SeqGAN: Sequential Generative Adversarial Network Model for Text Mining Using Facebook Bamboo Forest Data

Gang-Woo Kim, Se-Hee Lee, Sun-Mi Moon and Jae-Gul Choo*

Departure of Statistics, Korea University , Seoul 02841, Republic of Korea
Departure of Industrial Management Engineering, Korea University , Seoul 02841, Republic of Korea
Departure of Business School, Korea University , Seoul 02841, Republic of Korea
gamdengy@korea.ac.kr, lsh9382@korea.ac.kr, msm435@korea.ac.kr
*correspondence: jchoo@korea.ac.kr

Abstract

Generative Adversarial Network (GAN) is a model that has achieved considerable success in generating continuous real-value data by doing adversarial training the generator and discriminator in opposition. As a result, the application of GAN to image data has many successful cases. However, there is a limitation that the GAN is not suitable for text data. The limitations of GAN are as follows. First, GAN is suitable for continuous data generation, so it is difficult to generate a discrete token sequence. This is because gradient updates are difficult for discrete outputs of Generator. Second, GAN computes scores and loss based on the entire sequence (in one sentence). However, giving a feedback to a partially generated sequence before completing a sentence will be a better sequential generating model. SeqGAN is the model that can solve these problems. First problem of GAN, the gradient differentiation problem in the generator, is solved by the stochastic policy of reinforcement learning (RL). In addition, by using the fact that the reward is applied to the action taken in each state in the reinforcement learning, feedback can be given in the intermediate stage using the efficient method Monte Carlo Search. In addition, a complex pre-processing process was needed in that it used Korean text instead of English text in terms of terms of Bamboo Forest data of Facebook, and it is meaningful that it is 'Bamboo Forest' data reflecting interests and trends of 20's college students.

1 Introduction

In contrast to many previous generator models do research using English texts, this paper deal with Korean texts which are difficult to handle. In text mining and deep-learning research, there are large number of open sources and sophisticated analysis packages for English text. However, when it comes to Korean text, it is difficult to process it because there are fewer sophisticated tools. Furthermore, existing Korean analysis tools are the result of learning with well-documented articles such as the news. Therefore, there was a limit to the fact that it was difficult to apply to the bamboo forest post, which was grammatically flawed and consisted of colloquial words, rather than literary style.
Word2Vec was selected as embedding model. At the beginning of the study, the Word2Vec model itself was not constructed from scratch, it was expected that transfer learning on pre-trained Word2Vec models with already-learned weight values would result in better output. However, the pre-trained model could not learn further and there were few overlaps with tokens of bamboo forest data. Therefore, we built Word2Vec model by learning only tokens of the bamboo forest post from the

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

beginning. In terms of 'most similarity', the embedding of the self-trained model shows the trend of 21st century college students reflected in the bamboo forest post more than the embedding result of the pre-trained model.

We tried SeqGAN optimized for text generation, breaking away from the limits of GAN optimized for image generation. The architecture of SeqGAN was identified, the parameters were adjusted, and any significant differences were reviewed after the experiment. We also experimented on how well-preprocessed text and Word Embedding of two types of model affect output of SeqGAN. The limits of this study are as follows. This model has difficulties reflecting the context because it cannot be linked to the previous sentence. In other words, context is considered only within sentences. And because of characteristic of Korean, it is difficult to reflect the word "spacing" when generating sentences.

We also tried to refine as neatly as possible during the pre-treatment process, but we had a difficult time resolving irregular colloquialisms. This research also tried to refine as much as possible during the pre-processing process, but we had a difficult to resolve irregular colloquialisms. In terms of performance measure, it was difficult to determine performance through 'BLEU-4' scores. As a basic limitation of GAN, it is difficult to do objective assess about output text. output only can be viewed and judged subjectively.

2 Pre-Processing

Since the raw text data cannot be used in the model as it is, pre-processing process should be performed. The process of this is as follows. The first process of Pre-Processing is Crawling. The data used on this research is Korean text posted on the Facebook bamboo forest page. There are several bamboo forest pages by university and each university student posts anonymously. Since the data used in this paper was not open data, it was necessary to perform a web crawling. Tool of Crawling is selenium. Selenium is a testing framework for web applications that supports several functions for automated testing. Selenium can control the Chrome browser installed on the operating system through an API called Webdriver. In this study, Chrome browser is used. The source of data is Grouper, which provides posts originally posted on Facebook. About 80 universities are listed, and the study refers to bamboo forest posts of Korea University, Yonsei University, and Sungkyunkwan University. Crawling is automated in ascending order based on date. As a result, the number of posts is about 120,000. Second, process of Pre-Processing should be focused on characteristics of Korean Text. The language of data on this research is not English but Korean, and that it is a colloquial style, not a literary style.

