Reproduction Report on "MaskGAN: Better Text Generation via Filling in the___ "

Dmitry Popov
Departament of Mathematics
Higher School of Economics University
dgpopov@edu.hse.ru

Sinitsin Anton
Departament of Mathematics
Higher School of Economics University
agsinitsin@edu.hse.ru

Abstract

This work is a project report for Deep Learning course offered by the CS department of Higher School of Economics University. We have tried to reproduce the results of "MaskGAN: Better Text Generation via Filling in the___ " submitted to ICLR 2018 conference. Authors of this paper propose the new method for generating better quality text in comparison with the quality of texts obtained by autoregressive and seq2seq models.

1 Description of the paper

Typical models for text generation like autoregressive or seq2seq are usually trained via maximum likelihood method and although this method optimizes the perplexity, the quality of generated texts are rather poor. This is due to during sample generation, the model is often forced to condition on sequences that were never conditioned on at training time. This leads to unpredictable dynamics in the hidden state of the RNN. Methods such as Professor Forcing and Scheduled Sampling have been proposed to solve this issue but they do not directly specify a cost function on the output of the RNN that encourages high sample quality. However, a text generation model trained on in-filling presented by authors of this paper does so.

1.1 Notation

Let (x_t, y_t) denote pairs of input and target tokens. Let $< m >$ denote a masked token (where the original token is replaced with a hidden token) and let $\hat{x_t}$ denote the filled-in token. Finally, $\tilde{x_t}$ is a filled-in token passed to the discriminator which may be either real or fake.

1.2 Architecture

1.2.1 Masking

The task of imputing missing tokens requires that MaskGAN architecture condition on information from both the past and the future. Authors choose to use a seq2seq architecture. For a discrete sequence $x = (x_1, ..., x_T)$, a binary mask is generated (deterministically or stochastically) of the same length $m = (m_1, ..., m_T)$ where each $m_t \in \{0, 1\}$, selects which tokens will remain. The token at time t, x_t is then replaced with a special mask token $< m >$ if the mask is 0 and remains unchanged if the mask is 1.

1.2.2 The Generator

Authors of the paper chose seq2seq architecture for both generator and discriminator. The encoder reads in the masked sequence, which we denote as $m(x)$, where the mask is applied element-wise. The encoder provides access to a future context for the MaskGAN during decoding. As in standard

language-modeling, the decoder fills in the missing tokens auto-regressively, however, it is now conditioned on both the masked text $m(x)$ as well as what it has filled-in up to that point. The generator decomposes the distribution over the sequence into an ordered conditional sequence. $P(\hat{x_1}, ..., \hat{x_T}|m(x)) = \prod_{t=1}^{T} P(\hat{x_T}|\hat{x_1}, ..., \hat{x_{t-1}}, m(x))$

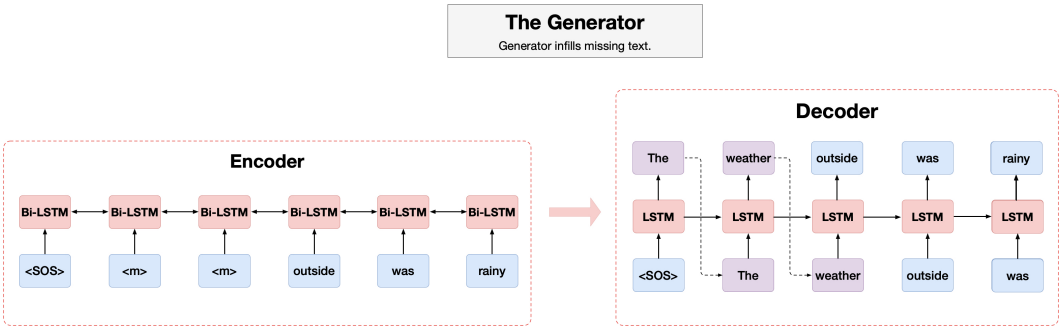

Figure 1: seq2seq generator architecture. Blue boxes represent known tokens and purple boxes are imputed tokens. We demonstrate a sampling operation via the dotted line. The encoder reads in a masked sequence, where masked tokens are denoted by an $< m >$ token, and then the decoder imputes the missing tokens by using the encoder hidden states.

1.2.3 The Discriminator

The discriminator is given the filled-in sequence from the generator, but importantly, it is given the original real context $m(x)$. Discriminator computes the probability of each token $\tilde{x_t}$ being real given the true context of the masked sequence $m(x)$.

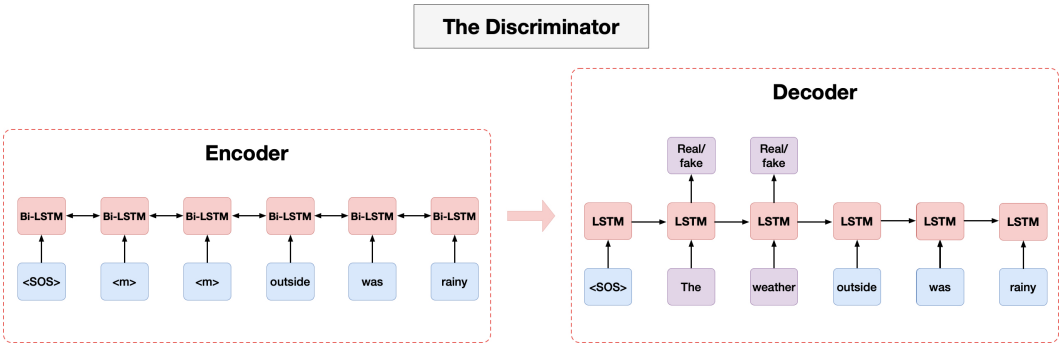

Figure 2: seq2seq discriminator architecture. Blue boxes represent known tokens and purple boxes are tokens that were masked . The encoder reads in a masked sequence, where masked tokens are denoted by an $< m >$ token, and then the decoder calculates the probability of given imputed token being real.

1.2.4 Training

MaskedGAN model is not fully-differentiable due to the sampling operations on the generator's probability distribution to produce the next token. Therefore, we need to use Reinforcement Learning magic to make this model work. Although we have some knowledge of RL, we need to learn more about actor critic algorithm to understand the training process.

2 Work log

2.1 Datasets overview

We are supposed to produce our experiments on IMDB and PTB datasets. We have to notice that in the paper some of the evaluations of the quality of generated texts were made by humans.

2.1.1 IMDB

The IMDB dataset consists of 100,000 movie reviews taken from IMDB The dataset is divided into 25,000 labeled training instances, 25,000 labeled test instances and 50,000 unlabeled training instances. As authors of the paper, we use the first 40 words of all of the training reviews. From samples on IMDB dataset given in the paper, we figure out that authors removed all punctuation marks and actually that is all preprocessing that was made for this dataset. In fact, they also have not added EOS token for an end of a sentence and the only indicator of a start of a new sentence is capitalized letters in words. What was less obvious is SOS token. Authors did not mention anything about it but we decided that it is necessary for the generator if a first word is masked.
Summing up our work with IMDB dataset:
Preprocessing $\rightarrow$ Building the vocabulary $\rightarrow$ Reviews encoding $\rightarrow$ DataLoaders

2.1.2 PTB

The Penn Treebank dataset has a vocabulary of 10,000 unique words. The training set contains 930,000 words, the validation set contains 74,000 words and the test set contains 82,000 words. Like in the paper, we train on the training partition. Fortunately, this dataset was preprocessed and encoded, so we only load all the data in DataLoaders.

2.2 Pretraining and MaskMLE

2.2.1 Language model

So far we mostly worked on IMDB dataset. Before training the MaskGAN authors pretrained a language model to a validation perplexity of 105.6. We built a language model using LSTM but due to some of our fails, we retrain it for the third time. In addition, we used LSTM with simple dropout. However, in paper authors were using variational LSTM for language model but variational LSTM is not implemented in PyTorch and it is not trivial to quickly implement it by yourself.

2.2.2 MaskMLE = pretrained Generator)

Because of the masking technique we had some troubles with an implementation of the decoder part of the generator. Luckily, we passed this stage and now we are going to train it to validation perplexity of 87.1 (like in the paper).

2.2.3 Pretraining the Discriminator

After pretraining the MaskMLE we train our discriminator on both real and generated texts.

2.2.4 Further steps

- All the above models are building blocks of the MaskGAN. So, we have to pretrain them all like in the paper.
- Understand the training part.
- Train the MaskGAN model and evaluate the results.

References

[1] William Fedus, Ian Goodfellow, and Andrew M Dai. 2018. MaskGAN: Better text generation via filling in the __. arXiv preprint arXiv:1801.07736.