) is a neural network model that uses BERT to generate word embeddings as the input for hidden layers. ColBERT gives a likelihood of whether it thinks the input is a joke. This model uses parallel paths of hidden

Model	Generator Pretrain MLE Epochs	Discriminator Pretrain	Adversarial Training Epochs	Adversarial Training Parameters
1. With Roll-out	40	d-step=20 k=3	20	g-step=1 d-step=5 k=3
2. Simplified	40	d-step=20 k=3	20	g-step=1 d-step=5 k=3
3. Simplified	100	d-step=50 k=3	50	g-step=1 d-step=5 k=3
4. Simplified	100	d-step=50 k=3	50	g-step=1 d-step=1 k=10
5. Simplified	100	d-step=50 k=3	50	g-step=100 d-step=1 k=10

Table 1: Experiments

layers for different latent features contained in the entire line and sentence fragments. ColBERT is trained on a dataset with 160,000 jokes, half of which are jokes scraped from Reddit. ColBERT (Annamoradnejad and Zoghi, 2021) achieves 98.2% accuracy on its validation set. Thus we think it makes a good evaluation metric for the quality of our generated text. In the five experiments, only the trial with rollout had a relatively low accuracy and F1 score, and we believe that is due to a bug in the rollout implementation that causes mode collapse. Other than that, the quality of jokes seems to be good. However, it is possible that the high quality is contributed by the fact that ColBERT joke dataset shares the same origin as the Reddit dataset, which are used to train the discriminator and generator in SeqGAN.

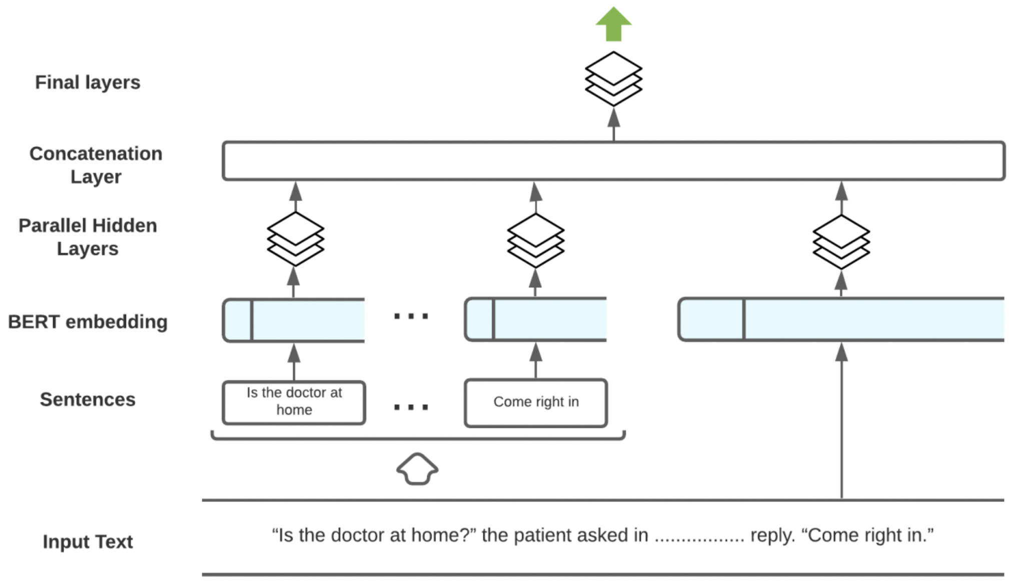
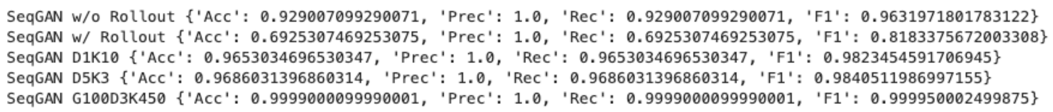


Figure 2: ColBERT Structure



```
SeqGAN w/o Rollout {'Acc': 0.929007099290071, 'Prec': 1.0, 'Rec': 0.929007099290071, 'F1': 0.9631971801783122}
SeqGAN w/ Rollout {'Acc': 0.6925307469253075, 'Prec': 1.0, 'Rec': 0.6925307469253075, 'F1': 0.8183375672003308}
SeqGAN D1K10 {'Acc': 0.9653034696530347, 'Prec': 1.0, 'Rec': 0.9653034696530347, 'F1': 0.9823454591706945}
SeqGAN D5K3 {'Acc': 0.9686031396860314, 'Prec': 1.0, 'Rec': 0.9686031396860314, 'F1': 0.9840511986997155}
SeqGAN G100D3K450 {'Acc': 0.9999000099990001, 'Prec': 1.0, 'Rec': 0.9999000099990001, 'F1': 0.999950002499875}
```

Figure 3: ColBERT Results

BLEU (Papineni et al., 2002) and Self-BLEU (Zhu et al., 2018) are two algorithms that are

commonly used as the originality and diversity score. BLEU score takes a list of reference words and a list of input sentence words, also known as hypothesis. It will find the n-grams that shows in both the reference list and the hypothesis list. Then it calculates the n-gram precision by dividing the n-grams that are showed in the hypothsis list by the total number of n-grams in the hypothesis. In this way, the BLEU score can test the to which extent the SeqGAN's outcome is different from MLE training samples and show the generated text's originality. Self-BLEU, building on top of BLEU, regards one sentence as hypothesis and the others in the generated text as reference, and calcualtes the BLEU score for every generated sentence.(Zhu et al., 2018) Then it takes the average BLEU score as the Self-BLEU of the input document. This metric can evaluate the diversity in the SeqGAN generated jokes.
In our experiment, the jokes generated by experiment with the parameters G100D3K450 is plagued by a high (generator update steps : discriminator update steps) issue, which leads to early convergences on the generator model. The outcome in this experiment is very homogeneous with only a dozen of unique sentences with similar starts. Therefore, we find its BLEU score to be high and Self-BLEU score to be low. The second experiment with rollout is the opposite. The model could not train stably thanks to mode collapse. This creates a highly diverse generated text with the lowest BLEU and Self-BLEU score. The third and fourth experiments produces the best jokes overall. This is also reflected by the BLEU and Self-BLEU scores.

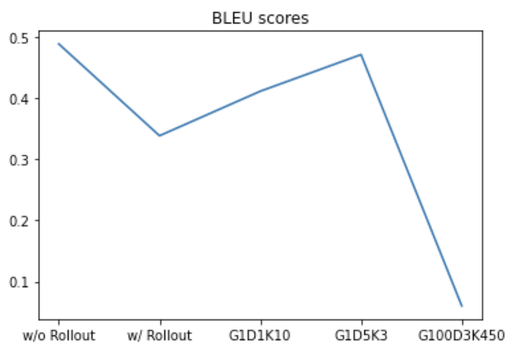


Figure 4: BLEU scores

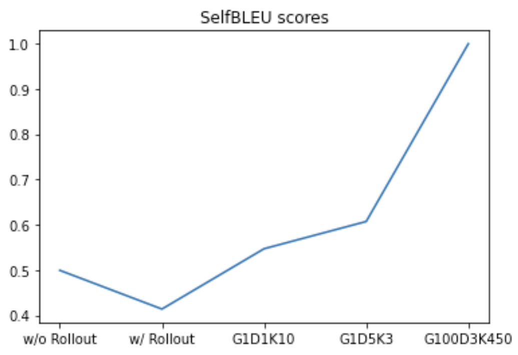


Figure 5: SelfBLEU scores

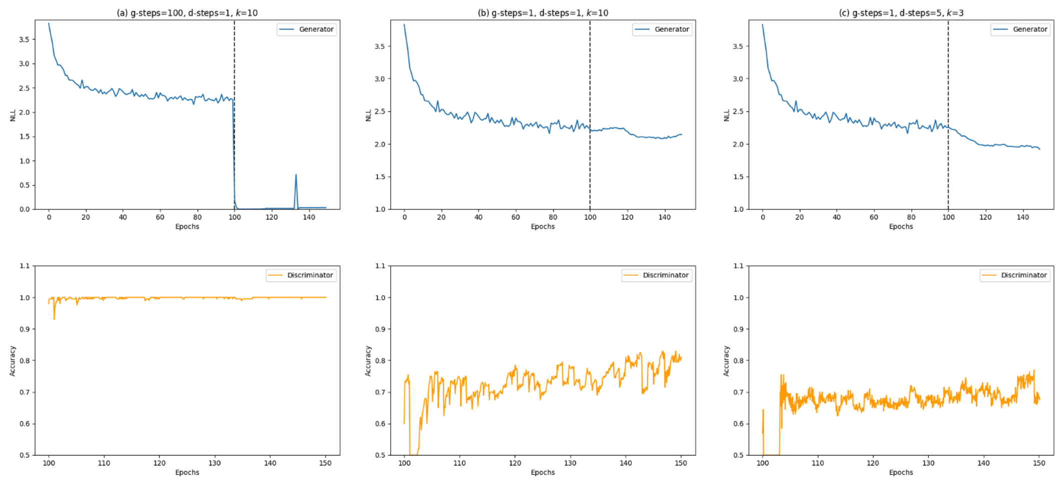


Figure 6: GAN NLL loss convergence performance with varying hyperparameters. Vertical lines indicate the start of adversarial training.

6 Discussion

The learning curves in Figure 6 correspond to experiments 3, 4, 5 where we vary the adversarial training hyperparameters. Column a) depicts likely a mode collapse that happened while training adversarially, and we know this because the loss of the generator goes quickly down to 0, while the discriminator accuracy stays high around 100%. Intuitively, this must mean that the generator is doing such a bad job at generating quality samples that the discriminator can very accurately tell them apart from the real data. The generated samples in figure 7 support this idea– you can see that all the jokes are merely identical, and thus as expected the self-BLEU score very weak because of poor diversity, which is a common feature of mode collapse.

In contrast, in column c) where we only train the generator for one step before updating the discriminator, we see a much more stable GAN training process. The discriminator accuracy stays relatively stable throughout, fluctuating around a reasonable number. This makes sense because we are updating the discriminator with more realistic negative examples before it can get fooled. The NLL loss of the generator shows a sharp decrease once adversarial training begins. Not surprisingly, the generated samples are both better in quality and diversity.

Overall, the best model hyperparameters when considering all of our evaluation metrics in addition to stable training performance is experiment 3 (g-step=1 d-step=5 k=3).

Figure 7: Samples generated from experiment 5 with g-step=100, d-step=1, k=10

```
request god spic spirits me . liabilitirary spit swap flying flying flying flying flying
request god spic spirits me . liabilitirary spit swap flying flying flying flying flying
request god spic spirits me . liabilitirary spit swap flying flying flying flying flying
request god spic spirits me me me . liabilitirary spit swap flying flying flying flying
```

Figure 8: Samples generated from experiment 3 with g-step=1, d-step=3, k=5

```
how many babies does it take a riot ? . blue because they are in 5
why is the race go to the pot ? . to the other
i hear my recently service delivers . no
the music joke about a canadian . german adjective irony . all the way .
what nickname is almond to a bookstore ? . newest mummy
how do hipster salmon ? . children .
```

7 Conclusion

In this project, we explored the SeqGAN, an approach for text generation that can be generalized to tasks like joke generation. We have shown that it is possible to train a GAN stably with SeqGAN's architecture on the humor dataset. We have also found a few metrics to evaluate the quality and originality of the generated jokes using ColBERT and BLEU. However, there remains to be many technical challenges coming with this project such as mode collapse, limited choices of humor dataset, as well as nonsensical generated jokes. One thing we can improve in the future is to use a Transformers network to replace LSTM and GRU RNN in the discriminator of SeqGAN. A better metric that can evaluate the level of humor will also be vastly helpful. Computational humor allows computer scientists to understand the real motive of a user, and we hope that this project sheds some light on its current development.

References

Issa Annamoradnejad and Gohar Zoghi. 2021. Colbert: Using bert sentence embedding for humor detection.

Fuli Luo, Shunyao Li, Pengcheng Yang, Lei Li, Baobao Chang, Zhifang Sui, and Xu Sun. 2019. Pun-gan: Generative adversarial network for pun generation. volume abs/1910.10950.

Surag Nair. 2018. Seqgan. GitHub.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2016. Seqgan: Sequence generative adversarial nets with policy gradient. volume abs/1609.05473.

Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, and Yong Yu. 2018. Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1097–1100.

ZiJianZhao. 2019. Seqgan-pytorch. GitHub.