Sequence Based Adversarial Text Summarization

Yuze Yang
17021414
zcahyy3@ucl.ac.uk

Ke Xu
20081693
ucabkxu@ucl.ac.uk

Qinghong Zhou
20060121
ucabqz9@ucl.ac.uk

Xinyu Shen
20063030
ucabxs0@ucl.ac.uk

Abstract

Recent sequence-to-sequence (seq-to-seq) based models lead to successful abstractive text summarization. Expanding from seq-to-seq, pointer-generator with coverage and attention handles out-of-vocabulary in datasets and avoids repetition of words in summaries. Their results through beam seach are captured as generator's input in the generative adversarial network (GAN) developed from SeqGAN. Reinforcement learning methods are used to evaluate the output from the generator. Then a convolutional neural network (CNN) with highway layers functions in the discriminator to update the summerization model.

1 Introduction

Text summarization, a popular task in the natural language processing (NLP), aims to generate a concise representation of the key information in the source text. Extractive and abstractive are two main approaches in this task. The extractive method extracts sentences from the source text directly, while the abstractive summarization is capable of outputting words that are not in the source text if the model beliefs that the new words better summarize the text (Yang et al., 2020). Though abstractive summarization breaks the inherent limitation in the extractive summarization, building an abstractive model with good performance is still a challenging task (Rush et al., 2015).

The sequence-to-sequence (seq-to-seq) model (Sutskever et al., 2014) applies long short-term memory (LSTM) in both encoder and decoder. It was developed to solve the limitation of only allowing fixed sequence length in the Deep Neural Network (DNN) models and gives impressive progress in tackling machine translation (Bahdanau et al., 2016). The seq-to-seq model was then adapted to abstractive summarization with promising performance. However, the seq-to-seq model would to some extent impede the summaries to be more concise and accurate since different decoder inputs are used in training and testing. In the training process, we feed target headlines into the decoder. Yet, in the testing process, the embedding of the last generated word from the model will be used in the decoder to produce the next word. Besides, most of the seq-to-seq based abstractive models would use maximum likelihood estimation to summarize in their decoders. Hence, the overall performance may be poisoned by some unexpected intermediate token generations, i.e. the exposure bias (Ranzato et al., 2015) will make each token's prediction less reliable in the process of time (Yang et al., 2020).

In order to alleviate the influence caused by the above drawbacks in seq-to-seq based models, we consider generating words as a sequential decision making problem. We can thus formulate the task in the reinforcement learning (RL) field to predict the long-term performance using the current information, meaning the final score of each sequence prediction would be evaluated at each intermediate step. Specifically in our work, RL is used along with proposals to combine seq-to-seq based models with the Generative Adversarial Network (GAN) (Goodfellow et al., 2014). The generator part of the GAN are chosen to use a basic seq-to-seq model or a pointer-generator model with coverage (See et al., 2017), and the discriminator part will involve a convolutional neural network with highway layers (Srivastava et al., 2015). Their methodologies, experiment set-up and results will be introduced in this report. Discussions on improvement of the model will follow afterwards.

2 Related Work

Abstractive summarization

Neural abstractive text summarization model was first utilized by Rush et al. (2015) on sentence-level summarization with the attention mechanism. To solve out-of-vocabulary problems in seq-to-seq model, Nallapati et al. (2016) applied switching Generator-Pointer in the decoder such that the model can point the location of OOVs in the source text. See et al. (2017) later combined attention seq-to-seq model with pointer network (Vinyals et al., 2017) to deal with OOVs, coverage model (Tu et al., 2016) is also used to avoid repetitions in results from the seq-to-seq model.

Generative adversarial network

The generative adversarial network (GAN) (Goodfellow et al., 2014), which transforms the training procedure of the generative model to a mini-max game by introducing another discriminator model as its opponent, has achieved great success in image generation tasks (Denton et al., 2015). However, the vanilla GAN has shown less help in generating discrete data. Kusner and Hernández-Lobato (2016) and Li et al. (2017), etc. developed on how GAN could be more useful for NLP problems. Additionally, Pfau and Vinyals (2017) proposed that the GAN and the actor-critic methods in RL have many features in common. This inspired us to connect GAN and RL. Then, based on similar ideas, SeqGAN (Yu et al., 2017) was proposed for text generation. SeqGAN models the text generation as a sequential decision-making process, such that the generator can be optimized under the RL framework via policy gradient methods.

In our work, we follow the idea of SeqGAN and adapt it to text summarization field. Also, since the original SeqGAN uses the bidirectional long short term memory network as the generator, we will further improve the model's performance by upgrading the generator to a pointer-generator network (PGN).

3 Method

In this section, we will first introduce two models, **sequence-to-sequence** (Sutskever et al., 2014) and **pointer-generator** with coverage (See et al., 2017) as the generator part of the **generative adversarial networks** (GAN). GAN and its further details on how the discriminator part operates would be discussed afterwards.

3.1 Sequence-to-Sequence

The sequence-to-sequence model (seq-to-seq) by itself can already be used to solve the summarization problem (Cho et al., 2014). It generates summaries using the vocabulary set pre-trained by the training set but can not handle **out-of-vocabulary** (OOVs) in the test set (Sumit Chopra and Rush, 2016).

In the encoder part, a single layer unidirectional long-short term memory (LSTM) model (Hochreiter and Schmidhuber, 1997) is used to avoid vanishing gradient issues. Embeddings on texts and headlines convert words $x_1, x_2, ..., x_T$ into vectors $\mathbf{x}'_1, \mathbf{x}'_2, ..., \mathbf{x}'_T$ to be used in the network. Expanding from a recurrent neural network (RNN), the LSTM model obtains the hidden state at step t from current input x_t and previous summary c_{t-1} as $h_t = f(h_{t-1}, x_t, c_{t-1})$. After running over all elements in a sentence, a summary $\mathbf{c}$ of the sentence would also be obtained from the current hidden state (Cho et al., 2014).

In the decoder part, another LSTM model is used to generate the probability distribution of predicting the next word (y_t) in the summaries (Cho et al., 2014). The hidden states in the decoder are updated using $h_t = f(h_{t-1}, y_{t-1}, \mathbf{c})$, so the distribution of y_t could be written as $P(y_t|y_{t-1}, y_{t-2}, ..., y_1, \mathbf{c}) = \textbf{softmax}(f(h_t, y_{t-1}, \mathbf{c}))$. Hence, the prediction for the next possible word in the summary could be taken as the one with the largest probability in the vocabulary set. After updating model parameters with optimization algorithms, chosen words are put together as the final summary of the input data.

3.2 Pointer-generator with coverage

Pointer-generator network (PGN) is experimented as the generator of the GAN model. Pointer-generator forms a combination of sequence-to-sequence with the attention mechanism (Vaswani et al., 2017) and a pointer network (Vinyals et al., 2017). In order to deal with the repetition issue of the seq-to-seq model, **coverage** vector (Tu et al., 2016) is adapted into the pointer-generator (See et al., 2017).

Encoder

In the encoder, a single-layer bidirectional LSTM is used to eliminate the delay of evaluating future information when minimizing the loss (Schuster and Paliwal, 1997). The embedded tokens of source text with length Z after padding are sequentially fed into the forward direction of the model, and the hidden states denoted as $\mathbf{h}_1$, ..., $\mathbf{h}_Z$ would be generated. Then the information that has already passed through the LSTM would be used in the reversed order in the backward state to update the weights for summarization (Schuster and Paliwal, 1997) .

Decoder

The decoder in our PGN model runs over a single-layer unidirectional LSTM, which iteratively produces decoder states $\mathbf{s}_1$,..., $\mathbf{s}_T$ for each step t (See et al., 2017). The summary of texts used in the encoder is embedded as $\mathbf{g}_t$ as the input of the decoder. The LSTM then operates similarly to the seq-to-seq model mentioned before.

Then we apply the attention mechanism (Bahdanau et al., 2016) in the decoder to obtain an attention distribution $\boldsymbol{\alpha}^t$ to determine the proportion of attention paid to each word. A coverage vector $\mathbf{cov}_t$ (Tu et al., 2016) is used at this step to reduce repeating words in the predicted summary. Hence the attention distribution is calculated by

$$\mathbf{u}_i^t = v^T \mathbf{tanh}(\mathrm{W}_1 \mathbf{h}_i + \mathrm{W}_2 \mathbf{s}_t + \mathrm{W}_3 \mathbf{cov}_t + \mathbf{b}_{att}) \quad (1)$$

$$\boldsymbol{\alpha}^t = \mathbf{Softmax}(\mathbf{u}^t) \quad (2)$$

where v^T, W_1, W_2 and W_3 are weighting parameters to be learned from the model (Tu et al., 2016).

Since the attention distribution only contains variables from the decoder state, we would need hidden layers of the encoder to represent all information passed at step t (See et al., 2017). This is defined as the context vector $\mathbf{c}_t$ using

$$\mathbf{c}^t = \sum_{i=1}^{Z} \boldsymbol{\alpha}_i^t \mathbf{h}_i \quad (3)$$

The probability distribution of predicting words in time step t is a weighted sum of vocabulary distribution and attention distribution. The weights p_{gen} and $1 - p_{gen}$ of two distributions allow us to either generate words from the vocabulary set or copy words from the source text. This solves the issue of handling out-of-vocabulary(OOV) words in abstractive summarization models (See et al., 2017). Denoting z_j as a candidate word, we can compute its probability as

$$P^t(z_j) = p_{gen}^t P_{vocab}^t(z_j) + (1 - p_{gen}^t) \sum \boldsymbol{\alpha}_j^t \quad (4)$$

where the weight of generating data is

$$p_{gen}^t = \boldsymbol{\sigma}(\mathrm{M}_1^T \mathbf{c}_t + \mathrm{M}_2^T \mathbf{s}_t + \mathrm{M}_3^T \mathbf{g}_t + b_{gen}) \quad (5)$$

and the weight of extracting words from the vocabulary set is

$$P_{vocab}^t(z_j) = \mathbf{Softmax}(\mathrm{V}_1(\mathrm{V}_2[\mathbf{s}_t, \mathbf{c}_t] + b_1) + b_2) \quad (6)$$

Other weights M_1, M_2, M_3, V_1, V_2, b_1, b_2 and b_{gen} are parameters to be learned from the model.

When pre-training the PGN generator, the loss of the model is a combination of negative log-likelihood for the target word z^t and a penalty for repetition from the coverage mechanism. The equation of loss at time step t can be written as:

$$l_t = -logP^t(z^t) + \lambda \sum_i \min(\boldsymbol{\alpha}_i^t, \mathbf{cov}_i^t) \quad (7)$$

Beam search

Since the output of the PGN model is the probability of words in the vocabulary set to be chosen in the summary. We would need to perform **beam search**, a greedy method, to select the most likely words that follow sequentially in the summary (Steinbiss et al., 1994). For a beam size k, we would select k candidates with the highest probability to be used given the previous chosen words at each step. The search stops either when the max step is reached or when we could not find things other than stop words where the summary ends. Indices generated from beam search will then be used in the discriminator as a part of the GAN model.

3.3 Generative Adversarial Networks (GAN)

Based on the original structure of Generative Adversarial Network (GAN) (Goodfellow et al., 2014), we propose a similar model for this text summarization task with two parts. A generator G takes texts as the input and produces their corresponding headline (summary), and then a discriminator D is used to distinguish whether the input instance is replicated by G. Therefore, during the adversarial training process, the generative model G would learn to generate high-quality summaries compared to the true data from the real world to confuse D. Meanwhile, the discriminative model D aims to classify sequences that are generated by G with high accuracy. The structure of how the pointer-generator would be used as G in the GAN is shown in Figure 1 below.

However, generating sequence data via GAN has two drawbacks, the first one is from the traditional maximum-likelihood-estimation method and recurrent neural network (RNN) structure. For example, when minimizing the cross-entropy loss with a RNN-based generator, the loss can only be given after all the summarization sequence has finished. This means that we cannot adjust the strategy accordingly after each token has been generated.

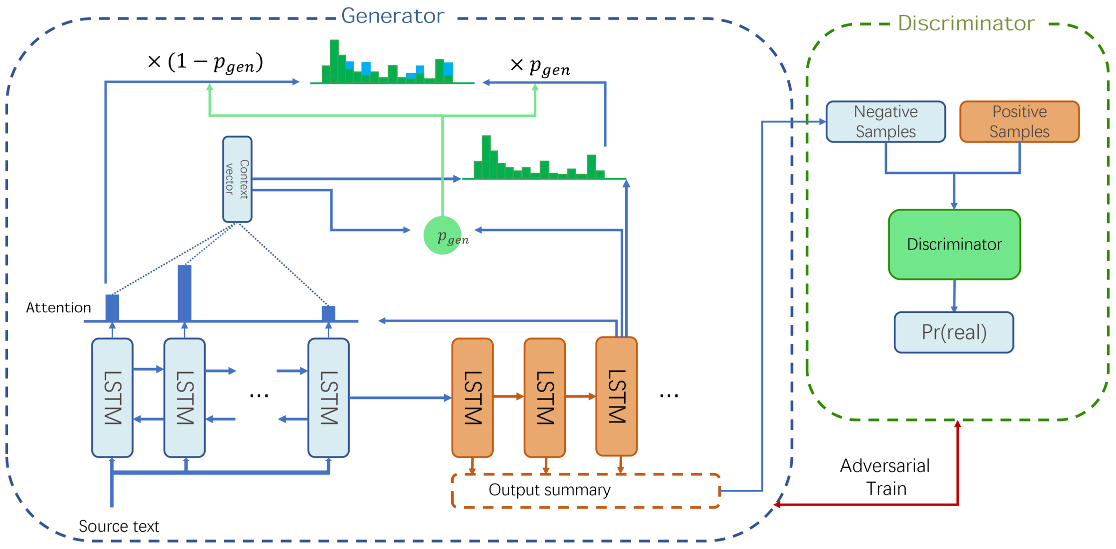

Figure 1: Visualization of structure of PGN-GAN model, adapted from (Yang et al., 2020)

Besides, this will cause the loss to increase accumulatively and become intractable as the length of the sequence increases. The second one is that GAN is originally applied to generate the real-valued continuous data, such as image generation. Hence it may struggle to produce discrete token sequences and learn from discrete sentences (Huszár, 2015).

In order to render GAN more applicable to solve the NLP tasks, we incorporate GAN with the idea of reinforcement learning (RL) inspired by the SeqGAN model (Yu et al., 2017). We first consider generating the text summarization as a sequential decision making problem. Then the agent becomes the generative net G with the state being the headline generated so far, and the action is the choice of the next token. In addition, we use the result from discriminator, i.e. the probability of a summary being a real data, as the corresponding reward function. Since this is the episodic reward, we can apply the Monte-Carlo search to estimate the action-value function. Hence by regarding generator G as the target policy, we can train G directly via the policy gradient (Sutton et al., 1999).

Generator

During the adversarial training, we use the **REINFORCE** algorithm (Williams, 1992) to train the policy of agent, i.e. the generator G. The objective function should be the expected total reward:

$$\max_{\theta} J_{pg}(\theta) = \frac{1}{T} \sum_{t_1}^{T} \sum_{y_t} q_{D_\phi}^{G_\theta}(y_t, s_{t-1}) G_\theta(y_t | s_{t-1}) \tag{8}$$

where θ is the parameter of the generator, and ϕ is the parameter of the discriminator. T is the length of the summary, y_t is the next token's prediction and $s_t = \hat{Y}_{0:t} = (\hat{y}_0, \hat{y}_1, ..., \hat{y}_t)$. In order to get the reward from the finished sequence to estimate the action-value function $q_{D_\phi}^{G_\theta}(y_t, s_{t-1})$ for each token choice at each time step $t < T$, we use the Monte-Carlo search to sample the remaining $T - t$ tokens N times according to a roll-out policy, with average taken afterwards. Noting that the roll-out policy is the same as the generator policy G_θ, although it can be considered as the 'constant' version of G without subscript θ and does not update during the optimization procedure.

Specifically, for every time step t, the Monte-Carlo search mentioned above samples N complete sequences $\{\hat{Y}_{1:T}^1, \hat{Y}_{1:T}^2, ..., \hat{Y}_{1:T}^N\}$, where $\hat{Y}_{1:t}^n = (\hat{y}_1, ..., \hat{y}_t)$ which are the current produced tokens, and $\hat{y}_{(t+1):T}^n$ is sampled by the roll-out policy. Then, the action-value function $q_{D_\phi}^{G_\theta}(y_t, s_{t-1})$ is estimated as below according to Yu et al. (2017),

$$q_{D_\phi}^{G_\theta}(y_t, s_{t-1}) = \frac{1}{N} \sum_{n=1}^{N} D_\phi(\hat{Y}_{1:T}^n) \ \forall t < T \tag{9}$$

For $t = T$, the action-value is equal to the final reward directly.

By applying the Monte-Carlo search, the performance of G at each intermediate time step will be considered. Hence the accumulative loss issue mentioned before could be solved accordingly.

Furthermore, the policy gradient theorem (Sutton et al., 1999) enables us to use the likelihood ratio trick to simplify the calculation of unbiased estimate of the **REINFORCE** objective function's gradient as below,

$$\nabla_\theta J_{pg}(\theta) = \frac{1}{T}\sum_{t_1}^{T}\sum_{y_t} q_{D_\phi}^{G_\theta}(y_t, s_{t-1})\nabla_\theta G_\theta(y_t|s_{t-1})$$
$$= \frac{1}{T}\sum_{t_1}^{T} E_{y_t}\left[q_{D_\phi}^{G_\theta}(y_t, s_{t-1})\nabla_\theta \log G_\theta(y_t|s_{t-1})\right] \quad (10)$$

where the expectation $\mathbf{E}[\cdot]$ is approximated by the sampling. Therefore, the generator can be trained, and its parameters θ could be updated via optimizing the objective function $J(\theta)$ with general gradient-based methods, such as stochastic gradient descent or Adam. In addition, we will try to use a more advanced setting of loss function as

$$J(\theta) = \lambda\, J_{pg}(\theta) + (1-\lambda)\, J_G(\theta) \quad (11)$$

where J_G is the general loss for the pre-training part of the generator. In particular, the negative log-likelihood loss, and the hyper-parameter λ would be used to trade-off between the policy gradient loss from adversarial training and the maximum likelihood loss. In the end, since the data set size is large enough to ensure convergence, during the optimization procedure, a relatively low learning rate and the gradient clipping would be used to avoid gradient vanishing.

Discriminator

Since the convolutional neural network (CNN) has shown great performance in the text classification tasks (Kim, 2014), we choose to build our discriminator basing on the CNN.

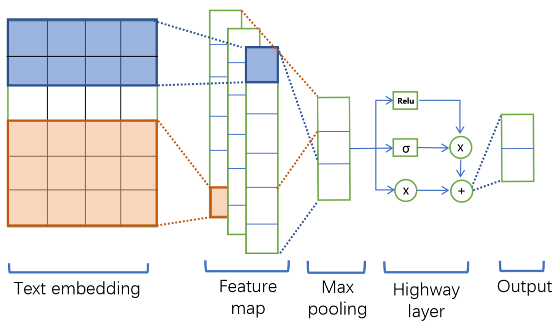

Figure 2: Visualization of an example CNN for text classification in the discriminator.

Input tokenized sequences would first be indexed using the vocabulary set as $(x_1, x_2, ..., x_T)$. Then they are passed to the embedding layer with the embedding dimension k, and subsequently they are concatenated vertically to form the embedded input matrix $\mathbf{X}_{1:T} \in \Re^{T\times k}$. Then, multiple kernels will be used in parallel to extract different features from the text sequences. For applying a specific kernel $\mathbf{w} \in \Re^{m\times k}$ with size m, we have the resulting feature c_i as

$$c_i = \mathbf{w}\cdot \mathbf{X}_{i:i+m-1} + b \quad (12)$$

where b is the bias term. Afterwards, we apply a non-linear function **F** over all features and concatenate them together to form the full feature map generated by this particular kernel

$$\mathbf{F}(\mathbf{c}) = [F(c_1), F(c_2), ..., F(c_{T-m+1})] \quad (13)$$

Then additional max-pooling layers are used, such that $\hat{c} = \max(\mathbf{c})$ (Kalchbrenner et al., 2014). Finally, the pooled features produced by different kernels would be passed through the fully connected layer to get the probability of whether the input sequences are from the real world or from the generative model G.

Additionally, to further improve the accuracy of the discriminator, another highway layer (Srivastava et al., 2015) is applied before the last fully-connected layer. Formally, suppose the original layer without highway architecture is $F(H(\cdot))$, where H is a linear layer and F is a non-linear activation function, such as ReLU function. Then, in order to upgrade this layer, we define the transform gate as $T(x) = \textbf{Sigmoid}(H(x))$, and the feature is transformed by the highway layer as

$$\begin{aligned} y =& F(H(\text{features})) \times T(\text{features}) \\ &+ \text{features} \times (1 - T(\text{features})) \end{aligned} \quad (14)$$

The above structure of the discriminator is visualized in Figure 2.

The training process of the discriminator is fairly simple compared to the generator. We will train the discriminator with respect to the following objective function

$$\begin{aligned} \min_\phi & - E_{Y\sim p_{\text{real}}}[\log D_\phi(Y)] \\ & - E_{Y\sim G_\theta}[\log(1 - D_\phi(Y))] \end{aligned} \quad (15)$$

where ϕ is the parameter of the discriminator. We can label the real sequences as $+1$ and those from the generator as -1. Then under the supervised learning framework, we can train the discriminator by minimizing the cross-entropy with padded indices ignored as,

$$L(y, \hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \quad (16)$$

Here, y is the true label of the input sequence and $\hat{y}$ is the predicted probability of the input sequence being real by D.

4 Experiment

Data Preparation

The WikiHow dataset (Koupaee and Wang, 2018) differs from conventional summarization datasets which mostly collect news articles with journalistic style. Articles in the WikiHow dataset contains a wide range of articles describing procedures of tackling issues in various areas. The WikiHow dataset provides the full texts of articles, their overall summary and their separated version split according to paragraphs. Data recording paragraphs and paragraphs' summaries is used in this experiment as the overall summary is equivalent to combining all paragraphs' summaries. This dataset thus contains 1,585,695 paragraphs and their corresponding summaries.

Special characters and symbols were cleaned before tokenizing the dataset. We then looked for the 10% quantile and the 95% quantile of paragraphs' lengths to remove data that are too long or too short to summarize. Hence, paragraphs with lengths less than 3 and those longer than 161 were removed. Simultaneously, data with headline length longer than 0.75 times of paragraph length were removed from the dataset (Koupaee and Wang, 2018). Additionally, we filtered out data with headline length shorter than 5 which is the 10% quantile length of headlines. 1,107,760 data are left after this step. Among the remaining data, we used 80% of the data as our training set, with the rest 20% equally split between validation and testing sets.

We then attempted to clean the data with stemming and stopwords. However, considering that stemming will eliminate letters in particular 'e' and 's' indiscriminately in words. Besides, stopwords contains lots of prepositions and adverbs which are important to join words together after predictions. Hence, these processes of data cleaning were not used in this experiment.

The vocabulary set of the dataset was established for both texts and headlines. In the seq-to-seq model, out-of-vocabulary (OOV) is not used, hence unrecognized words would be treated as unknown with a specific index given. While in the pointer-generator model, since the model is capable of handling OOV, we extended the vocabulary set for words not included in the training data.

Datasets were then sorted in the descending order of length with padding applied afterwards to expand all sentences to the same length for training. Since batching processing will be used in training, we considered padding the dataset using either the longest length of the full dataset or these in each batch, also known as bucketing. Quick sample run showed that the validation loss would fall faster when using the max-length for each batch.

We also experimented with two embedding methods in our models. Pre-trained GloVe (Pennington et al., 2014) and embedding with random vectors were compared on small sample test runs. The GloVe method out-performed greatly on the seq-to-seq model but did not significantly differ for the pointer-generator model.

Sequence-to-sequence

In this model, we studied the influence made by parameters by varying the hidden states and embedding vectors in the model. The shape of hidden states is emphasized and trialed on hidden sizes of 128 and 256. The results will be shown in next section.

Pointer-generator with coverage

The pointer-generator model outputs the vocabulary probability which requires beam search to predict the summary. Yet, beam search computes one sequence at a time which makes the prediction process very slow over large batches of dataset. Hence we attempted to search on the whole batch simultaneously, but the results are not too desirable. Further trials were given on creating a small beam batch for one sequence, each containing only the repetition of the text. Since our model is trained on dataset without repeated texts, these small batch did not seem to out-perform the original method.

Adversarial training

The generator and discriminator were both trained via ADAM optimizer, with batch size equals to 64 and the learning rate $1 \cdot 10^{-3}$. We set either seq-to-seq or pointer-generator networks to be the generator. The discriminator is a CNN which consists of 5 convolution layer over the sentences with an extra highway layer, this means that the parameters of our discriminator becomes large and we would need to feed in more data to avoid overfitting. Besides, the parameters were initialized by uniform or normal distributions with mean 0 and very small variance to enhance the training efficiency at early stage. Hence, based on the original settings of GAN (Goodfellow et al., 2014), we updated two parts of model iteratively, i.e. after the pre-training of generator and discriminator via maximum-likelihood methods, at each adversarial training epoch, we updated the discriminator $k = 5$ times while updating the generator once.

5 Results and Discussions

The performance of each model measured by ROUGE metrics are shown in Table 1.

Model	Rouge-1	Rouge-2	Rouge-L	time
Model 1	11.328	2.242	11.058	49
Model 2	13.273	3.032	13.033	70
Model 3	13.376	3.065	13.123	78
Model 4	8.953	0	8.905	/
Model 5	7.037	0	6.845	/

Table 1: ROUGE metrics for different models

Time represents the average time for each epoch. Model 1 is a seq-to-seq model with random embedding (128) on 100,000 data. Model 2 is a seq-to-seq model with GloVe embedding (128) on 100,000 data. Model 3 is a seq-to-seq with GloVe embedding (256) on 100,000 data. Model 4 is a seq-to-seq with GAN model with GloVe embedding (128) on 1,000 data while Model 5 is seq-to-seq model with GloVe embedding (128) on 1,000 data.

From the result, it shows that *nn.Embedding* will cost more time but with less accuracy which is not be desirable. Then the comparison between Model 2 and Model 3 shows that increasing the size of hidden states will increase the accuracy however it will be more time-consuming. As a result, a trade-off between accuracy and efficiency needs to be further studied. Finally, for the last two models, model of seq-to-seq with GAN will have better result but how much it out-performs the seq-to-seq model would require further studies on a larger dataset.

As above, the results of our baseline and other models are smaller than the result in the contemporary paper (Koupaee and Wang, 2018) which may be caused by the following aspects:

1. With the limit of time and efficiency of our computational power, we could only run our model on a much smaller subset compared to the full dataset (about one million data).

2. About a half of the vocabulary are transformed from other words including different tenses for verbs and plural form for nouns, which may influence the distribution of our outputs.

3. Generating predictions using the beam search is very slow, and could not be processed bathwise. This extended the time scale of running over one epoch enormously.

4. Training issues in GAN such as mode collapse (Thanh-Tung and Tran, 2020) leads to unsatisfactory results.

5. The quality of GAN training procedure depends on choosing a lot of hyper-parameters due to its adversarial architecture. Hyper-parameters are referred from other similar empirical experiments which may need to be further tuned to fit our model.

Potential methods tackling the above issues would be researched further. Additional pre-trained embedding methods, such as BERT (Devlin et al., 2018) embeddings may enable a better representation of word vectors in the dataset. Faster methods to generate outputs from beam search such as Best-First beam search (Meister et al., 2020) could also be considered to improve the efficiency of connecting the pointer-generator with the discriminator. Some modified adversarial training architectures, such as Wasserstein Generative Adversarial Networks (WGAN) (Arjovsky et al., 2017) and Deep Regret Analytic Generative Adversarial Networks (DRAGAN) (Kodali et al., 2017) could also be applied in the future work to optimize the model. With more efficient training procedures over a larger portion of WikiHow dataset, we could expect an improvement on the current results with our proposed model.

6 Conclusion

In this paper, we proposed an adversarial method for abstractive text summarization. Combinations of sequence-to-sequence and pointer-generator with an reinforcement learning oriented GAN methods are applied on the WikiHow dataset. Though limitations from practical aspects restrict an obvious improvement, methods and further discussions are carried forward to plan for a more efficient training and optimization in the future.

References

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein gan.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural machine translation by jointly learning to align and translate.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. 2015. Deep generative image models using a laplacian pyramid of adversarial networks.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Ferenc Huszár. 2015. How (not) to train your generative model: Scheduled sampling, likelihood, adversary?

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A convolutional neural network for modelling sentences.

Yoon Kim. 2014. Convolutional neural networks for sentence classification.

Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. 2017. On convergence and stability of gans.

Mahnaz Koupaee and William Yang Wang. 2018. Wikihow: A large scale text summarization dataset.

Matt J. Kusner and José Miguel Hernández-Lobato. 2016. Gans for sequences of discrete elements with the gumbel-softmax distribution.

Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. 2017. Adversarial learning for neural dialogue generation.

Clara Meister, Tim Vieira, and Ryan Cotterell. 2020. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

David Pfau and Oriol Vinyals. 2017. Connecting generative adversarial networks and actor-critic methods.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal. Association for Computational Linguistics.

Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks.

Volker Steinbiss, Bach-Hiep Tran, and Hermann Ney. 1994. Improvements in beam search. In *Third International Conference on Spoken Language Processing*.

Michael Auli Sumit Chopra and Alexander M. Rush. 2016. Abstractive sentence summa- rization with attentive recurrent neural networks. Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies:93–98.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. 1999. Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer.

Hoang Thanh-Tung and Truyen Tran. 2020. On catastrophic forgetting and mode collapse in generative adversarial networks.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2017. Pointer networks.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Min Yang, Xintong Wang, Yao Lu, Jianming Lv, Ying Shen, and Chengming Li. 2020. Plausibility-promoting generative adversarial network for abstractive text summarization with multi-task constraint. *Information Sciences*, 521:46–61.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient.