Neural Models Comparison in Natural Language Generation

Lixue Zhang, Sha Tong, Ran Xu, Wen Cui, Yifeng Liu

1 Motivation

In this project, we will explore different models for natural language generation. We firstly implement current state-of-art model which is utilizing Recurrent Neural Networks (RNNs). And in the meanwhile we also experiment Generative Adversarial Networks (GANs) which recently achieved breakthrough in image generation. Then we will compare their performance in the aspect of the quality of the text generated. We will expect some difficulty in training GANs due to the nature complexity of text process. We hope to gain some insight and handful experience in this project.

2 Dataset

We will use the TV transcripts from Big Bang Theory as a dataset. We have scraped 10 seasons(220 episodes) transcripts from this fan blog https://bigbangtrans.wordpress.com/. The dialogues distribution is listed in Table 1.

Character	Sheldon	Leonard	Penny	Howard	Raj	Amy	Bernadette	Rest
Dialogues	10949	9251	7276	5552	4181	3079	2437	8123

Table 1: The number of dialogues of major character in the whole dataset

3 Neural Models

3.1 RNN

There is a vast amount of data which is inherently sequential, such as speech, time series (weather, financial, etc.), sensor data, video, and text, just to mention some. Recurrent Neural Networks (RNNs) are a family of neural networks designed specifically for sequential data processing. Neural language models attempt to solve the problem of determining the likelihood of a sentence in the real world. Once we have such system in place, we can use it later to generate new text with some fantastic outcomes.

3.1.1 RNN for text generation

Humans base much of their understanding from context. Let's consider the following sequence – Paris is the largest city of_____. It is easy to fill the blank with France. This means that there is information about the last word encoded in the previous elements of the sequence.

The idea behind this is to exploit this sequential structure of the data. The name of this neural networks comes from the fact that they operate in a recurrent way. This means that the same operation is performed for every element of a sequence, with its output depending on the current input, and the previous operations.

This is achieved by looping an output of the network at time t with the input of the network at time $t+1$. These loops allow persistence of information from one time step to the next one.

We can explain this with the following example. Imagine we want to compute the likelihood of the sentence "And the boss is a cat named Joe". To do so we need to estimate the following probabilities,

$$p(And), p(the|And), p(boss|Andthe), ..., p(Joe|Andthebossisacatnamed)$$

3.1.2 RNN backward propagation

The forward path is as usual, the key point for the iteration of RNN is backward path also named as loss function. So we should define an appropriate loss function for the network to actually learn what we are expecting it to learn. The loss for a given sequence is the negative log probability that the model assigns to the correct output. This is $L = -log(p(w_1, w_2, ..., w_T))$. Using the chain rule and the fact that the log of a product equals the sum of the logs, we obtain, for a sentence x,

$$L(x) = -\sum_t logP_{model}(w_t = x_{t+1}) = -\sum_t log\mathbf{o}_t[x_{t+1}]$$

where $\mathbf{o}_t[x_{t+1}]$ is the element of the output softmax corresponding to the real word x_{t+1}.

With the loss defined and given that the whole system is differentiable, we can backpropagate the loss through all the previous RNN units and embedding matrices and update its weights accordingly.

3.2 GANs

In this section, we would like to train GANs for language generation. The major difference with respect to the previous model mentioned in Section 3.1 is that we don't need to pre-train the model. Text will be generated while the training process and we will employ an RNN based GANs for both generator and discriminator.

3.2.1 RNN generator

We will employ a GRU or LSTM based RNN for our generator using maximum likelihood estimation. The generator is fed by a noise vector z as the hidden state, and an embedded start-of-sequence symbol as input. The output will be a sequence of distributions over characters using a softmax layer over the hidden state at each time step. The loss of generator is [1]:

$$L_G = -\mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})]$$

3.2.2 RNN discriminator

The discriminator is a recurrent neural network(LSTM or GRU based) that receives a matrix M with the dimension of sequence length times the vocabulary size. And then determine if the matrix if from real data or fake data(output of the generator G). In particular, in the input matrix each row represents one-hot vector from the real data and softer distribution from the fake data. Then the loss of discriminator D becomes the equation below [1],

$$L_D = \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x}) - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2]$$

4 Evaluation

Automatic evaluation of generated text itself is already a challenge task. However we would like to compare the performance using the entropy loss of the generated text and semantic level measures such as fluency, readability and grammatic errors.

5 References

References

[1] Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, Lior Wolf. 2017. Language Generation with Recurrent Generative Adversarial Networks without Pre-training. *arXiv:1706.01399v2.*