Maximum-Likelihood Augmented Discrete Generative Adversarial Networks (MaliGAN)

Tong Che, et al.

Presenting: Yengeny Tkach

https://qdata.github.io/deep2Read/
Executive Summary

• MaliGAN is a GAN based generative model for discrete sequences, trained using RL methods for variance reduction.
• The optimization objective of the generative function is replaced in this work with $KL(Q||P_G)$ where $P_G$ is the distribution of the generated data and $Q$ is a self-normalized importance sampling (SIS) estimation of the data distribution.
• To reduce the variance of the gradient signal the authors mix sampling from the true data and the generated data distributions.
Outline

- GAN – Basic Idea
- Discrete data challenges
- Importance Sampling
- MaliGAN – Basic
- policy gradient
- Sequential MaliGAN with Mixed MLE Training
- seqGAN
- Experiments
Basic Idea of GAN
GAN Formally

• Value Function:
\[ V(\mathbb{P}, G_\theta, D_\phi) = E_{x \sim P}[\log D(x)] + E_{x \sim Q}[\log(1 - D(x))] \]
\[ = E_{x \sim P}[\log D(x)] + E_{z \sim h(z)}[\log(1 - D(G(z)))] \]

• Monte-Carlo Approximation:
\[ \tilde{V}(\mathbb{P}, G_\theta, D_\phi) = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^i))) \]

• Discriminator target:
\[ \max_{\phi} \tilde{V}(\mathbb{P}, G_\theta, D_\phi) \]

• Generator target:
\[ \min_{\theta} \max_{\phi} \tilde{V}(\mathbb{P}, G_\theta, D_\phi) \]
Algorithm

Initialize $\phi_d$ for D and $\theta_g$ for G

In each training iteration:

- Sample $m$ examples $\{x^1, x^2, ..., x^m\}$ from data distribution $P(x)$
- Sample $m$ noise samples $\{z^1, z^2, ..., z^m\}$ from the prior $h(z)$
- Obtaining generated data $\{\tilde{x}^1, \tilde{x}^2, ..., \tilde{x}^m\}$, $\tilde{x}^i = G(z^i)$
- Update discriminator parameters $\theta_d$ to maximize
  \[
  \tilde{V} = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(\tilde{x}^i)\right)
  \]
  \[
  \phi_d \leftarrow \phi_d + \eta \nabla \tilde{V}(\phi_d)
  \]
- Sample another $m$ noise samples $\{z^1, z^2, ..., z^m\}$ from the prior $P_{prior}(z)$
- Update generator parameters $\theta_g$ to minimize
  \[
  \tilde{V} = \frac{1}{m} \sum_{i=1}^{m} \log D(x^i) + \frac{1}{m} \sum_{i=1}^{m} \log \left(1 - D(G(z^i))\right)
  \]
  \[
  \theta_g \leftarrow \theta_g - \eta \nabla \tilde{V}(\theta_g)
  \]
GAN for Discrete sequences

Adapting GAN to generating discrete data is challenging:
• How do we calculate $\nabla \tilde{V}(\theta_g)$? $G(z)$ is discontinuous.
• How can we reduce the variance of $\nabla \tilde{V}(\theta_g)$ for long sequence generation
Importance Sampling

\[ E_{x \sim p}[f(x)] = \int f(x)p(x)dx \]

\[ = \int f(x)\frac{p(x)}{q(x)}q(x)dx \]

\[ = \int f(x)w(x)q(x)dx \]

\[ = E_{x \sim q}[f(x)w(x)] \]

\[ = \frac{E_{x \sim q}[f(x)w(x)]}{E_{x \sim q}[w(x)]} \]

\[ w(x) = \frac{p(x)}{q(x)} \]

In case \( p \) or \( q \) are scaled density functions

\[ w(x) \text{ - Importance Weights} \]
Importance sampling in MaliGAN

Basic idea: optimal discriminator $D^*(x)$ holds:

$$D^*(x) = \frac{p_d(x)}{p_\theta(x) + p_d(x)} \iff p_d(x) = \frac{D(x)}{1 - D(x)} p_\theta(x)$$

Where $p_d(x)$ in true data distribution and $p_\theta(x)$ is generated. We can estimate $p_d(x)$ by $q(x)$:

$$q(x) = \frac{r_D(x)}{\mathbb{E}[r_D(x)]} p_\theta(x) \quad , \quad r_D(x) = \frac{D(x)}{1 - D(x)}$$

Generator loss:

$$L_G(\theta) = KL(q(x) \| p_\theta(x))$$

$$\nabla L_G(\theta) = -\mathbb{E}_{p_d} [\nabla_\theta \log p_\theta(x)] = -\mathbb{E}_{p_\theta} \left[ \frac{r_D(x)}{\mathbb{E}[r_D(x)]} \nabla_\theta \log p_\theta(x) \right]$$
Why self normalization?

If we would use $r_D(x)$:

- In the beginning of the training $D(x)$ close to 0 and $r_D(x)$ will offer a very poor gradient direction with very little change.
- For some instances during the training $D(x)$ will be close to 1 and $r_D(x)$ will explode.
- This ensures that the model can always learn something as long as there exist some generations better than others and controls the decreases the gradient variance.
MaliGAN Algorithm

Algorithm 1 MaliGAN

Require: A generator \( p \) with parameters \( \theta \).
A discriminator \( D(x) \) with parameters \( \theta_d \).
A baseline \( b \).
1: for number of training iterations do
2: for \( k \) steps do
3: Sample a minibatch of samples \( \{x_i\}_{i=1}^{m} \) from \( p_\theta \).
4: Sample a minibatch of samples \( \{y_i\}_{i=1}^{m} \) from \( p_d \).
5: Update the parameter of discriminator by taking gradient ascend of discriminator loss
\[
\sum_i [\nabla_{\theta_d} \log D(y_i)] + \sum_i [\nabla_{\theta_d} \log(1 - D(x_i))]
\]
6: end for
7: Sample a minibatch of samples \( \{x_i\}_{i=1}^{m} \) from \( p_\theta \).
8: Update the generator by applying gradient update
\[
\sum_{i=1}^{m} \left( \frac{r_D(x_i)}{\sum_i r_D(x_i)} - b \right) \nabla \log p_\theta(x_i)
\]
9: end for
Policy Gradient

- \( J(\theta) \) the expected reward under a stochastic policy \( \pi_\theta \)
- \( r(\tau) \) is the reward of trajectory \( \tau \)

\[
J(\theta) = E_{\tau \sim \pi_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau) d\tau
\]

- Stochastic policy gradient:

\[
\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau
\]

\[
= E_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]
\]

- In discrete GANs \( \pi_\theta \) is the generator \( G_\theta \) that produces a distribution over discrete objects (actions)
- \( r(\tau) \) in MaliGAN is \( \frac{r_D(x)}{\mathbb{E}[r_D(x)]} \)
\( \pi(\tau) \) is defined as:

\[
\pi_\theta(s_1, a_1, \ldots, s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t)p(s_{t+1}|s_t, a_t)
\]

Take the log:

\[
\log \pi_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t)
\]

The first and the last term does not depend on \( \theta \) and can be removed.

\[
\nabla_\theta \left[ \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \log p(s_{t+1}|s_t, a_t) \right]
\]
mixed MLE-MaliGAN

To further decrease the variance that maybe accumulated over long sequences:

• use the training data for $N$ time steps and switch to free running mode for the remaining $T-N$ time steps.
• For the first $N$ tokens, that are from the training data, the generator objective is MLE and for the rest is the MaliGAN

\[
\nabla L_G = \mathbb{E}_q[\nabla \log p_\theta(x)] \\
= \mathbb{E}_{p_d}[\nabla \log p_\theta(x_{\leq N})] + \mathbb{E}_q[\nabla \log p_\theta(x_{> N}|x_{< N})] \\
= \mathbb{E}_{p_d}[\nabla \log p_\theta(x_0, x_1, \ldots x_T)] \\
+ \frac{1}{Z} \mathbb{E}_{p_\theta}\left[\sum_{t=N+1}^{L} r_D(x) \nabla \log p_\theta(a_t|s_t)\right]
\]
mixed MLE-MaliGAN

• for each $0 \leq N \leq T$:

\[ \nabla L_G^N \approx \sum_{i=1, j=1}^{m, n} \left( \frac{r_D(x_{i,j})}{\sum_j r_D(x_{i,j})} - b \right) \nabla \log p_\theta(x_{i,j}^N|x_{i,j}^{\leq N}) \]

\[ + \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{N} p_\theta(a_t^i|s_t^i) = E_N(x_{i,j}) \]

(4)

• During the training procedure N is decreased from T towards 0
Algorithm 2 Sequential MaliGAN with Mixed MLE Training

Require: A generator \( p \) with parameters \( \theta \).
A discriminator \( D(x) \) with parameters \( \theta_d \).
Maximum sequence length \( T \), step size \( K \).
A baseline \( b \), sampling multiplicity \( m \).

1: \( N = T \)
2: Optional: Pretrain model using pure MLE with some epochs.
3: for number of training iterations do
4: \( N = N - K \)
5: for \( k \) steps do
6: Sample a minibatch of sequences \( \{ y_i \}_{i=1}^{m} \) from \( p_d \).
7: While keeping the first \( N \) steps the same as \( \{ y_i \}_{i=1}^{m} \),
sample a minibatch of sequences \( \{ x_i \}_{i=1}^{m} \) from \( p_{\theta} \) from
time step \( N \).
8: Update the discriminator by taking gradient ascend of
discriminator loss.
\[
\sum_{i}[\nabla_{\theta_d} \log D(y_i)] + \sum_{i}[\nabla_{\theta_d} \log(1 - D(x_i))]
\]
9: end for
10: Sample a minibatch of sequences \( \{ x_i \}_{i=1}^{m} \) from \( p_d \).
11: For each sample \( x_i \) with length larger than \( N \) in the mini-
batch, clamp the generator to the first \( N \) words of \( s \), and
freely run the model to generate \( m \) samples \( x_{i,j}, j = 1, \cdots m \) till the end of the sequence.
12: Update the generator by applying the mixed MLE-Mali
gradient update
\[
\nabla L_G^N \approx \sum_{i=1, j=1}^{m, n} \left( \frac{r_D(x_{i,j})}{\sum_j r_D(x_{i,j})} - b \right) \nabla \log p_{\theta}(x_{i,j}^{>N} | x_{i,j}^{\leq N})
\]
\[
+ \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{N} p_{\theta}(a_{t} | s_{t}^{i})
\]
13: end for
Policy Gradient

- Alternative forms:

\[
g = \mathbb{E}\left[ \sum_{t=0}^{\infty} \Psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right],
\]

where \(\Psi_t\) may be one of the following:

1. \(\sum_{t=0}^{\infty} r_t\): total reward of the trajectory.
2. \(\sum_{t'=t}^{\infty} r_{t'}\): reward following action \(a_t\).
3. \(\sum_{t'=t}^{\infty} r_{t'} - b(s_t)\): baselined version of previous formula.
4. \(Q^\pi(s_t, a_t)\): state-action value function.
5. \(A^\pi(s_t, a_t)\): advantage function.
6. \(r_t + V^\pi(s_{t+1}) - V^\pi(s_t)\): TD residual.

The latter formulas use the definitions

\[
V^\pi(s_t) := \mathbb{E}_{s_{t+1:\infty}, a_{t:\infty}} \left[ \sum_{l=0}^{\infty} r_{t+l} \right]
\]

\[
Q^\pi(s_t, a_t) := \mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty}} \left[ \sum_{l=0}^{\infty} r_{t+l} \right]
\]

\[
A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function)}.
\]

Fig. 1. A general form of policy gradient methods. (Image source: Schulman et al., 2016)
SeqGAN Algorithm

\[ \nabla J(\theta) = \mathbb{E} \sum_{t=1}^{T} Q(y_t, Y_{1:t-1}) \nabla \log p_\theta(y_t | Y_{1:t-1}) \]

\[ Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t) = \]

\[
\begin{cases} 
\frac{1}{N} \sum_{n=1}^{N} D_\phi(Y^n_{1:T}), & Y^n_{1:T} \in \text{MC}^{G_\beta}(Y_{1:t}; N) \quad \text{for} \quad t < T \\
D_\phi(Y_{1:t}), & \text{for} \quad t = T,
\end{cases}
\]
SeqGAN Algorithm

**Algorithm 1 Sequence Generative Adversarial Nets**

**Require:** generator policy $G_\theta$; roll-out policy $G_\beta$; discriminator $D_\phi$; a sequence dataset $S = \{X_{1:T}\}$

1: Initialize $G_\theta$, $D_\phi$ with random weights $\theta, \phi$.
2: Pre-train $G_\theta$ using MLE on $S$
3: $\beta \leftarrow \theta$
4: Generate negative samples using $G_\theta$ for training $D_\phi$
5: Pre-train $D_\phi$ via minimizing the cross entropy
6: repeat
7: for g-steps do
8: Generate a sequence $Y_{1:T} = (y_1, \ldots, y_T) \sim G_\theta$
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) $\theta \leftarrow \theta + \alpha_h \nabla_\theta J(\theta)$, \hspace{1cm} (8)
13: end for
14: for d-steps do
15: Use current $G_\theta$ to generate negative examples and combine with given positive examples $S$
16: Train discriminator $D_\phi$ for $k$ epochs by Eq. (5) $\min_{\phi} -\mathbb{E}_{y \sim p_{data}}[\log D_\phi(Y)] - \mathbb{E}_{y \sim G_\theta}[\log (1 - D_\phi(Y))]$. \hspace{1cm} (5)
17: end for
18: $\beta \leftarrow \theta$
19: until SeqGAN converges
MaliGAN with MCTS

- Alternative loss function where $r(\tau)$ is replaced by $Q(a, s)$

$$\nabla L_G(\theta) \approx \frac{\sum_i L_i}{m} \sum_{i,t} Q(a_t^i, s_t^i) \nabla \log p_\theta(a_t^i | s_t^i)$$

- Che et al. use MCTS
Experiments

- Discrete MNIST

*Figure 2. Samples generated by REINFORCE-like model (left) and by MaliGAN (right).*
Experiments

- Chinese poem generation

<table>
<thead>
<tr>
<th>Model</th>
<th>Poem-5 BLEU-2</th>
<th>Poem-5 PPL</th>
<th>Poem-7 BLEU-2</th>
<th>Poem-7 PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLE</td>
<td>0.6934</td>
<td>564.1</td>
<td>0.3186</td>
<td>192.7</td>
</tr>
<tr>
<td>SeqGAN</td>
<td>0.7389</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MaliGAN-basic</td>
<td>0.7406</td>
<td>548.6</td>
<td>0.4892</td>
<td>182.2</td>
</tr>
<tr>
<td>MaliGAN-full</td>
<td>0.7628</td>
<td>542.7</td>
<td>0.5526</td>
<td>180.2</td>
</tr>
</tbody>
</table>

- Sentence-Level Language Modeling

<table>
<thead>
<tr>
<th></th>
<th>MLE</th>
<th>MaliGAN-basic</th>
<th>MaliGAN-full</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid-Perplexity</td>
<td>141.9</td>
<td>131.6</td>
<td>128.0</td>
</tr>
<tr>
<td>Test-Perplexity</td>
<td>138.2</td>
<td>125.3</td>
<td>123.8</td>
</tr>
</tbody>
</table>
Discussion

Main takeaways:
• Try to reduce the variance and keep the bias unchanged to stabilize learning.
• Off-policy gives us better exploration and helps us use data samples more efficiently.
• Experience replay (training data sampled from a replay memory buffer);
• Batch normalization;
References


https://medium.com/@jonathan_hui/rl-policy-gradients-explained-9b13b688b146