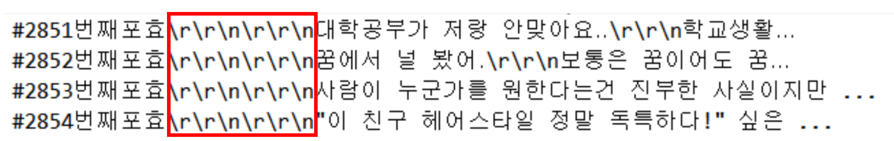

Figure 1: Text example with CRLF

Next Process is Regular Expression. The process of founding patterns and removing them is automated. Line breaks are included because posts are imported from an HTML structure. But they do not have meaning, they should be removed. In addition, the following regular expression process is included.

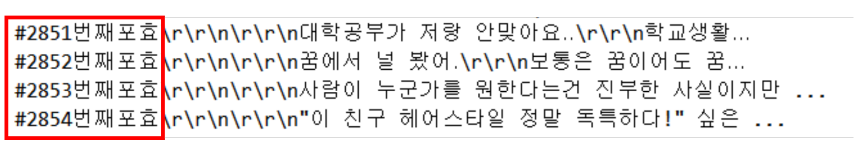

Figure 2: Example of Repeated Pattern in Bamboo Post

As shown in the figure above, Repeated things in a pattern that was removed at the same time. Converting the special characters to spaces. 'NA' and blank were also removed. Also, the first letter of the sentence starting with a special character was removed. Also, process of selecting texts of good quality is needed. The document containing tokens used less than three times was considered insignificant and was removed. After that, based on the number of the 'likes', top 30,000 posts were selected as text data to be utilized.

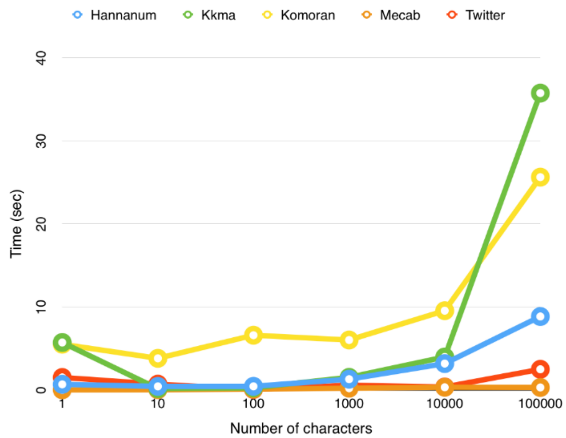

Figure 3: Execution Time of 5 POS packages of KoNLPy

```
'앞으로 핑크빛 돋는 제보들은 운영자 분들이 자체 생략 해주셨으면. 하 진짜 많은 분들이 솔로 들 마음에 불을 지르네요'
['앞', '으로', '핑크빛', '돋', '는', '제보', '들', '은', '운영자', '분', '들', '이', '자체', '생략', '하', '어', '주', '시', '었', '으면', '.', '하'
'진짜', '많', '은', '분', '들', '이', '솔로', '들', 'ㄹ', '마음', '에', '불', '을', '지르', '네요']
```

Figure 4: Exampe of Tokenization on morpheme unit

To conform to the generator model, the text data was tokenized on morpheme unit, not on a word. 'Mecab' out of KoNLP's five POS tagging packages is used as a tool of Tokenization. 'Figure 3' shows a comparison of the time of evolution of five packages. Also, Stop words were excluded from the tokenized words. Finally, SeqGAN cannot receive tokenized data directly. Therefore, each token was given a non-overlapping index. Ultimately, the text in sentences expressed by index rather than in Raw-Korean tokens is the input data of SeqGAN.

3 Word2Vec

In 2013, Tomas Mikolov introduced word embedding model <word2vec> which try to minimize computational complexity. They found that most of the complexity is caused by the non-linear hidden layer in the model. While this is what makes neural networks so attractive, they decided to explore simpler models that might not be able to represent the data as precisely as neural networks, but can possibly be trained on much more data efficiently. Because of the much lower computational complexity, it is possible to compute very accurate high dimensional word vectors from a much larger data set.

They proposed two architecture, Continuous Bag-Of-Words Model and Continuous Skip-gram Model.

The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word. CBOW architecture is similar to the feedforward NNLM, where the non-linear hidden layer is removed and the projection layer is shared for all words (not just the projection matrix); thus, all words get projected into the same position (their vectors are averaged). The weight matrix between the input and the projection layer is shared for all word positions in the same way as in the NNLM. Training complexity is then

$$Q = N * D + D * log_2(V).$$

Skip-gram architecture is similar to CBOW, but it tries to maximize classification of a word based on another word in the same sentence. More precisely, we use each current word as an input to a log-linear classifier with continuous projection layer, and predict words within a certain range before and after the current word. We found that increasing the range improves quality of the resulting

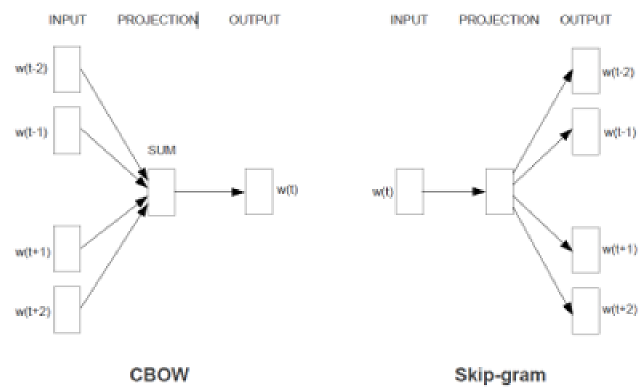

Figure 5: Left: CBOW, Right: Skip-Gram[7]]

word vectors, but it also increases the computational complexity. The training complexity of this architecture is proportional to

$$Q = C * (D + D * log_2(V)).$$

3.1 Pre-trained Model

We used 2 different word2vec models to embed our word token. First model is pretrained models from https://github.com/Kyubyong/wordvectors.

Table 1: Pre-Trained Model

# of vocabularies	30185
vector size	200
source	wikipedia

We chose this model because there are not many available Korean word2vec models but it has some limits. First, there are only 10406 words which are overlapped with our word token from bamboo facebook pages. So we need to add our new words which is not included in pretrained model and train and embed them with existing words in pretrained model. But there is unsolved bugs in genism word2vec module which make us impossible to additional training on pretrained model. Therefore, We had no choice but to embed new words with random initialization which could not be expected to perform well. Second, the source of the word in pretrained model is Wikipedia which is far from the interest of university students. We will treat this problem again when we compare pretrained model and our self-trained model.

3.2 Self-trained Model

We decided to build self-trained model to overcome the limitations of pretrained model. We use skip-gram model instead of CBOW model because it performs better. Even though skip-gram is slower than CBOW, that is not critical to our project data size and current computing power.

With this model, we can solve the limitations of pretrained models and we can improve our existing model by adding new word and training them with our existing vocabularies.

3.2.1 Variant of self-trained model

After some experiments, we add some options in preprocessing step to improve performance and build variant of self-trained model. We add '@' before the start token of the sentence and add '#' after the end token of the sentence to mark start and end of the sentence. Because when we give input to seq-Gan model as real data, we use array of index numbers replacing word tokens in the 'post' (not

Table 2: Self-Trained Model

# of vocabularies	30210
# of trained sentences	30000
# of trained word tokens	5499058
vector size	100
source	Bamboo facebook page

sentence) which include several sentences. We think if we marked the start and end of the sentence it helps our model to distinguish each sentence and by learning it generate full sentence. And we also expected that if we can use '@' as start token, it would be great.

3.3 Pre-trained model vs Self-trained model

	most_similar('학교')	most_similar('연애')	most_similar('가')
pretrained model	1.학교의 4.전문학교 2.강습소 5.사립학교 3.중고등학교 6.소학교	1.성행위 4.애정 2.로맨스 5.고뇌 3.동성애 6.풍자	1.놀드 4.수가 2.머지않 5.스널 3.를 6.오
self-trained model	1.연세대 4.대학교 2.고려대 5.대학 3.경희대 6.초등	1.짝사랑 4.CC 2.결혼 5.장거리 3.사귀 6.썸	1.& 4.'.' 2.를 5.로서 3.는 6.잘못

Figure 6: Similar words according to model type

We checked the embedding of two models by using 'most similar' function on three word, '학교', '연애', '가'(postposition). We found the generation gap between trained word between two models. Pretrained model printed '소학교', '강습소' as similar word to '학교'. These words are not used these days. Self-trained model printed '짝사랑', 'CC'(campus couple), '썸' as similar word to '연애' which are frequently used among university student these days. On the other hand, words printed by pretrained model '성행위','고뇌' are far from the interest of young students about '연애'. When we printed word similar to postposition '가', self-trained model evidently shows the effect of morpheme tokenization. Pretrained model printed meaningless, unidentifiable words as similar word to '가' because token of pretrained model is based on word token. So in pretrained model, postposition is combined with noun like '학교의' when it tokenized. On the other hand, self-trained model printed '&'(space mark in our project) and other postposition word '를', '로서','는'. As you know there is always space after postposition word. This result obviously shows the difference between pretrained model based on word token and self-trained model based on morpheme token.

4 Model Architecture

There are many deep learning model on these days. Typical deep-running models include CNN, RNN, and LSTM. There are several structures that have been modified appropriately. This section of this paper introduces a brief description of CNN, RNN and seqGAN, a deep learning model that is directly involved in this research.

4.1 CNN: Convolutional Neural Network

CNN has a structure with several convolutional layers, nonlinearity layers and pooling layers in front of the traditional Fully-Connected layer. Convolutional layers and pooling layers are used to extract image features. The final layer, the Fully-Connected layer, labels classes of images. To extract image features, CNN filters input data and performs a composite multiplication operation, resulting in a feature map. When going through the convolutional layer, shape of the output data changes according to the Filter size, Stride, Padding size, and Max Pooling size. It is also possible to classify text data as well as image data.

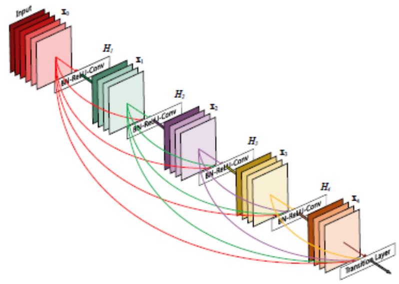

Figure 7: various architecture of CNN(1: Densenet[3])]

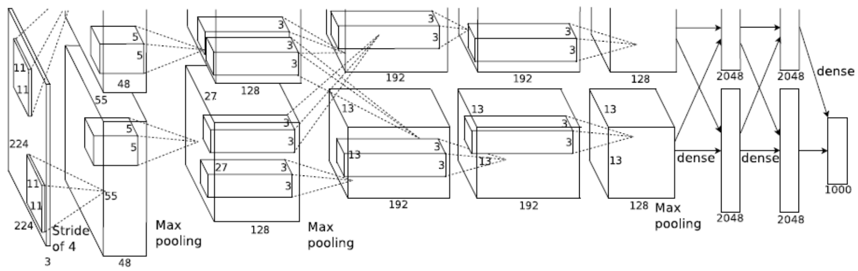

Figure 8: various architecture of CNN(2: Imagenet[1][2])

The well-known CNN Architecture includes LeNet (Yann LeCun), AlexNet(Alex Krizhevsky, Ilya Sutskever, Geoff Hinton, Geoff Hinton), ZFNet(Matthew Zeiler, Rob Fergus), GoogleNet(Szegedy et al.), VGGNet(Karen Simonyan, Andrew Zisserman), ResNet(Kaiming He et al.), DenseNet(Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q.), etc.

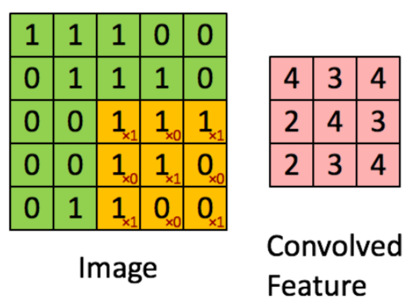

Figure 9: convolution operation

Forward propagation learning of CNN is a process of passing a filter on an image data x, performing a composite product operation, applying an activation function such as sigmoid or ReLU, and going through a pooling layer. In the fully connected layer, which is the last layer, classification is performed using softmax function and cross entropy. Conv Layer is summarized as follows. Input is a volume of W1 * H1 * D1 size and requires four parameters. K is the number of filters, F is the horizontal / vertical size of the filter, S is the stride, and P is the padding size. The output is $W2 * H2 * D2$ and $W2 = (W1 - F + 2P)/S + 1$, $H2 = (H1 - F + 2P)/S + 1$. $D2 = K$. Since there is a parameter sharing, we have total $(F * F * D1) * K$ weights and K biases. This computation of the CNN allows the neural network to utilize the spatial information of the image. The pooling layer reduces the size of the feature map. Thus, noise is reduced and speed is increased. Also, the field of view at a glance broadens and some spatial information can be taken. However, there is a disadvantage of losing information that the image has. Typical types of pooling are max pooling and average pooling.

4.2 RNN: Recurrent Neural Network

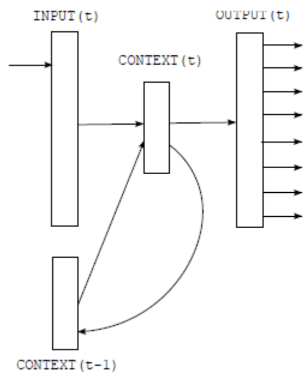

Figure 10: simple recurrent neural network

A Recurrent Neural Network is a straightforward adaptation of the standard feed-forward neural network to allow it to model sequential data. At each timestep, the RNN receives an input, updates its hidden state, and makes a prediction. The RNN's high dimensional hidden state and nonlinear evolution endow it with great expressive power, enabling the hidden state of the RNN to integrate information over many timesteps and use it to make accurate predictions. Even if the non-linearity used by each unit is quite simple, iterating it over time leads to very rich dynamics.

4.2.1 Forward Propagation of RNN

The standard RNN is formalized as follows: Given a sequence of input vectors $(\mathrm{x}1, \ldots, \mathrm{x}T)$ the RNN computes a sequence of hidden states $(h1; :::; hT)$ and a sequence of outputs $(o1; :::; oT)$ by iterating the following equations Generating Text with Recurrent Neural Networks for $t = 1$ to T :

$$h_t = tanh(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \qquad (1)$$

$$o_t = W_{ot}h_t + b_o \qquad (2)$$

In these two equations, W_{hx} is the input-to-hidden weight matrix, W_{hh} is the hidden-to-hidden (or recurrent) weight matrix, W_{oh} is the hidden-to-output weight matrix, and the vectors b_h and b_o are the biases. The undefined expression $W_{hh}h_{t-1}$ at time t = 1 is replaced with a special initial bias vector, h_{init}, and the $tanh$ is non-linearity function.

4.2.2 Backward Propagation of RNN

The gradients of RNN are easy to calculate because they are computed through backpropagation over time. So, RNN seems to be easy to do gradient update. In practice, however, the structure of the RNN makes the gradient descent inefficient. Hochreiter (1991)[5] and Bengio et al. (1994)[6] proved that the gradient exponentially decays when backpropagation proceeds with time. Thus, the RNN is not suitable for use when there are long-time dependencies. In addition to gradient vanishing problems, back-propagated gradients may be blown up exponentially. This increases the variance of the gradient, and the learning becomes unstable. The problem with this gradient descent is the biggest problem with the RNN model. Therefore, a model called LSTM, which is a structure in which a cell state is added to the hidden state of the RNN, has been devised.

4.3 SeqGAN: Sequantial Generative Adversarial Network

4.3.1 Limitations in GAN

The GAN is a model that has achieved considerable success in generating continuous real-value data by learning the generator and discriminator adversially. As a result, the application of GAN to

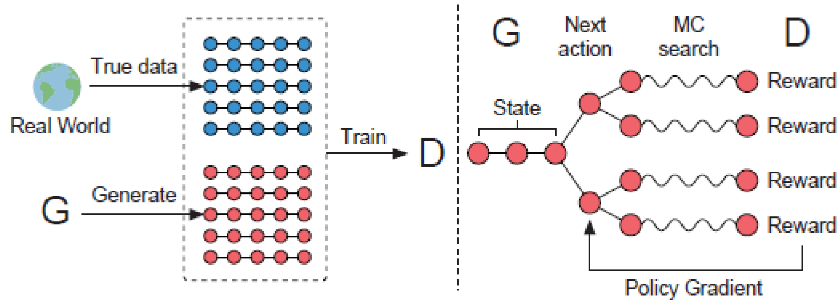

Figure 11: SeqGAN summary

image data has many successful cases. However, GAN has a limit to not being suitable for text data. SeqGAN is a model that can overcome these GAN limitations. GAN 's first problem, the gradient differentiation problem in the generator, was solved by the stochastic policy of reinforcement learning (RL). In addition, reinforcement learning can be given feedback in the intermediate stage by imposing a reward for the action taken in each state and using the efficient method Monte Carlo Search. In seqGAN in this paper, LSTM is applied to the generator and CNN is applied to the Discriminator.

