TEXT GENERATION WITH GAN NETWORKS USING FEEDBACK SCORE

Dmitrii Kuznetsov

Department of Software Engineering, South China University of Technology, Guangzhou, China

ABSTRACT

Text generation using GAN networks is becoming more effective but still requires new approaches to achieve stable training and controlled output. Applying feedback score to control text generation is one of the most important topics in NLP nowadays. Feedback or response is a natural part of conversations and not only consists of words, but also can take other shapes such as emotions, or other reactions. In dialogue processes feedback is a factor influencing the next phrase or reaction. Depending on this feedback or response we correct our possible answers by trying to change the tone, context, or even structure of the sentences. Applying feedback as part of the GAN model structure will give us new ways to apply feedback and generate well-controlled outputs with defined scores which is very important in real-world applications and systems. With GAN networks and their instability in training and unique architecture, it becomes trickier and requires new ways of solving this problem. The matter of feedback usages for text generation task using GAN networks we will review in this paper and experiment with integrating score values into GAN's generator model layers.

KEYWORDS

Neural Networks, Text generation, GAN networks, Autoencoders, Controlled text generation

1. INTRODUCTION

Controlled text generation can be viewed as a separate topic of the natural language process and text generation area since it has its own difficulties and output structure. Applying machine learning techniques for AI-driven systems requires re-designing neural network architectures, techniques, and even datasets.

While text generation by itself is mostly sequence-2-sequence prediction, conversational text generation or sentiment-controlled text generation has important differences such as reactions to changing topics, emotions, and feedback from other participants. The generation of correct output also depends on a clear understanding of conversational parties of questions and responses. Even in natural human communication a misunderstanding often is a part of conversations.

In this work, we will apply feedback to the evaluation part of the GAN network and its generative model. For the experiment, we will simplify the feedback form and assume that conversational participants will have a simple way to evaluate text. That way we will use a score metric the same as e-commerce websites use in their review systems with a range of 1-5 where 1 is very bad and 5 is excellent.

1.1. Issues of GAN Architectures for Text Generation

GAN models can perfectly solve image generation or filtering tasks but in the case of text generation, it faces specific obstacles depending on the context and irregular length of the texts. Losing context over the time steps is a natural behavior of any generative model.

Usually, in n-gram text generation, we repeat N time steps in RNN (LSTM for example) to predict the next word in a sentence to create text. Latent vector ***z*** is the input hidden state *h0* and the generator output *G(z)* is the sentence. But here we not training RNN to minimize cross-entropy loss with respect to the target – we are training it to produce such output to make the discriminator think that sentence is "real" to minimize *1 – D(G(z)).*

When we are predicting the next word in RNN we are using the output of the softmax function, but this "next-word-generation" is not differentiable and we cannot do back-propagation, to get and calculate gradients of this operation of generating the next words.

This problem doesn't exist in the image type of data where generated data is continuous and can be passed directly from GAN's generator to the discriminator. Solutions for text generation tasks using GAN models are:

a. Applying reinforce learning algorithms and policy gradients, for example in the work "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient" [1]
b. The Gumbel-Softmax approximation (continuous approximation of the softmax function)
c. Produce continuous output of the generator (auto-encoders)

1.2. Related Works

In the paper "Wasserstein GAN" [2] authors proposed an alternative way to the GAN architecture and training process.

Two probability distributions on a certain metric space can be separated by the Wasserstein distance (also known as the Earth Mover's distance). It makes intuitive sense to think of it as the least amount of effort required to change one distribution into another. Work is defined as the sum of the mass of the distribution that needs to be transferred together with the distance that needs to be covered. The cost of the ideal transportation strategy is then the Wasserstein distance.

Jensen-Shannon divergence minimization is the original GAN goal, while Wasserstein distance provides the following advantages:

a. Since Wasserstein distance is continuous and almost differentiable everywhere, we may train the model to its best possible state. On the other hand, JS divergence has a vanishing gradients problem.
b. As the distributions move closer to one another and more apart, the Wasserstein distance diverges, making it a useful statistic.
c. Using Wasserstein distance in training is more stable than JS divergence.
d. In the case of Wasserstein distance, the "mode collapse" issue happens less often.

WGANs provide far more reliable training and a purposeful training goal.

In the work "Emotional Text Generation Based on Cross-Domain Sentiment Transfer" [3] authors using the autoencoders model converted original negative review text to positive text. This work demonstrated the ability of the model to extract patterns of the original text and reconstruct it with another emotional tone. Using autoencoders gives us new ways to use text representation in models' architecture.

2. APPLYING FEEDBACK FOR TEXT GENERATION TASK IN GAN

2.1. Structure and Types of Feedback

The most popular datasets for NLP tasks usually have some sort of customer feedback, usually is kind of score (as an integer value) or positive-negative value. It is very useful when we do sentiment analysis which helps us to predict the emotional value of the text.

This kind of feedback oversimplifies real-world scenarios, but it is suitable to build test cases and doing experiments.

Working with a single value describing the reaction of the text (or generated output) is becoming straightforward and can be easily applied to the neural network structure.

2.2. Applying Feedback to Discriminator's Loss Function

In the first case, we are using an approach to apply the reward to the discriminator during its process of calculating losses. The reward is a score given as feedback to the discriminator's loss function. This approach can work well when we need to generate texts with higher score values.

Discriminator (D) calculates 2 losses: one from real data and one from fake generated data. When the discriminator is trying to evaluate whether data is real or fake, with real data *y* is always 1. (*y*=1). That way we can calculate losses using binary cross-entropy:

$$L(1, \hat{y}) = -(1.log(D(x)) + (1-1).\, log(1 - D(x))) = -log(D(x))$$

Formula 1. Loss calculation for real data

Similar way we do losses calculation for evaluating fake data:

$$L(0, \hat{y}) = -(0.log(D(G(z))) + (1-0).\, log(1 - D(G(z)))) = -log(1 - D(G(z)))$$

Formula 2. Loss calculation for fake data

Combining losses, we get the total loss of Discriminator:

$$L(Discriminator) = -[log(D(x)) + log(1 - D(G(z)))]$$

Formula 3. Total GAN's discriminator losses

The main task of the GAN model is to minimize this loss value: *min[L(Discriminator)]*. We will apply the reward to the real data loss function. The main intention is to:

a. Have a bigger loss value if the reward is smaller than 1 and closer to 0 (0%)
b. Have a smaller loss value if the reward is closer to 1 (100%)

We apply feedback score to the discriminator as a factor, multiplying values according to:

R - feedback score as factor (in range 0…1, such as 0.9 = 90%, 0.5 = 50%, etc.), $0<R<1$
L = Loss of discriminator from real data training output (modulus):

$$L_{updated} = |L| * \frac{1}{R}$$

Formula 4. Updated discriminator losses

That way we are re-calculating ground true values before the actual losses' calculation. The final formula for calculating losses:

$$L(D) = -[log(D(x)) * \frac{1}{R} + log(1 - D(G(z)))]$$

Formula 5. Total discriminator losses

That way model will try to produce output that has a higher review score. This way of applying feedback factor simplifies the experiment to get initial results that can be used for future research.

Generator losses in GAN are calculated from discriminator losses. Need to mention that during the training of the generator, the discriminator is not updating its weights. The generator trying to fool the discriminator to classify fake data as real. That way generator loss function can be written as:

$$L(Generator) = min[log(1 - D(G(z)))]$$

Formula 6. Generator loss function

Combining losses from the generator and discriminator gives us the next formula for the calculation of total GAN losses:

$$L(GAN) = GminDmax[log(D(x)) * \frac{1}{R} + log(1 - D(G(z)))]$$

Formula 7. Generator loss function

It can be described as min / max game where the generator is trying to minimize the loss when the discriminator is trying to do the opposite: to increase losses.

This simplified approach works well when we intent to have one direction score value to reward or penalize the discriminator. But for controlled output of model with score differentiation such simple algorithm doesn't work.

2.3. Feedback Score Differentiation

To control model output and generate diverse texts depending on feedback score value we need to apply another approach.

For this task, we need to change the structure for the input layers of the generator model to be able to feed text and score vectors together to the model.

The generator as input gets some noise (latent space) and score value, and as output generates text. Then we feed the discriminator with those values along with the real text. Then we calculate losses usual way and update the discriminator and generator accordingly to the algorithm.

