Generating Text using Generative Adversarial Networks and Quick-Thought Vectors

David Russell and Longzhuang Li
Dept. of Computing Sciences
Texas A&M Uni.-Corpus Christi
Corpus Christi, TX, USA 78412
e-mail: drussell3@islander.tamucc.edu

Feng Tian
System Engineering Institute
Xi'an Jiaotong University
Xi'an, Shanxi, China 710049
engtian@mail.xjtu.edu.cn

***Abstract*—Generative Adversarial Networks (GANs) have been shown to perform very well with the image generation tasks. Many advancements have been made with GANs over the past few years, which are making them more and more accurate in their generation tasks. State-of-the-art methods of natural language processing (NLP) involve word embeddings such as global vectors for word representation (GLoVe) and word2vec. These word embeddings help apply text data to a neural network by converting the textual data to numbers that the networks could use. The main focus for GANs has been image generation and in the past few years there have been research works to apply GANs to the text generation task. This paper presents a Quick-Thought GAN (QTGAN) to generate sentences by incorporating the Quick-Thought model. Quick-Thought vectors offer richer representations than prior unsupervised and supervised methods and enable a classifier to distinguish context sentences from other contrastive sentences. The proposed QTGAN is trained on a portion of the BookCorpus dataset that has been converted to Quick-Thought embeddings. The embeddings generated from the generator are then classified and used to pick a generated sentence. BLEU scores are used to test the results of the training and compared to the Skip-Thought GAN. The increases in BLEU-3 and BLEU-4 scores were achieved with the QTGAN.**

Keywords-generative adversarial network (GAN), word vector, word embedding, quick thought vector, skip thought vector, sentence embedding, natural language processing (NLP)

I. INTRODUCTION

Deep learning has had many advances over the years and one of those is the generative adversarial network (GAN). GANs are a framework that use two neural networks that are going against each other. These two networks are the generator and discriminator. The goal of the GAN is to get the generator to fool the discriminator. This will allow the generator to produce samples that should be acceptable for use in whatever it was aimed at. In the deep learning, images and sound are transformed into high dimensional tensors that allow them to be mathematically represented. To generate text, words also need to be able to be represented by embedding vectors using methods like word2vec [14] and GLoVe [15]. This allows a neural network to be able to use those word embeddings in tasks that relate to natural language processing (NLP).

Skip-Thought model [12] is a method of embedding sentences into vectors for use in NLP. Skip-Thoughts encode the whole sentences and the created vector holds the context of a sentence. This means that entire sentences can be used to classify a context of the text instead of just words individually. The decoder for the sentence embeddings uses a greedy approach to build the sentence. It creates a sentence from scratch one word at a time. This can be costly and may not always lead to a great result. Skip-Thought vectors has been utilized in STGAN to generate text with good BLEU scores [4]. Quick-Thought vectors are the new method of sentence embedding and encode similarly to the Skip-Thought model [11]. The Quick-Thought model can train faster and produce better results. When the embeddings are decoded it does not generate a sentence word by word as Skip-Thought. Instead it classifies that embedding and grabs the best matching pre-existing sentence from some candidate sentences. The Quick-Thought vector model is incorporated in the QTGAN proposed in this paper.

The rest of paper is organized as follows. In Section II, related works in NLP and GANs are reviewed. In Section III, a new Quick-Thought GAN (QTGAN) is proposed. In Section IV, the proposed QTGAN is trained by using the BookCorpus dataset, and the performance is compared with the STGAN. In Section V, the future works are discussed. In Section VI, we conclude the paper.

II. RELATED WORK

A. Natural Language Processing

NLP is a field in Computer Science that deals with how computers process human languages. This field focuses on research topics including speech recognition, language translation, and text generation. Speech recognition allows programs to be able to hear and understand human's voice commands, languages translation targets on converting text from one language to another, and text generation leverages knowledge in computational linguistics and artificial intelligence to automatically generate natural language texts. This work falls into the subfield of natural language generation or text generation.

One important technique in NLP is word embeddings, which are a type of word representation that allows words with similar meaning to have a similar representation of a dense vector of real numbers. Since word2vec [14] and GloVe [15] were developed to obtain word embeddings,

978-1-7281-2871-9/19/$31.00 ©2019 IEEE

other methods, such as Skip-Thought embeddings [12] and Quick-Thought embeddings [11], were proposed to perform better in practice.

B. Generative Adversarial Networks

Generative Adversarial Networks (GAN) were first introduced by Goodfellow et al. in [10]. GANs are made up of two neural networks. One called the generator and the other called the discriminator. The goal of a GAN is to get the generator network trained enough to be able to create good samples that appear to be authentic to the discriminator. The generator and discriminator play what is called a minimax game. The discriminator wants to maximize the probability that it labels the data correctly while the generator wants to minimize the probability that the discriminator will detect the fake samples. GANs are primarily used for computer vision tasks such as generating a picture of a certain animal after it has been trained or even turning one animal into another like a horse to a zebra. GANs have excelled in the computer vision field and there has been a lot of work done in the NLP field. This paper implements a GAN to be trained on Quick-Thought embeddings to generate text.

1) Generator: The generator often referred to as *G* of the GAN is what is being trained to create whatever the task is. Whether that be images, music or text which is what the work aims at creating. The generator takes in a random noise vector *z* and generates a sample from that random noise vector. The goal is to be able to create samples that are indistinguishable from real samples. In some implementations the generator is trained slightly faster than the discriminator to give the generator a slight edge in training. This prevents the discriminator from instantly being able to tell that the samples are fake.

2) Discriminator: The discriminator often referred to as *D* of the GAN is what is being trained to classify whether or not the sample it receives is real or fake. In addition to the vector it takes in an input that tells it whether or not the current input is real or fake. In text generation, The discriminator takes a vector as input. The vector is either a real piece of data or a generated piece of data.

3) Training Loss Function: The vanilla loss function for a GAN as shown in Equation 1 is defined as the discriminator maximizes the probability that a sample is correctly classified, and the generator minimizes the probability that the discriminator will be correct [10]. This is applying gradient ascent on the discriminator and gradient descent on the generator. The loss function is shown below in Equation 1.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data(x)}}[logD(x)] + \mathbb{E}_{x\sim P_{data(z)}}[log(1-D(G(z)))] \quad (1)$$

However there exists a problem with this function for the generator. The gradient signal for the generator will be small when it is generating bad samples and the discriminator is easily catching it. Then when it starts generating good samples and fooling the discriminator it will have a much larger gradient signal. This means that it is learning more when it is already generating good samples than it was when it was generating bad samples. The opposite scenario is desired. The generator should learn more when generating bad samples than it does when generating good samples. To do this the loss function is changed for the generator to maximize the probability of the discriminator being incorrect. Equation 2 shows the change to the generators function.

$$\max_G \max_D V(D,G) = \mathbb{E}_{x\sim P_{data(x)}}[logD(x)] + \mathbb{E}_{x\sim P_{data(z)}}[log(D(G(z)))] \quad (2)$$

C. TextGAN

One example would be TextGAN from [13] that uses a GAN to generate text. The generator of the TextGAN is made of a Long-Short-Term-Memory (LSTM), and the discriminator is made of a Convolutional Neural Network (CNN). The TextGAN was trained using a random sample of 1 million sentences from BookCorpus and 1 million sentences from the ArXiv dataset. The goal was for the GAN to learn both informal and scientific writing. The result showed that a GAN can be used for text generation.

D. SeqGAN

SeqGAN proposed in [1] is another GAN that is used for the text generation task. It aims to solve two problems that GANs face in text generation. One being that it has difficulties producing discrete tokens and the other problem being that the GAN scores its performance after the entire sequence has been generated. The model was trained using Chinese poems, political speeches from the former US President Obama and music samples. The SeqGAN model performed well with its implementation in generating Chinese poems, political speeches and music.

E. Skip-Thought GAN (STGAN)

Skip-Thought vectors as proposed by [12] are a framework that uses the encoder-decoder model. The encoder takes the words of a sentence and encodes a sentence vector instead of encoding each word embedding like word2vec or GLoVe. The Skip-Thought sentence embedding is then decoded using a greedy algorithm for choosing the next word in the sentence. This means that the Skip-Thought model constructs a brand-new sentence for each embedding which can be computationally costly. The STGAN is one of many GANs in [4] that make use of the Skip-Thought Encoder-Decoder models. These models include the vanilla GAN, a GAN with gradient policy updates, as well as a wasserstein distance version. The wasserstein distance is an alternative loss calculation that is aimed at improving the training. The model is trained using sentences from the BookCorpus dataset that are encoded

using the Skip-Thought encoder. These encodings are passed to the discriminator for training. The generator also makes samples for the discriminator to evaluate. Once training of the STGAN is complete the generator will produce Skip-Thought sentence embeddings to be decoded by the Skip-Thought decoder, which will generate new sentences based on the embedding.

III. PROPOSED WORK

In this paper, the Quick-Thought GAN (QTGAN) is proposed, which is mostly close to the STGAN. The QTGAN is trained using sentence embeddings from a different model name Quick-Thoughts. Quick-Thoughts are the newer method of sentence encoding providing better results on generating sentences. Once the training of the QTGAN has completed then it will be used for generating sentence embeddings and then decoding those embeddings into sentences using the Quick-Thought decoder.

The Quick-Thought GAN is implemented based on a standard GAN model that is shown in [10]. The architecture of the QTGAN can be seen in Figure 1. There are four major components making up the architecture, which are the Quick-Thought Encoder, Generator, Discriminator and the Quick-Thought Decoder.

A. *Quick-Thought Vectors*

Quick-Thought vectors as proposed by Logeswaran & Lee in [11] improve on the Skip-Thought framework. The Quick-Thought vector framework uses the same encoder-decoder model. The encoder takes a sentence and encodes it into a vector just like the Skip-Thought framework. The difference is in the decoder. The decoder of the Quick-Thought framework does not try to generate the words of the sentences around the encoded sentence. Instead the decoder will classify the sentence embedding and pick a sentence from a selection of candidate vectors to use. This saves computational power as it works directly in the embedding

vector space instead of using a greedy algorithm to generate sentences one word at a time like the Skip-Thought model. Quick-Thought vectors will be used in this GAN. The loss function for the GAN is the improved loss function shown above in Equation 2.

B. *Quick-Though Encoder*

The Quick-Thought model being used for the encoder is the model from [11]. The model is inspired by the multi-channel CNN model in [7]. The model is a concatenation of two bi-directional recurrent neural networks (RNNs). One of the RNNs uses a large vocabulary of around 3 million words while the other uses a much smaller vocabulary of about 50 thousand words. The model uses GLoVe to kick start its training. The model is shown to produce lower amounts of redundant features than the bi-directional model discussed in [12].

C. *Quick-Thought Decoder*

The Quick-Thought decoder differs from the Skip-Thought decoder in that it does not actually generate a new sentence. Instead it looks for the best sentences that fit the embedding and then chooses the best out of those. So the decoder uses a set of sentence embeddings from the training corpus. In other words, the decoder classifies the current embedding and then finds candidate sentences for that embedding. The candidate with the highest probability is chosen as the new sentence (see Equation 3). For a given sentence position in the context of s, the probability that a candidate sentence $s_{cand} \in S_{cand}$ is the correct sentence for that position is computed in Equation 3 as follows:

$$p(s_{cand}|s, S_{cand}) = \frac{exp[c(f(s), g(s_{cand}))]}{\sum_{s' \in S_{cand}} exp[c(f(s), g(s'))]} \quad (3)$$

where c is a scoring function, f and g the parametrized functions that encode an input sentence into a fixed length vector, and S_{cand} a set of candidate sentences. This is where the computational cost is saved. It works directly in the vector space and does not generate its own sentences. Instead it works with pre-existing sentences from the corpus and allows quick results of sentence embedding decoding.

