Generative Adversarial Networks

Improving performance

Daniel Yukimura

yukimura@impa.br

October 10, 2018

Generative Adversarial Networks (GANs)

Review:

We have two neural networks competing

- **Generator**: $G : \mathcal{H} \to \mathcal{X}$.
- **Discriminator**: $D : \mathcal{X} \to [0, 1]$

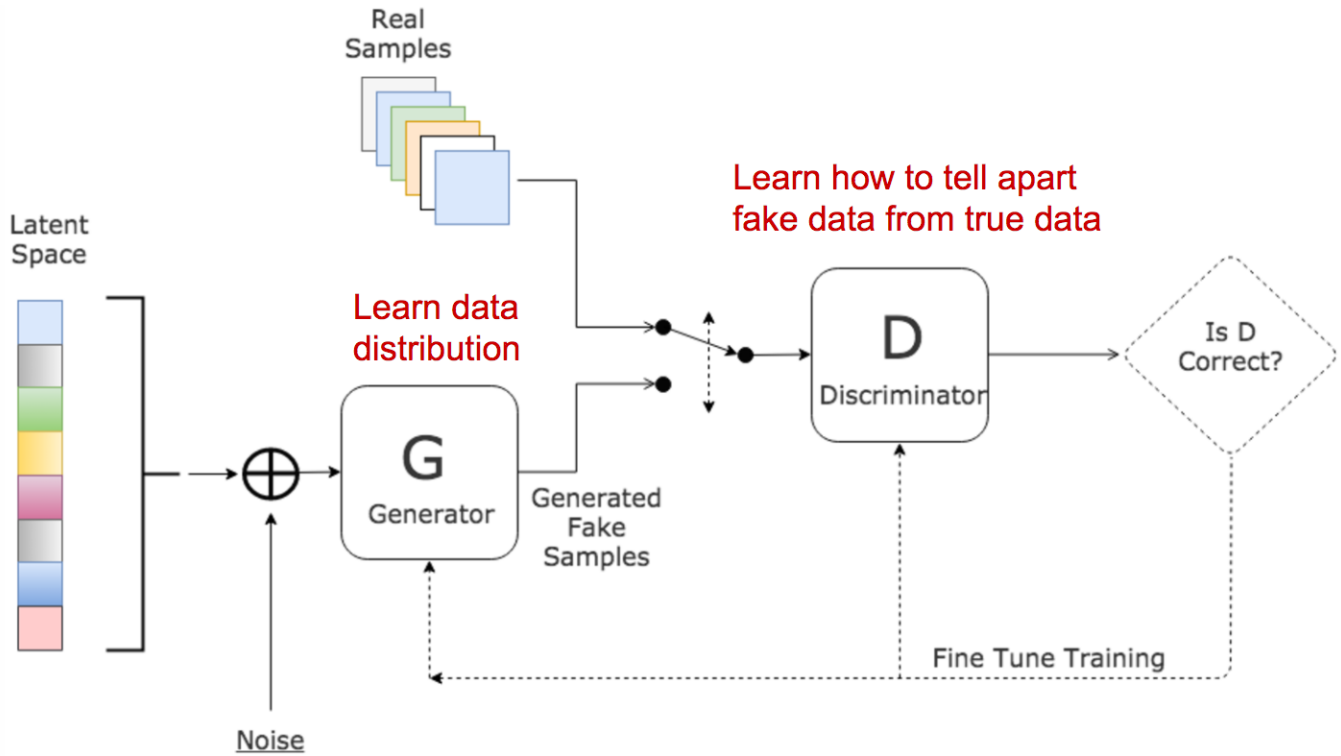
We want to find the parameters that reach equilibrium for the minmax game

$$G^* = \operatorname*{argmin}_G \max_D \frac{1}{2} \left(\mathbb{E}[\log D(X)] + \mathbb{E}[\log (1 - D(X_G))] \right). \tag{1}$$

where $X_G = G(H)$ are the "fake" samples, given by the distribution induced by G from p_H.

Generative Adversarial Networks (GANs)

Review:



Generative Adversarial Networks (GANs)

Game Saturation:

- The cost for the generator function $\mathbb{E}[\log(1 - D(X_G))]$ is useful in theory, but performs badly in practice.
- The "fake" samples can be so "unrealistic" in the beginning, that the response of D saturates.
- The cross-entropy approach implies that if the discriminator successfully rejects X_G with high confidence, the gradient vanishes, this is common in the end of supervising training, but here we have from the beginning

Generative Adversarial Networks (GANs)

Game Saturation:

- The cost for the generator function $\mathbb{E}[\log(1 - D(X_G))]$ is useful in theory, but performs badly in practice.
- The "fake" samples can be so "unrealistic" in the beginning, that the response of D saturates.
- The cross-entropy approach implies that if the discriminator successfully rejects X_G with high confidence, the gradient vanishes, this is common in the end of supervising training, but here we have from the beginning

Generative Adversarial Networks (GANs)

Game Saturation:

- The cost for the generator function $\mathbb{E}[\log(1 - D(X_G))]$ is useful in theory, but performs badly in practice.
- The "fake" samples can be so "unrealistic" in the beginning, that the response of D saturates.
- The cross-entropy approach implies that if the discriminator successfully rejects X_G with high confidence, the gradient vanishes, this is common in the end of supervising training, but here we have from the beginning

Generative Adversarial Networks (GANs)

Game Saturation:

- Instead we modify the cost when updating θ_G

$$c_G = -\frac{1}{2}\mathbb{E}[\log(D(X_G))] \tag{2}$$

- The formulation of this alternative game is heuristic, the idea is that each player has a strong gradient when that player is "losing" the game.

Generative Adversarial Networks (GANs)

Game Saturation:

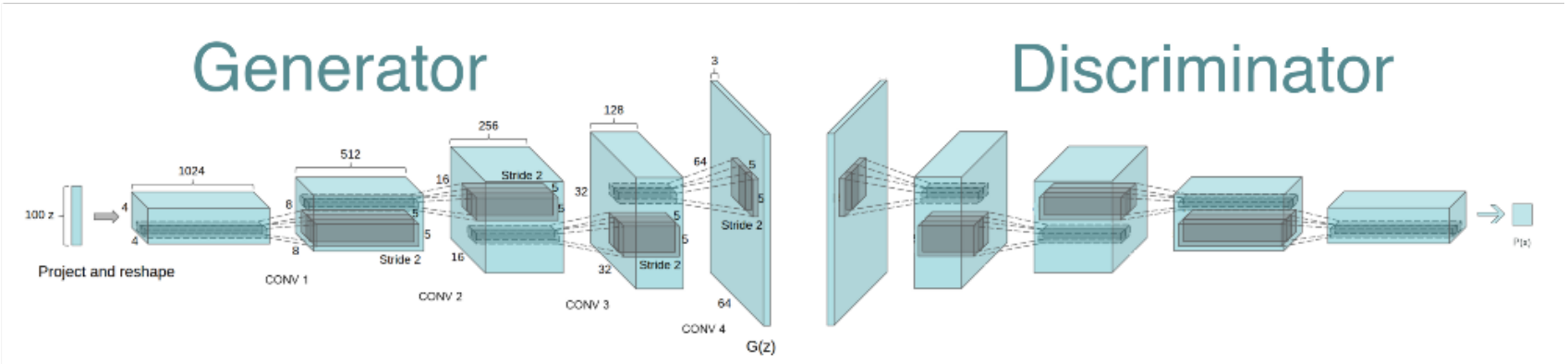
- Instead we modify the cost when updating θ_G

$$c_G = -\frac{1}{2}\mathbb{E}[\log(D(X_G))] \tag{2}$$

- The formulation of this alternative game is heuristic, the idea is that each player has a strong gradient when that player is "losing" the game.

Deep Convolutional GANs (DCGANs)

Reference: Radford et al. 2015, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.*



- PyTorch has DCGAN in his set of examples:

https://github.com/pytorch/examples.git

Deep Convolutional GANs (DCGANs)

The authors did an extensive model exploration and identified a set of rules that resulted in more stable training:

- Only convolutional layers, no fully-connected.
- Instead of pooling layers, use strided convolutions for D and strided transpose convolutions for G.
- Use batchnorm on both G and D.
- Use ReLu in G except for the output, where it uses Tanh.
- Use the LeakyReLu for all D.

colab notebook

Wasserstein GAN (WGAN)

This version presents better stability, ans prevents “model collapse”.

- The original optimization problem in (1) is equivalent to

$$G^* = \operatorname{argmin}_G D_{\mathrm{JS}}(p_{\mathrm{data}}, p_G) \tag{3}$$

  where D_{JS} is the **Jensen-Shannon Divergent**, a standard dissimilarity measure on the space of distributions.

- Nevertheless, D_{JS} doesn’t really capture a metric structure on the space of distributions.

- The alternative choice is to use the **Wasserstein metric**.

Wasserstein GAN

- The **Wasserstein distance** we'll use is defined by

$$W(\mu, \nu) = \inf_{\eta \in \Pi(\mu,\nu)} \mathbb{E}_{(X,\tilde{X})\sim\eta} \left[\|X - \tilde{X}\| \right] \tag{4}$$

  where $\Pi(\mu, \nu)$ are distributions with marginals μ and ν (coupling).

- This distribution is sometimes known as the *"earth moving distance"*, which gives the interpretation of the minimum mass one has to move to transform one distribution into the other.

Wasserstein GAN

- The idea would be finding a generator matching on this metric

$$G^* = \operatorname*{argmin}_G W(p_{data}, p_G) \tag{5}$$

  which is unfortunately unfeasible.

- We use instead an equivalent form, given by a result of Kantorovich and Rubinstein

$$W(\mu, \nu) = \max_{\|f\|_L \leq 1} \mathbb{E}_{\substack{x \sim \mu \\ \tilde{x} \sim \nu}} \left[f(x) - f(\tilde{x}) \right] \tag{6}$$

  where $\|f\|_L = \sup_{x,\tilde{x}} \frac{\|f(x) - f(\tilde{x})\|}{\|x - \tilde{x}\|}$ is the Lipschitz seminorm.

Wasserstein GAN

- Such formulation allows to formulate the problem as

$$\begin{aligned} G^* &= \operatorname*{argmin}_G W(p_{data}, p_G) \\ &= \operatorname*{argmin}_G \max_{\|D\|_L \leq 1} \left(\mathbb{E}_{X \sim p_{data}} [D(X)] - \mathbb{E}_{X_G \sim p_G} [D(X_G)] \right) \end{aligned} \tag{7}$$

- The main difficulty here is to be able to optimize D while restricted to the condition $\|D\|_L \leq 1$.

Wasserstein GAN

- The original way of training is by clipping the weights of D.
- The alternative is known as WGAN-GP, because it adds a gradient penalty, a smooth replace for the original discriminator search

$$D^* = \operatorname*{argmax}_D \mathbb{E}_{X \sim p_{\text{data}}}[D(X)] - \mathbb{E}_{X_G \sim p_G}[D(X_G)] - \lambda \mathbb{E}_{X_u \sim p_u}[(\|\nabla D(X_u)\| - 1)^2] \quad (8)$$

where p_u is a uniformly sample between a real (p_{data}) and a fake one (p_G). That is $X_u = UX + (1 - U)X_G$ for $U \sim U[0, 1]$.

Wasserstein GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Wasserstein GAN

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} =$ 0.0001, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ, the number of critic iterations per generator it n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters w_0, initial generator parameters θ_0.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U$
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Tips and Tricks

Train with labels:

- When available, using labels in any way in most cases results in a dramatic improvement in the quality of the samples.
- This was observed while the community explored ways of class-conditioning the distributions.
- Perhaps, the reason for improvement is that giving extra information facilitates optimization during training.
- Classes also point out information that we humans tend to focus, and therefore the model can develop a bias that pleases our perspective.
- Sometimes just by training the model class by class also helps convergence.

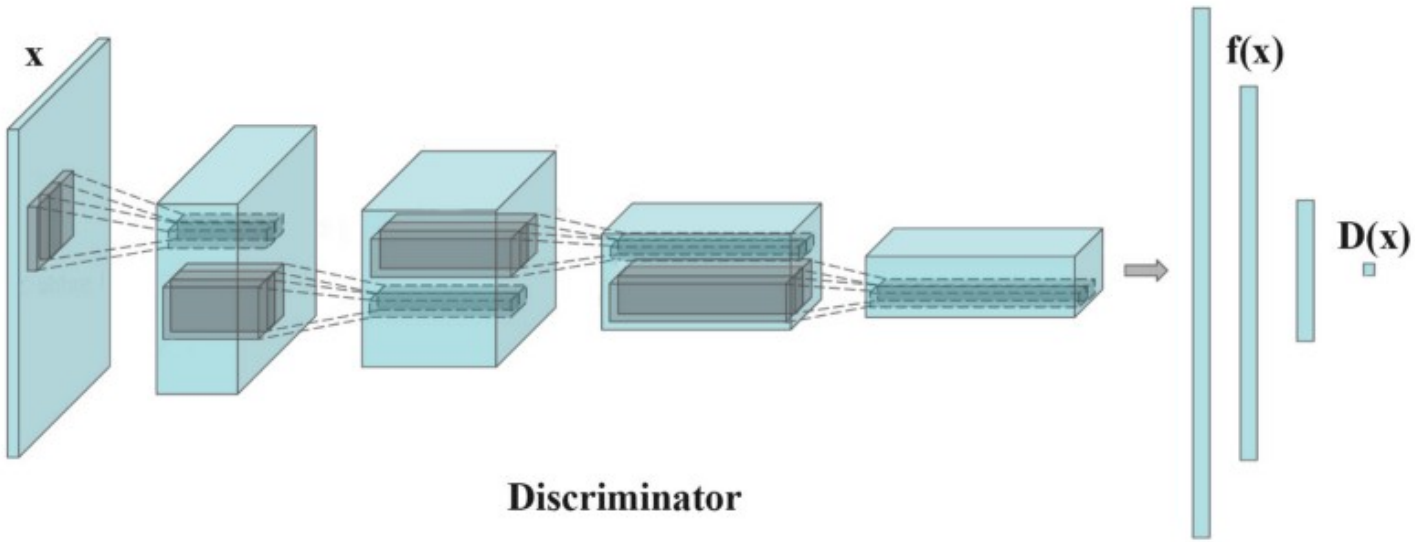
Tips and Tricks

Feature Matching:

- The idea is to improve statistical difference in the generator cost

$$\|\mathbb{E}_{x\sim p_{\text{data}}} f(X) - \mathbb{E}_{x_G\sim p_G} f(X_G)\|_2^2 \tag{9}$$

using an intermediate feature response instead of $D(x)$.



Tips and Tricks

Discriminator and generator network capacity

- Usually the discriminator is more complex than the generator. The opposite can even make the game diverge.
- When doing model selecting, starting from your base model.
  - Adding more neurons in a layer, usually doesnt harm the model, and convergence is maintained.
  - Adding more layers can go on the opposite direction, of actually harming performance. (This might change if your architecture has residual layer for example)
  - Is important to identify the bottleneck for your application.