In this work, we are using a combination of autoencoders model along with GAN architecture.

The autoencoders model (AE) consists of 2 parts: the encoder which takes text embeddings as input and produces abstract text representation and the decoder part which take this text representation and outputs reconstructed text. In the AE model encoder consists of an embeddings layer and a bi-directional GRU layer. The output of the encoder has shape [b, s, u] where:

b – number of samples
s – the fixed maximum length of texts (with zero paddings)
u – number of hidden units

Decoder consists of an embeddings layer, a GRU layer, an attention layer (to prevent losing context over the time steps), and a normalization layer.

The input of our GAN network is text embeddings processed by the AE encoder part. That way we get abstract text representation to be able to feed the discriminator directly from the generator output.

This way our model scheme will look like this:

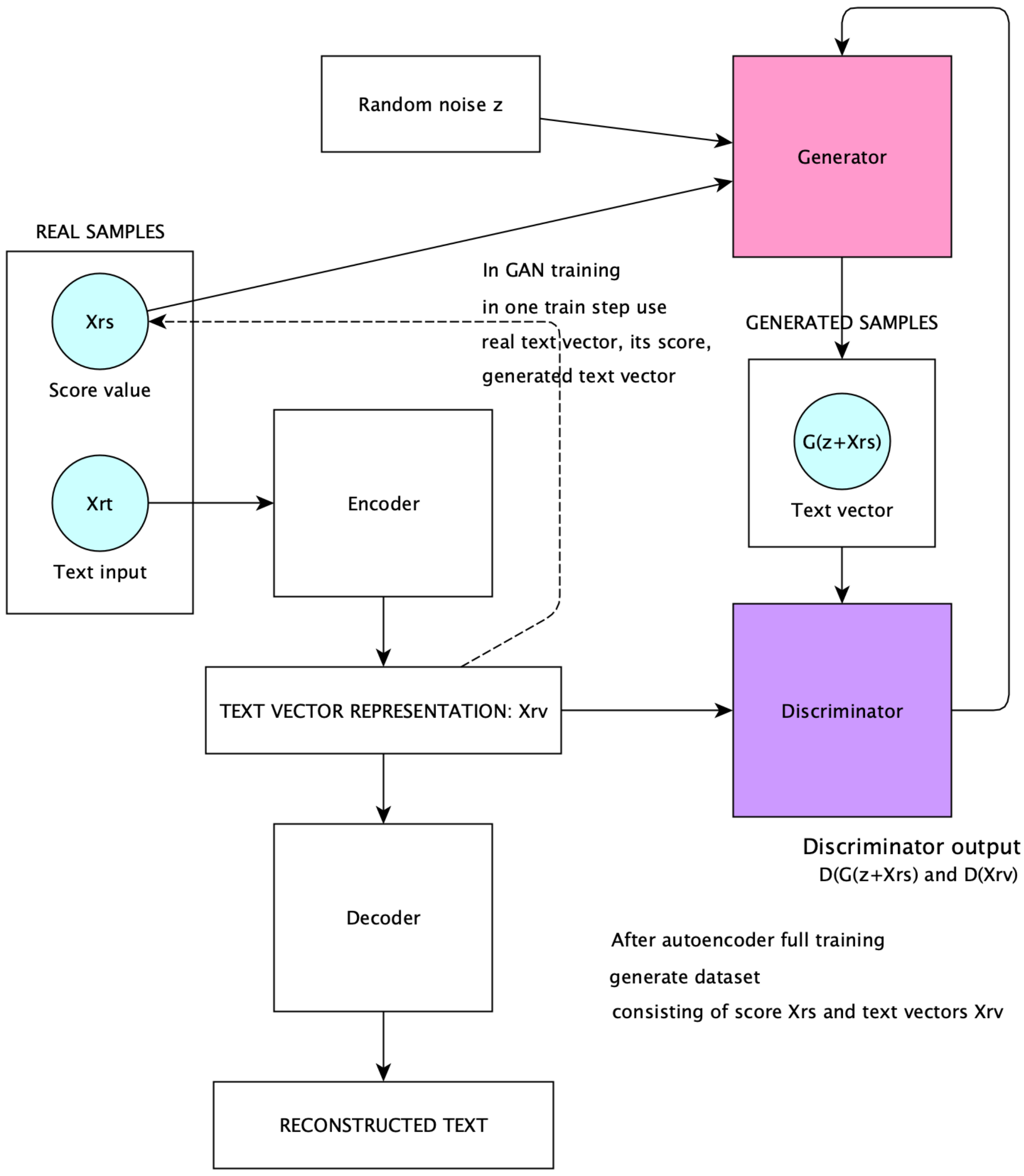

Figure 1: GAN model with an applied feedback score

Let's define components of algorithm as: *z* – generated noise input, *Xrt* – real text input, *Xrv* - real text vector processed by AE, *Xrs* – real feedback score value, *gp* – gradient penalty (from WGAN). Then total losses of GAN's model can be computed as:

$$L_{GAN} = G_{min}D_{max}[log(D(X_{rv}))) + log(1 - D(G(z + X_{rs})] + gp$$

Formula 8. Total GAN losses with applied feedback score

Need to notice that during GAN training we should use the same feedback score value for the generator's input as the real text has in this training step. For example, if in real sample text has a score of 3 then we pass to generator noise with this score value. After we get the generator output, we feed the discriminator real text which has a feedback score of 3 and generated a sample with the same score. It makes the discriminator effectively do its work.

Algorithm 1: GAN with score differentiation

Require: *AE* – Autoencoders, *AE Encoder* – encoder model of AE, *AE Decoder* – decoder model of AE, *GAN* – adversarial model, *G* – generator, *D* – discriminator, *z* – generated noise input, *Xrt* – real text input, *Xrv* - real text vector processed by AE, *Xrs* – real feedback score value, *gp* – gradient penalty, ae_num_epochs – training epochs for AE model, gan_num_epochs – training epochs for GAN model

```
1: repeat
2: for ae_num_epochs do:
3:     for batch in data do:
4:         pass text samples Xrt to encoder
5:         generate AE Encoder's output vector
           representation Xrv
6:         pass Xrv to AE Decoder
7:         get AE Decoder's output Yrt
8:         compute AE losses as: $Loss(Xrt - Yrt)$
9:         apply gradient descent
10:    end for;
11: until AE can produce the same output text as input
12: Converting text samples Xrt to vectors Xrv using trained AE Encoder
13: Combine text vectors and score values to GAN dataset: $(Xrv, Xrs)$
14: Start adversarial training with new dataset
15: repeat:
16: for gan_num_epochs do
17:     for batch in data do
18:         generate noise z and pass z and Xrs to G
19:         get G output as: $G(z + Xrs)$
20:         get D output 1 as: $D(Xrv)$
21:         get D output 2 as: $D(G(z + Xrs))$
22:         compute G losses as: $Loss(D(G(z + Xrs)))$
23:         calculate D losses as: $Loss(D(Xrv))$
24:         calculate discriminator total losses as: $Ltotal = -Loss(D(Xrv)) + Loss(D(G(z + Xrs)))$
25:         calculate gradient penalty (gp)
26:         add gp to D total losses: $Ltotal + gp$
27:         calculate D gradients apply its optimizer
28:         if train G on this step
                calculate G gradients
                and apply its optimizer
29:     end for
30: until GAN converges
```

3. EXPERIMENT

3.1. Dataset and Data Pre-Processing

For this experiment was used “Amazon Review Data” and its musical instruments reviews subset, which includes reviews with rating (score), texts, “helpfulness votes” and some metadata about the product.[1]

In this experiment, we use NLTK (Natural Language Toolkit) for the tokenization of the sentences to make the dataset suitable for processing by the model. The complete dataset has been tokenized, allowing the model to map each distinct token to a corresponding embedding vector.

At the beginning of the experiment were used Tensorflow’s tokenization module but during the work, this text pre-processing was optimized to work with NLTK and Gensim.

We use Gensim's word2vec function that trains a skip-gram model. After cleaning and removing rare words final training dataset contains about 35,505 reviews with a vocabulary size of 10,382 tokens. The maximum text length is 162 tokens.

3.2. Model

To make the model able to feed the discriminator directly from the generator output we are using Autoencoders representation of the sentence. That way training autoencoders before adversarial GAN training is a required step.