D. *Generator*

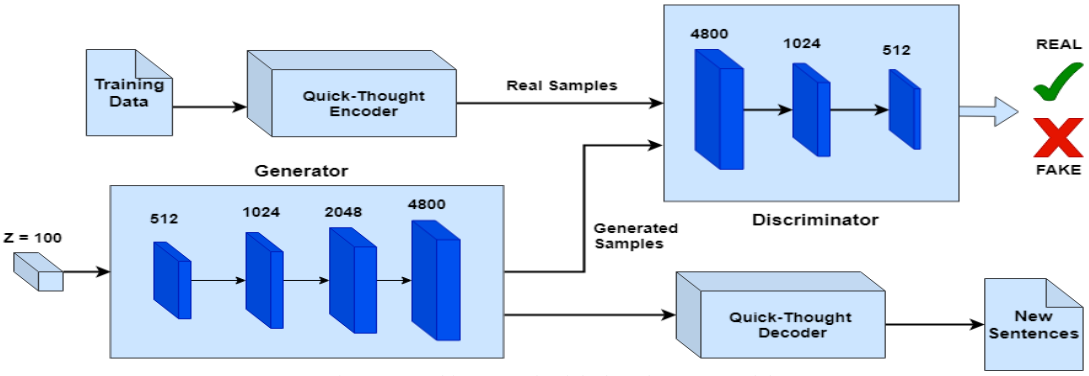

Figure 1. Architecture of quick-thought GAN model

The generator of the Quick-Thought GAN is based on the GAN from [10]. The generator takes in as input a random noise vector z. The length of the input vector z is 100. The generator is made of four densely connected layers. The first layer takes the vector of 100 as input and expands it to 512. The second layer expands the 512 to 1024. The

third layer will expand it to 2048. These first 3 layers use LeakyReLU as their activation functions with an alpha of 0.2. They also use batch normalization with a momentum of 0.5. The final layer is the layer which expands the 2048 to 4800 which is our final size that matches the word embeddings of the Quick-Thought model. This final layer uses a tanh activation function. These samples that the generator creates is similar to the real embeddings from the Quick-Thought model. The samples are then sent to the discriminator during training for evaluation. Once the training is complete then the generator is able to generate embeddings for use in the Quick-Thought decoding part of the system, which produces newly generated sentences from the generated sentence embeddings.

E. Discriminator

The discriminator is used for checking whether or not the input it receives is real or fake. The input for the discriminator is a 4800-dimensional vector that represents a sentence embedding. It also takes in another input that represents whether or not the provided embedding is real or a fake sample from the generator. The discriminator evaluates the samples it receives and determine the authenticity of them during training of the generator. It does this using 3 densely connected layers. The first layer shrinks the vector from 4800 to 1024, and the second layer shrinks it from 1024 to 512. The first and second layers use LeakyReLU as the activation function like the generator with an alpha of 0.2. The final layer outputs a value from 0 to 1 using a sigmoid activation function. This value represents whether it thinks the sample is real or fake and is used in the loss calculation as well. Once the training of the generator is complete then the discriminator has served its purpose and is no longer used.

IV. EXPERIMENTS

The experiments in this paper were done using Google Colab's services. A Python 2 run time was used with GPU acceleration. This provided easy to use method of running code in a GPU accelerated environment. The run time provides ~12GB of RAM and ~350GB of storage space. The GPU provided by Google Colab is a Nvidia Tesla K80.

A. Dataset

The dataset used in the experiments is the BookCorpus dataset [16], which is made up of free books from unpublished authors. It contains 16 different genres and about 45 million sentences. This dataset is used in a lot of text generation works for training so it will be used for training this GAN as well. This dataset is no longer publicly distributed so in order to obtain it you have to use a script to scrape the data from the smashwords website which is where the books in the BookCorpus dataset come from. In order to do this, we used code from a GitHub repository by Soskek [1] This repository contains a snapshot of the BookCorpus dataset and give you the ability to download the dataset in full. Then perform needed actions such as making the dataset one sentence per line and tokenizing if necessary. Due to the RAM limitations in Colab using the Quick-Thought encoder only 64K sentences were used from the dataset. The Quick-Thought model being used is the pre-trained MC-QT model from [11].

[1] https://github.com/soskek/bookcorpus

B. Training

As previously mentioned, the training was done on a Google Colab Python 2 GPU instance using the pre-trained model from [11]. The generator takes in a random noise vector z of dimension 100, and outputs a sample vector that has a dimension of 4800 since Both Skip-Thought and Quick-Thought bi-directional encoders produce embeddings with a dimension of 4800. The training uses batch sizes of 128 with 500 batches. This gives a total of 64,000 training examples to use. Training was done for 5001 iterations for a total of 10 epochs. Going above 10 epochs sometimes leads to undesirable results. After each epoch in the training session the generator produces 50 samples for evaluation. In addition to this it also saves the states of the discriminator and generator models so they can be loaded later for more training. During the training the discriminator receives samples from the real embeddings and the generated embeddings along with labels to tell it if they are real or fake. The loss function used is Equation 2 that was shown above.

C. Results

The bilingual evaluation understudy BLEU scores are used to compare the performance of QTGAN, STGAN and STGAN minibatch. The BLEU scores check to see how much the candidate sentence which in this case is the generated sentence resembles the sentences in the corpus. The BLEU-2 is to check with groups of two words at a time and see if those two words appear in the corpus in the same order and scores from that. The BLEU-3 and BLEU-4 are very similar except they use groups of 3 and 4 respectively.

The scores for BLEU-2, BLEU-3 and BLEU-4 are shown below in Table I. The BLEU scores show an improvement when using the Quick-Thought model in all but BLEU-2 for the STGAN minibatch model. To be more specific, the BLEU-2 score of QTGAN is 0.736, which is higher than that of STGAN and lower than STGAN minibatch. For both BLEU-3 and BLEU-4 scores, QTGAN achieves 0.684 and 0.771, respectively, while STGAN offers 0.564 and 0.525 and STGAN minibatch 0.607 and 0.531. The examples of decoded sentences are shown in Table II.

TABLE I: BLUE SCORES FOR 50 SAMPLES AVERAGED

BookCorpus Reference			
Model	BLEU-2	BLEU-3	BLEU-4
STGAN	0.709	0.564	0.525
STGAN (minibatch)	0.745	0.607	0.531
QTGAN	0.736	0.684	0.771

TABLE II: THE EXAMPLES OF DECODED SENTENCES

Example Sentences
“An all-inclusive wakeup call into collective hive greater good action.”
“He means is there anything in his life that could help us now.”
"Almost out of nowhere, a figure emerged and started towards Ray."
"We had run from so much danger in that car, but now I had to keep dealing with boring politics."
"I wish Phoenix or even Elathan would come back and just… fix everything.”
“Kenneth, can you help us find your brother?”

V. FUTURE WORK

The proposed QTGAN can be expanded and changed in many ways for better results. The loss function currently used is the original loss function from Equation 2. This can be changed to something like the wasserstein distance function in order to improve the loss function. The wasserstein distance provides a better insight during the training because it converges better which GANs tend to have issues doing. Another addition that could be done is a change to the generator. Currently the generator is not conditioned on any input. It just receives its random noise vector and generates a sentence embedding based on that random vector. Instead the conditional GAN model can be implemented to change this [6],[8],[9]. The conditional GAN model allows the generated sample to be conditioned on an input. Once a sentence is encoded the sentence embedding y can be concatenated to the random noise vector z and then becomes the input to the GAN's generator. This will produce a sample that is conditioned on the input y. For example, if a sentence about having a party is entered in the conditional GAN, then the sentence that is decoded from the generated embedding should have something to do with having a party. Once conditional sentences can be generated then another addition could be making sure that the future generated sentences are related to each other. In this way, multiple sentences could be produced out of multiple encodings, which allow the model to write a complete short story [2],[3],[5]. More future work would be expanding the training data to be more of the dataset. Because of the limit imposed by Google Colab only a small portion of BookCorpus was used. It would be better to be able to use a larger amount of BookCorpus.

VI. CONCLUSION

In the paper, the QTGAN is proposed and implemented by incorporating the Quick-Thought encoding and decoding models into it. The BookCorpus dataset was used as the training data and encoded using the Quick-Thought encoder. During the training, the QTGAN used these encodings of the sentences as input for the discriminator to help the generator learn and produce better embeddings. It was shown that the QTGAN model does perform well after training and have the ability to create Quick-Thought sentence embeddings. Decoding these embeddings and using the BLEU scoring system allows a comparison between the QTGAN and the STGAN. This comparison showed that the QTGAN had better results than the STGAN. With higher scores in all tests but the BLEU-2 against the STGAN that used minibatches. This shows that the QTGAN is capable of producing good text results that are on par if not better than the STGAN. In conclusion a GAN can be trained on quick thought encodings and be able to produce good results for text generation.

REFERENCES

[1] L. Yu, W. Zhang, J. Wang and Y. Yong, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient", in 31st AAAI Conference on Artificial Intelligence, 2017.

[2] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu and J. Wang, “Long Text Generation via Adversarial Training with Leaked Information”, in 32nd AAAI Conference on Artificial Intelligence, 2018.

[3] A. Fan, M. Lewis and Y. Dauphin, “Hierarchical Neural Story Generation”, in 56th AMACL, 2018.

[4] A. Ahamad, “Generating Text through Adversarial Training using Skip-Thought Vectors”, in NAACL Student Research Workshop (SRW), 2019.

[5] P. Tambwekar, M. Dhuliawala, A. Mehta, L. Martin, B. Harrison and M. Riedl, “Controllable Neural Story Generation via Reinforcement Learning”, arXiv preprint arXiv:1809.10736v1. 2018.

[6] M Mirza and S. Osindero, "Conditional Generative Adversarial Nets", arXiv preprint arXiv:1411.1784v1. 2014.

[7] Y. Kim, "Convolutional neural networks for sentence classification", in Conference on Empirical Methods in Natural Language Processing, 2014.

[8] A. Dash, J. Gamboa, S Ahmed, M Liwicki and M. Afzal, “TAC-GAN – Text Conditioned Auxiliary Classifier Generative Adversarial Network”, arXiv preprint arXiv:1703.06412v2. 2017.

[9] X. Deng, Y. Zhu and S. Newsam, “Using Conditional Generative Adversarial Networks to Generate Ground-Level Views from Overhead Imagery”, arXiv preprint arXiv:1902.06923v1. 2019.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Nets", in 28th Conference on Neural Information Processing Systems, 2014.

[11] L. Logeswaran and H. Lee, "An Efficient Framework for Learning Sentence Representations", in ICLR, 2018.

[12] R. Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, A. Torralba, R. Urtasun and S. Fidler, "Skip-Thought Vectors", in NIPS, 2015.

[13] Y. Zhang, Z. Gan and L. Carin, "Generating Text via Adversarial Training", Workshop on Adversarial Training", in NIPS, 2016.

[14] T. Mikolov, K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", in International Conference on Learning Representations, 2013.

[15] J. Pennington, R. Socher and C. Manning, "GloVe: Global Vectors for Word Representation", in International Conference on on Empirical Methods in Natural Language Processing, 2014.

[16] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Utrasun, A. Torrabla and S. Fidler, "Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books", in International Conference on Computer Vision, 2015.