4.3.2 Model Architecture

The sequence generation problem can be defined as follows. Given a sequences data of real-world, train a θ -Parameterized generator G which generates a sequence $Y_{1:T} = (y_1, ..., y_t, ..., y_T), y_t \in \mathcal{Y}$, where $\mathcal{Y}$ is the vocabulary of candidate tokens. We can see this task with aspect of reinforcement learning. In each timestep t, the state s is the previously generated tokens $(y_1, ..., y_{t-1})$ and the action a is same with producing new token y_t. Thus, the generator $G_\theta(y_t|Y_{1:t-1})$ could be interpreted as the policy model in RL.
Also, a ϕ -parameterized discriminator D_ϕ acts as a guidance for improving generator G_θ. $D_\phi(Y_{1:T})$ denotes for a probability indicating how likely a sequence come from a real-world dataset. As illustrated in Figure(11), discriminator D_ϕ is trained for classifying fake data synthesized by G from true data of real-world.
When discriminator D classifies true and fake data, D can calculate a loss for partially generated sequences by employing MC(Monte Carlo) search on the basis of the expected end reward. For making this process fast, the roll-out policy is employed at this time. Conclusively, the reward will increase as expected complete sequences have a high likelihood that it fools D.

4.3.3 SeqGAN via Policy Gradient

In SeqGAN, the objective of the generator(policy) model $G_\theta(y_t|Y_{1:t-1})$ is to generate sequence from the beginning state s_0 to maximize its expected end reward.

$$J(\theta) = E[R_T|s_0, \theta] = \textstyle\sum_{y_1 \in \mathcal{Y}} G_\theta(y_1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1)\ , \qquad (1)$$

where R_T is the reward for a complete sequence, $Q_{D_\phi}^{G_\theta}(s_0, a)$ is the action-value function of a sequence. What this equation means is that given state s, G generate next token y_t (action) and get reward from its action-value function(Q-function). So, let's specify this action-value function.

$$Q_{D_\phi}^{G_\theta}(a = y_T, s = Y_{1:T-1}) = D_\phi(Y_{1:T}) \qquad (2)$$

As we can see in the equation (2), D has a role as action-value function in SeqGAN. However, it has a limit that D can provide a reward value for only a complete sequence. To overcome this it applies Monte Carlo search with a roll-out policy G_β to sample the unknown next T - t tokens. As AlphaGo calculated all possible cases whenever a go-stone is laid on a go-board to win the game, SeqGAN calculates the reward of a last sequence likely to be constructed at the end. So, we can define a reward for intermediate state as follows.

$$Q_{D_\phi}^{G_\theta}(a = y_T, s = Y_{1:T-1}) = \tfrac{1}{N} \textstyle\sum_{n=1}^{N} D_\phi(Y_{1:T}), Y_{1:T}^{n} \in MC^{G_\beta}(Y_{1:t}; N) \text{ for t < T} \qquad (3)$$

where MC^{G_β} means N samples of expected complete sequence given $Y_{1:t}$. Conclusively, (LantaoYu, 2016) suggests the gradient function as follows by going through several steps.

$$\nabla_\theta J(\theta) \simeq \sum_{t=1}^{T} E_{y_t \sim G_\theta(y_t|Y_{1:t-1})}[\nabla_\theta log G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)] \quad (4)$$

5 Experiments

5.1 Training setting

Above table is a specific pseudo code for an algorithm for training SeqGAN. At the beginning of the training, it is trained by Maximum likelihood estimation(MLE) to pre-train G_θ with the training set and then train D_ϕ to classify the fake data from pre-trained G_θ with true sequence dataset S.
After the-pretraining step, the generator and discriminator start the adversarial training. As the generator gets progressed via training by a policy gradient of each states on g-steps updates, discriminator keeps its weights, which means doesn't learn. Since there are few improvements when one of the generator and discriminator is overwhelmed by the other, it is important to keep a balance between genrator and discriminator. Thus, it is our choice to adjust a ratio between g-steps and d-steps, each of which means the number of training iteration of D or G in one adversarial step. And to reduce the variability of the estimation, it applies different sets of negative samples with positive ones, which is similar to bootstrapping(Lantao, 2016).
Firstly, we set up 10,000 training dataset S, each of which have 30 length of sequences. We use RNN as our generative model, specifically LSTM(Long-Short-Term-Memory) to implement the performance. It is widely used model for generating a sequences of tokens and it is appropriate model to get a gradient for each state t. And we choose the text-CNN as our discriminator since CNN has recently been shown of great effectiveness in text classification(Zhang and Lecun, 2015). Additionally, to enhance the performance, we also add the highway architecture, dropout and L2 regularization to avoid over-fitting and to improve optimization.

5.1.1 Evaluation Metric

During training, we checked whether learning progresses well and how good the performance of output model is. So, we use NLL(Negative log-likelihood) and PL(Policy gradient Loss) as training error in the pre-training and the adversarial training step each. The detail about NLL is as follows

$$NLL = -E_{X_{1:T} \sim S}[\sum_{t=1}^{T} log G_\theta(x_t|X_{1:t-1})], \quad (5)$$

where S is training sequence dataset and X is a set of tokens from it.
We use BLEU(Bilingual Evaluation Understudy) score as our output senetence evaluation. BLEU score is an algorithm for evaluating the quality of text by calculating modified precision score comparing with reference sentences. We sample 100 output sentences and calculate BLEU-4 score as final metric.

5.2 Experiments Specific

We have progresses 3 experiments in training processes for finding best wy to train this model. Firstly, for checking an effectiveness of adversarial training, we compare MLE model that goes through only pre-training process and SeqGAN model that goes through both of training processes. As we can see in the Figure(12), a training error NLL goes down similarly before starting adversarial training but after that SeqGAN model goes down distinctly fast. Also as we can see in table(3), SeqGAN model has enormously higher BLEU-4 score than that of MLE model.
Secondly, for checking an effectiveness of pre-training, we compare PL of No-pretrain model that goes through the adversarial training directly with that of SeqGAN model we used in first experiments. The learning curves shown in Figure(12) illustrate a need of pre-training process. We can see that the Policy loss of No-pretrain model at first iteration over 1,000 which means the explosion of loss. On the other hands, SeqGAN model has stable PL value in entire process. As we can find in Table(3), SeqGAN has also higher BLEU-4 socre than that of No-pretrain model. However, PL values of both models are quite instable as PL value of SeqGAN increases at the last step. It would be discussed later in this chapter.

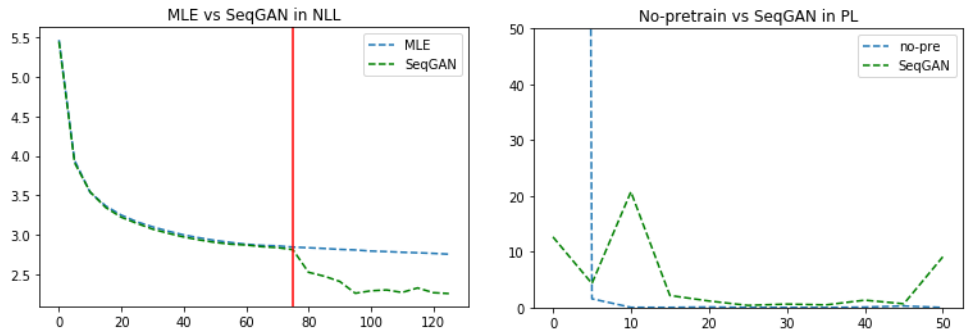

Figure 12: The training curves on experiment 1 and 2

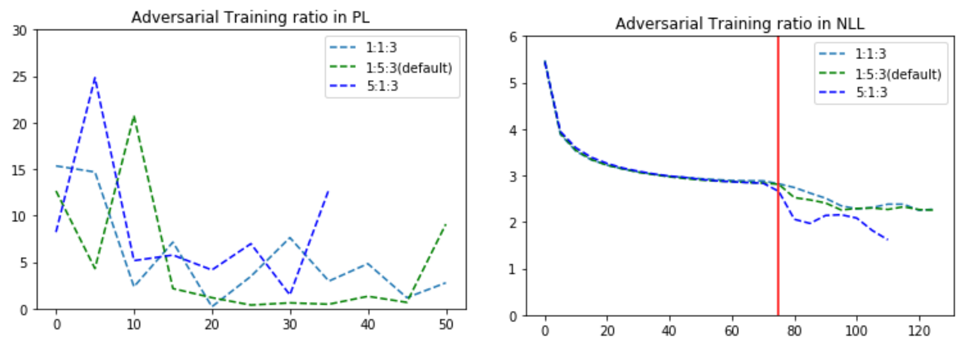

Figure 13: The training curves on experiment 3

Lastly, we change the ratio of g-steps and d-steps in one adversarial training step. SeqGAN model used in previous two experiments was 1:5:3 (g-step:d-step:k, where k is the number of repeated negative sampling for one positive sample). The learning curves are shown in Figure(13) and BLEU-4 scores are shown in Table(3). We can find slight differences between model 1:1:3 and model 1:5:3 whereas model 5:1:3 has poor performance. This aspect was same in BLEU-4 score. It also would be discussed in 'Result' section in this chapter.

Table 3: BLEU-4 score metric

Model	BLEU-4 score
MLE	0.1279
No-pretrain	0.2427
SeqGAN(1:5:3)	0.7472
SeqGAN(1:1:3)	0.7813
SeqGAN(5:1:3)	0.5230

6 Result

6.1 Training

In summary as we can see in the previous section, we can get such conclusions. Firstly, model SeqGAN in which pre-training process and adversarial training process are properly combined was more superior than the model in which one of the two training process is skipped. It can be verified in result of training loss and BLEU-4 score. Secondly, it turned out that the ratio between g-steps and d-steps have a lot of influence on the performance and training of the model. According to our results,

1:1 is best choice to train.
On the other hands, some limitations and issues are identified during experiments. While the monotonous decreasing patterns are observed in NLL values, there were striking fluctuations during adversarial training in PL, which actually constructs the reward for generator G. Although we thought it can be solved or at least specified by much more adversarial training process, we couldn't try this solution due to deadly huge computing cost of adversarial training. However, it was sufficient meaningful to find out an effectiveness of adversarial training with relatively small iterations of iterations.

6.2 Output sentences

GoodCase	BadCase
그렇게 어느 날은 먼저 잘하고 싶은 친구도 아니었어요.	대숲!
연애하고 싶다.	우리는 부당하게 하게 수백우유해서 유전자에서 살고싶은 헌신 같다.
작년 이맘때 까지 감사하고 그렇게 하고 싶었어요.	생각합니다.
모두 좋은 공간이 너무 힘들어.	진짜로 갈수록 감사하고 부끄럽어.
그런데 나같고 싶은 후배들은 좋아하지!	내가 너를 수술하고 싶었어.

Figure 14: Good / Bad cases of Output sentences

Figure(14) is some examples from output of our generative models. Above 10,000 sentences, we can choose some qualitatively good/bad case of generated sentences. In good case of examples, we can find these sentences have almost no grammatical error so that we can guess the model have learned about the grammatic system of Korean. Above all, it is astonishing for the model to learned the word of specific community. The interests of university student are also well reflected on these sentences. However, it also has limitations found in bad cases. Firstly, it seems that it is hard for the model to construct a long sentences. A lot of sentences have a few tokens. And vocabularies thought to be rarely used even in this texts appear in sentences, which make all of sentences weird.

7 Conclusion

The conclusion of study is as follows. We did a lot of research and try and error to train SeqGAN model for Korean text properly, preprocessing tremendously and training our own Word2Vec model so on. Also, we conducted experiments for finding optimal modified model of SeqGAN. Although we found our optimal model, there can be some way to be desired. By improving the model structure and optimizing hyperparameter search, stabilizie policy loss of model and increases its completeness of generated texts.

8 Future Works

At the end of the study, there are many things to be desired. First, for making the model able to construct longer sentences, we can set padding-mask for loss function of generator G. Since discriminator D only can get fixed size of sequences as input, we should make a padding in blank sequences, which is suspicious for making the model learn to generate padding more frequently. Also, we can try to apply various model architecture in generator G. Attention-based model like transformer or Seq2Seq models could be an superior alternatives.

References

[1] Sahiner, B., Chan, H. P., Petrick, N., Wei, D., Helvie, M. A., Adler, D. D., & Goodsitt, M. M. (1996). Classification of mass and normal breast tissue: a convolution neural network classifier with spatial domain and texture images. IEEE transactions on Medical Imaging, 15(5), 598-610.

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[3] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017, July). Densely connected convolutional networks. In CVPR (Vol. 1, No. 2, p. 3).

[4] Sutskever, I., Martens, J., & Hinton, G. E. (2018). Generating text with recurrent neural networks.

[5] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma, Technische Universität München, 91(1).

[6] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157–166, 1994

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient es-timation of word representations in vector space. CoRR, abs/1301.3781,2013.

[8] Yu, L., Zhang, W., Wang, J., & Yu, Y. (2017, March). SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In AAAI (pp. 2852-2858).

[Zhang and LeCun 2015] Zhang, X., and LeCun, Y. 2015. Text understanding from scratch. arXiv:1502.01710.