Since the generator and discriminator are fully connected networks, we create ResNet with similar layers for both GAN components.

To stabilize the training process, we use the gradient penalty algorithm. Gradient penalty set a restriction that requires the gradients of the discriminator's output relative to its inputs to have a unit norm. During training, gradient penalty requires to train the discriminator more iterations than the generator.

3.3. Autoencoders (AE) Training

After text pre-processing, we run autoencoders training. Autoencoders (AE) consist of encoder and decoder components. The main purpose of AE is to learn how to compress text into low-dimensional vectors, so we use Gated Recurrent Unit (GRU) layer for both components of autoencoders. Decoder will try to reconstruct a sentence from a latent vector and hidden state.

By utilizing an encoder network to condense information about each phrase into a finite vector, autoencoders are made to learn a low-dimensional representation of text. Reconstructing the input representation from the vector is the job of a decoder network.

The encoder's latent representation and the prior hidden state are inputs utilized by the decoder to create a probability distribution that is used to choose the word at that time step during sentence reconstruction.

[1] Code is available at: https://github.com/e8dev/scut_gan

Using trainable embedding layers to each AE helps to incorporate the pre-trained word embeddings from our skip-gram model. This algorithm used teacher forcing in order to assure a speedier and more reliable training process for the AEs.

For inference mode later we use the trained decoder to reassemble sentences from the output of the generator. As a result, the decoder is able to input its earlier forecast for the following time step. Inference mode is started by giving the generator's latent representation and the start of the sequence token (we encoded <Start> as the start token). Following that, the decoder output is projected onto the vocabulary size, and then we use a softmax operation to get tokens prediction. The next word in the sequence needs to be chosen from the resultant probability distribution until we get <End> token or text reaches its fixed maximum length.

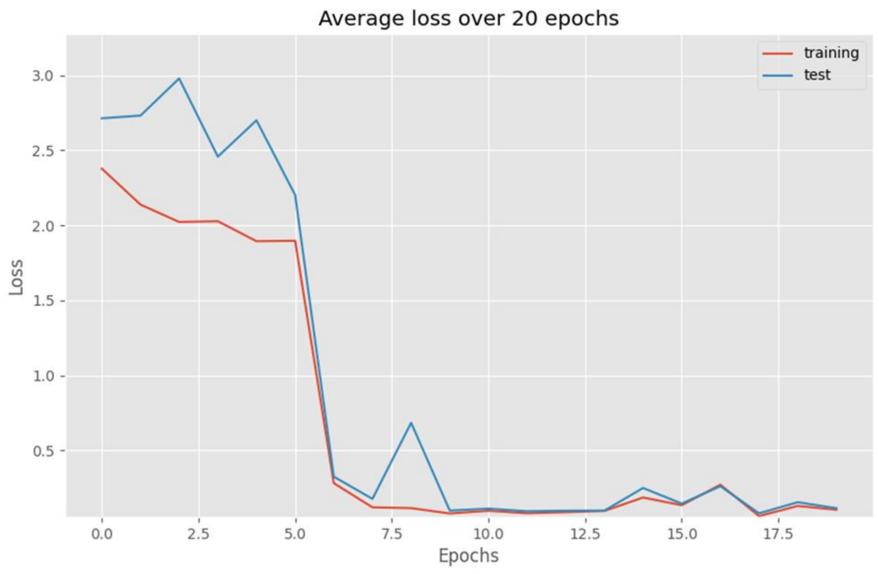

Figure 2. Autoencoders training losses

3.4. Adversarial Training for GAN with an Applied Score to Discriminator's Loss Functions

After AE training is completed and AE model is able to encode text to latent space and decode it back, we run adversarial training. Adversarial training heavily depends on the learning rate and has specific problems related to that such as vanishing gradients or mode collapse.

In the experiment, we chose the learning rate after a few training attempts and faced the vanishing gradient problem even using the Wasserstein space approach. With the learning rate of 0.001 and Adam optimizer, we faced a vanishing gradient problem, so we needed to change the learning rate to 0.0001 to have stable GAN training.

In most cases, we need GAN to generate a wide range of outputs. But during training, we faced a situation in which the generator produced the same result for each random input. Such a situation is when the generator learns to only generate that output to make the discriminator thinks is a real text called "mode collapse".

The discriminator's best way is to learn to consistently reject that output if the generator begins to consistently produce the same output. But, if the subsequent iteration of the discriminator is getting stuck in a local minimum and fails to identify the optimal way, it will be far too simple for the subsequent iteration of the generator to identify the discriminator's most likely result. Every generator's iteration overoptimizes for a certain discriminator, and the discriminator never learns how to escape this local minimum. The generators, therefore, cycle through a constrained number of outputs. Two of the best ways to overcome such a situation are:

a. Wasserstein loss – where discriminator learns to reject similar output of generator and generator forced to produce a new output.
b. Unrolled GAN – where generator loss function takes not only current discriminator output but also its future versions.

3.5. Results Evaluation

We trained the model using samples from the Amazon Reviews dataset. After training autoencoders and constantly decreasing losses autoencoder model started to produce reconstructed text correctly around the 20th epoch. During adversarial training discriminator and generator, losses started to converge around the 20th epoch but only after the 100th epoch with a learning rate of 0.0001 WGAN model started to produce output that can be qualified as satisfying. Since this epoch model was able to generate text which more looks like positive review text.

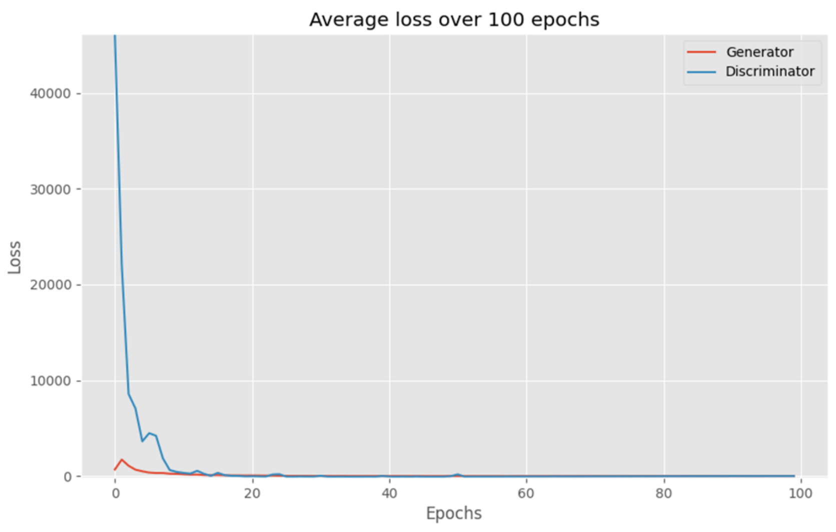

Figure 3. GAN training losses

Generated output examples (dataset is reviews about musical instruments):

“i have been playing guitar for <num> years and have had many different picks out there. these are great, and they are very easy to hold on.”

“i bought this for my <num> year old son. he loves it. it is a great little amp for the price.”

The model demonstrated the ability to produce output similar to human-generated text with a positive score. More training epochs with lower learning rates can increase the quality of a generated text. The sequence length of generated text like training dataset sample and overall satisfy requirements to output. Shorter sentences have better quality and are more predictable due to the shorter distance between words.

3.6. Adversarial Training with Controlled Output and Feedback Score Differentiation

For this work, we changed GAN structure and added an additional input layer to GAN's generator model. The generator takes the real score value and generates "noise" with the same shape as the text representation processed by AE's encoder. Generator's output and real text representation we pass to the discriminator, and it is calculating losses.

With a learning rate of 0.0001 model was able to generate text and get context and its score relation after 30-40 epochs. The model demonstrated the ability to control text output based on feedback score value.

At some moment model became not able to produce diverse output and started to generate the same texts again and again. This "mode collapse" happened when the generator finds data that makes the discriminator to be fooled easily. This can be avoided by implementing additional metrics and applying them as reward/penalty to the discriminator's loss function.

Table 1. Results evaluation

Generated text	**Real text to compare**	**BLEU score**	**BERT score (F1)**
i bought this for my daughter who is just starting out learning how to play . it is a great learners guitar , but for the price , it is a wonderful ukulele .	i bought this set for my husband and he is extremely happy ! he loves that it hooks onto his drums so he can grab one really fast if he drops one . the bag is amazing ! its very good quality nd all of the sticks match and are very nice . it couldnt believe how cheap they were when i told him !	0.0195	0.8584
i bought this for my daughter who is just starting to play guitar . she loves it . it is very light and is very sturdy .	i always use elixirs for my electric and acoustic guitars . they feel awesome on your fingers , sound smoother and last longer than any other string ive ever used in about <num> years of playing . the pick that comes with this set are junk though , just toss them . i know theyre expensive but sometimes you get what you pay for .	0.00584	0.8489
great product . great packaging . great price . great people to do business with.mahalo nui loa (hawaiian : thank you very much) for fulfilling your obligation in a glorious way.cheers	i bought this set for my husband and he is extremely happy ! he loves that it hooks onto his drums so he can grab one really fast if he drops one . the bag is amazing ! its very good quality nd all of the sticks match and are very nice . it couldnt believe how cheap they were when i told him !	0.00308	0.8087
this is a great little dac for enabling me to any music stand . it is light enough to hold a lot of keyboards and it works great .	these strings are really great ! martin strings have a nice tone to them and are also pretty sturdy . the tone is mellow and it doesnt take too much pressure to make a note . not to mention the fact that it is a three pack and in the event that one string breaks	0.03080	0.8494

	you have two more to choose from . this is a great buy !		
i use this for my recording studio . it is a great mic for the price . it is very easy to use and a great price .	i ordered this for a parts bass build i was doing for a client . its a good thing that i checked the thickness of the bass headstock and the length of the supplied screw . the screw is way too long and would have come out on the back side of the headstock .	0.0078	0.8530
i have used this pedal for over <num> years and it has been a great price . it is easy to use and has a nice sound .	it was exactly what i was looking for . perfect fit for a usa stratocaster <num> hole guitar . came packed in a nice sturdy flat packaging . wasnt warped which is usually what happens when you buy a pick guard online .	0.0279	0.8546
i have used this pedal and it is a great addition to my pedal board . it does exactly what it is supposed to do . it is a classic distortion and a dirty ac booster with the drive .	picks with no texture or friction enhancement are old technology . these pick are cutting edge . none better . possibility created by the multi flex feature and the grip . awesome . these make me a better player . i used to have trouble holding on to picks . problem solved . stress gone .	0.00984	0.8454
i have been playing guitar for <num> years and have used many different types of strings over the years .	great feel to them and nice sounding . they do seem to stay in tune a bit longer but then they are thicker than what i normally use . i think i will try these in the super slinky	0.010070	0.8399
i have used elixir strings for years and they are great	this is a nice pedal . appears to be built good in a rugged case . i like to use just a touch of phase to make my guitar growl and this does the job nicely ! cant beat it for the price !	0.003086	0.8499
this is a great little dac for enabling me to any music stand . it is light enough to hold a lot of keyboards and it works great .	ive used these strings for years and the tone and quality are top notch . its good to experiment with others brands as well , but you really cant lose with these babies .	0.006337	0.8375

4. CONCLUSION

We successfully applied feedback score for text generation and got promising results in its simple form and with controlled output. Our model can produce text reflecting the original feedback score.

In table 1, we can see the results of generated samples and measured BLEU and BERT scores. The BERT algorithm has a better ability to catch context without relying on sample length which gives better results. Most of the generated samples have about 0.85 BERT score. BERT algorithm correctly recognizes text context (in our case musical instruments) and gives a better score.

Generated texts are very well structured and show great potential to improve results in future experiments. The abstract idea behind this feedback application can be used not only in conversational or text-generation tasks but also in other areas.

4.1. Problems and Limitations

The model has tended to generate repetitive samples after some period of training. This “mode collapse” is another problem that should be solved in future works by increasing the size of the dataset or implementing additional metrics. In 2018 Richardson and Weiss [4] proposed a new method named Number of “Statistically-Different” bins (NDB). This metric can be used to show how generator sampling noise differs from defined fixed noise. Calculating this NBD metric during training gives us the ability to reward or penalize GAN losses which makes the generator produce more diverse output.

5. FUTURE RESEARCH

One of the most important parts of the conversational text is feedback because depending on the response we can build the next part of the conversation and make a logical sequence. The subjects of future research should become feedback and its types and forms. The feedback that the model can use to score and evaluate answers can be not only represented as some integer value but can take different shapes and types.

For example, those responses can be text messages, reading emotions (or reactions) from videos and pictures, and evaluating responses accordingly, or they can be sensors’ signals from medical devices. Future works can include the classification of such responses and each of these types requires detailed work to make a correct interpretation of them.

Although GANs are relatively new in the field of machine learning, they have achieved outstanding success in several areas, such as computer vision. It was successful in the field of sequence production as well such as text generation, and this work's findings show different ways to apply feedback in text generation tasks using GAN networks.

REFERENCES

[1] Lantao Yu, Weinan Zhang, Jun Wang, Yong Yu, (2017) “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient”, arXiv:1609.05473v6 [cs.LG].

[2] Arjovsky, Martin, Soumith Chintala, and Léon Bottou, (2017) “Wasserstein generative adversarial networks”, International conference on machine learning. PMLR.

[3] R. Zhang, Z. Wang, K. Yin, Z. Huang, (2019) “Emotional Text Generation Based on Cross-Domain Sentiment Transfer”, IEEE Access, vol. 7, pp. 100081-100089, 2019, doi: 10.1109/ACCESS.2019.2931036.

[4] Eitan Richardson, Yair Weiss, (2018) “On GANs and GMMs", arXiv:1805.12462 [cs.CV].

[5] Tuan, Yi-Lin, and Hung-Yi Lee, (2019) “Improving conditional sequence generative adversarial networks by stepwise evaluation”, IEEE/ACM Transactions on Audio, Speech, and Language Processing. 2019. 27.4. 788-798.

[6] Salakhutdinov, R., (2009) “Learning deep generative models”, Ph.D. Dissertation, University of Toronto.

[7] Silver D., Huang A., Maddison C. J., Guez A., Sifre L., (2016) “Mastering the game of go with deep neural networks and tree search”, Nature, 529(7587), p. 484–489.

[8] Srivastava, N., Hinton G. E., Krizhevsky A., Sutskever I., and Salakhutdinov, (2014) "Dropout: a simple way to prevent neural networks from overfitting", JMLR, 15(1):1929–1958.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, (2017) "Attention is all you need", arXiv:1706.03762 [cs.CL].

[10] Donahue, David, and Anna Rumshisky, (2019) "Adversarial text generation without reinforcement learning", arXiv preprint, arXiv:1810.06640

[11] Jianmo Ni, Jiacheng Li, Julian McAuley, (2019) "Justifying recommendations using distantly labeled reviews and fined-grained aspects", Empirical Methods in Natural Language Processing (EMNLP).

[12] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, (2016) "Improved techniques for training gans", NIPS

[13] Bengio, Y., Yao, L., Alain, G. and Vincent, P., (2013) "Generalized denoising auto-encoders as generative models", In Advances in Neural Information Processing Systems, Vol. 1, p. 899-907.

[14] S. Pragaash Ponnusamy, Alireza Roshan Ghias, Chenlei Guo, Ruhi Sarikaya, (2008) "Feedback-Based Self-Learning in Large-Scale Conversational AI Agents", arXiv:1911.02557v1 [cs.LG].

[15] Matthew Henderson, Paweł Budzianowski, Iñigo Casanueva, Sam Coope, Daniela Gerz, Girish Kumar, Nikola Mrkšic, Georgios Spithourakis, Pei-Hao Su, Ivan Vulic, and Tsung-Hsien Wen, (2019) "A Repository of Conversational Datasets", arXiv:1904.06472 [cs.CL].

[16] Linqing Liu, Yao Lu, Min Yang, Qiang Qu, Jia Zhu, Hongyan Li, (2017) "Generative Adversarial Network for Abstractive Text Summarization", arXiv:1711.09357 [cs.CL].

AUTHORS

Dmitrii Kuznetsov is a Master's student from the South China University of Technology who currently focuses on his research on NLP and generative neural networks. Having extensive experience in software engineering, he applies practical knowledge to scientific tasks.