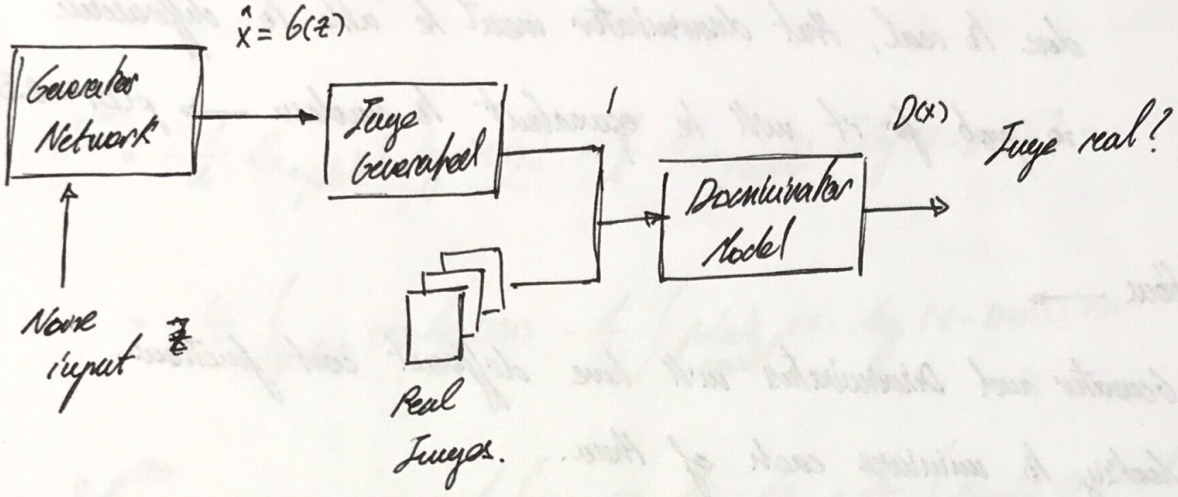
Generative Adversarial Networks

* Model →



* Generator is a differentiable network so it can be trained through gradient descent.

* Main intuition →

1. Generator →

* