Paper

- **Title:** SeqGAN Sequence Generative Adversarial Nets with Policy Gradient
- **Authors:** Lantao Yu, Weinan Zhang, Jun Wang, Yong Yu
- **arXiv link:** https://arxiv.org/abs/1609.05473

TL;DR

The paper describes a method to train a Generative Adversarial Network(GAN) to generate discrete sequential data (eg. text, music) by using an RNN as a generator and policy gradients for training, to account for the discreteness of the data.

Motivation

- Although LSTM is widely used for generating sequential data, Bengio et al. 2015 showed the exposure bias problem while generating with an LSTM at test time and suggested a curriculum learning method Scheduled Sampling for training to overcome it (Note: They won the MSCOCO image captioning challenge 2015 with this setup)
- However, Huszar 2015 showed theoretically that Scheduled Sampling is an inconsistent strategy i.e. it does not minimize the KL divergence between real distribution and the learned distribution.
- The paper aims to solve these problems by using a GAN for text generation.

Why not use vanilla GAN?

- Ian Goodfellow, the inventor of GAN, explains in this reddit comment why vanilla GAN does not work with text data.
- In addition, the discriminator can only give likelihood of an entire sequence and it is not exactly clear how to balance the likelihood for a partial sequence.

SeqGAN

- A generator G_θ produces sequence of tokens $Y_{1:T} = (y_1, y_2, ..., y_T)$ where $y_t \in \mathcal{Y}$ which is the vocabulary
- The Generator is modeled as a reinforcement learning agent.

- At timestep t, the state s is the current set of tokens produced $(y_1, y_2, ..., y_{t-1})$ and the action a is the next token y_t to select.
- The generator model here is an LSTM that outputs the action given the current state.
- $D_\phi(Y_{1:T})$ is a probability indicating how likely the sequence $Y_{1:T}$ is to be from the real data. During policy update, we consider this probability as the reward.
- As there is no intermediate reward, the objective of the generator model is to maximize the expected end reward:

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

Here, $Q_{D_\phi}^{G_\theta}(s_0, y_1)$ is the action value function.

- To estimate the Q function, use REINFORCE algorithms by Williams 1992 and use the estimated probability of being real by the discriminator as the end reward.

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

- This reward is only for a finished sequence. To evaluate the action-value for an intermediate state, we apply Monte Carlo search with a roll-out policy G_β to sample the last $T - t$ tokens.
- To reduce the variance and get more accurate assessment of the Q function, the roll-out policy is run starting from current state till the end of the sequence for N times to get a batch of output samples. Thus,

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:T-1}, a = y_t) = \begin{cases} D_\phi(Y_{1:T}) & t = T \\ \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y_{1:T}^n), Y_{1:T}^n \in MC^{G_\beta}(Y_{1:t}) & t < T \end{cases} \tag{2}$$

The discriminator

- The discriminator here is a CNN followed by fully connected layers.
- The discriminator has a straightforward loss function:

$$-\mathbb{E}_{Y \sim p_{data}}[log D_\phi(Y)] - \mathbb{E}_{Y \sim G_\theta}[log(1 - D_\phi(Y))]$$

Training algorithm

Algorithm 1 Sequence Generative Adversarial Nets

Require: generator policy G_θ; roll-out policy G_β; discriminator D_ϕ; a sequence dataset $\mathcal{S} = \{X_{1:T}\}$

1: Initialize G_θ, D_ϕ with random weights θ, ϕ.
2: Pre-train G_θ using MLE on $\mathcal{S}$
3: $\beta \leftarrow \theta$
4: Generate negative samples using G_θ for training D_ϕ
5: Pre-train D_ϕ via minimizing the cross entropy
6: **repeat**
7:     **for** g-steps **do**
8:         Generate a sequence $Y_{1:T} = (y_1, \dots, y_T) \sim G_\theta$
9:         **for** t in $1 : T$ **do**
10:             Compute $Q(a = y_t; s = Y_{1:t-1})$ by Eq. (4)
11:         **end for**
12:         Update generator parameters via policy gradient Eq. (8)
13:     **end for**
14:     **for** d-steps **do**
15:         Use current G_θ to generate negative examples and combine with given positive examples $\mathcal{S}$
16:         Train discriminator D_ϕ for k epochs by Eq. (5)
17:     **end for**
18:     $\beta \leftarrow \theta$
19: **until** SeqGAN converges

Thus, after pretraining the generator in step 2, the exploration policy β is set to be the same as θ i.e. an LSTM is trained using Maximum likelihood estimate initially. We will improve this LSTM using SeqGAN setup but use the same distribution learned by it for exploration.

Gradient update of the Generator

- This is a math heavy section that shows how to get gradients of (1), if you are only interested in the implementation, you can check out the code here.

The setup

- Note that the state transition function δ is deterministic, i.e. when $s = Y_{1:t-1}, s' = Y_{1:T}$; $\delta^a_{s,s'} = 1$ if $a = y_t$ and $\delta^{a'}_{s,s'} = 0$ otherwise.

- Also, we set the intermediate reward $\mathcal{R}^a_s = 0$ is 0.

The Q function and Value function will be:

- $Q^{G_\theta}(s = Y_{1:t-1}, a = y_t) = \mathcal{R}_s^a + \sum_{s' \in S} \delta_{ss'}^a V^{G_\theta}(s')$
- $V^{G_\theta}(s = Y_{1:t-1}) = \sum_{y_t \in \mathcal{Y}} G_\theta(y_t|Y_{1:t-1}) \cdot Q^{G_\theta}(Y_{1:t-1}, y_t)$
- Note that $Q^{G_\theta}(Y_{1:t-1}, y_t) = V^{G_\theta}(Y_{1:T})$ using simple substitution of $\mathcal{R}_s^a = 0$

Hence the value function for our start state s_0 will be:

$$V^{G_\theta}(s_0) = \sum_{y_1 \in \mathcal{Y}} G_\theta(y_1|s_0) \cdot Q^{G_\theta}(s_0, y_1)$$

This is nothing but our objective function from (1), it makes intuitive sense to maximize the value function for the start state.
Let's differentiate it and see where we can go:

$$\begin{aligned}
&\nabla_\theta J(\theta) \\
&= \nabla_\theta[\sum_{y_1 \in \mathcal{Y}} G_\theta(y_1|s_0) \cdot Q^{G_\theta}(s_0, y_1)] \quad \text{(on substituting the objective)} \\
&= \sum_{y_1 \in \mathcal{Y}} [\nabla_\theta G_\theta(y_1|s_0) \cdot Q^{G_\theta}(s_0, y_1) + G_\theta(y_1|s_0) \cdot \nabla_\theta V^{G_\theta}(Y_{1:1})] \quad \text{(on applying product rule of derivatives)} \\
&= \sum_{y_1 \in \mathcal{Y}} [\nabla_\theta G_\theta(y_1|s_0) \cdot Q^{G_\theta}(s_0, y_1) + G_\theta(y_1|s_0) \nabla_\theta[\sum_{y_2 \in \mathcal{Y}} G_\theta(y_2|Y_{1:1}) Q^{G_\theta}(Y_{1:1}, y_2)] \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(on substituting the value function)} \\
&= \sum_{y_1 \in \mathcal{Y}} [\nabla_\theta G_\theta(y_1|s_0) \cdot Q^{G_\theta}(s_0, y_1) + G_\theta(y_1|s_0) \sum_{y_2 \in \mathcal{Y}} [\nabla_\theta G_\theta(y_2|Y_{1:1}) Q^{G_\theta}(Y_{1:1}, y_2) \\
&\qquad\qquad + G_\theta(y_2|Y_{1:1}) \nabla_\theta Q^{G_\theta}(Y_{1:1}, y_2)] \quad \text{(on applying product rule of derivatives)} \\
&= \sum_{y_1 \in \mathcal{Y}} [\nabla_\theta G_\theta(y_1|s_0) \cdot Q^{G_\theta}(s_0, y_1) + \sum_{Y_{1:1}} P(Y_{1:1}|s_0; G_\theta) \cdot \sum_{y_2 \in \mathcal{Y}} \nabla_\theta G_\theta(y_2|Y_{1:1}) Q^{G_\theta}(Y_{1:1}, y_2) \\
&+ \sum_{Y_{1:2}} P(Y_{1:2}|s_0; G_\theta) \nabla_\theta V^{G_\theta}(Y_{1:2})
\end{aligned}$$

($G_\theta$is nothing but the probability of generation and then using chain rule of probability)

$$\begin{aligned}
&\vdots \\
&= \sum_{t=1}^{T} \sum_{Y_{1:t-1}} P(Y_{1:t-1}|s_0, G_\theta) \sum_{y_t \in \mathcal{Y}} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q^{G_\theta}(Y_{1:t-1}, y_t) \quad \text{(just unrolling the loop)} \\
&= \sum_{t=1}^{T} \mathbb{E}_{Y_{1:t-1} \sim G_\theta}[\sum_{y_t \in \mathcal{Y}} \nabla_\theta G_\theta(y_t|Y_{1:t-1}) \cdot Q^{G_\theta}(Y_{1:t-1}, y_t)]
\end{aligned}$$

Using the above derivative and taking expectations by sampling, we can update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$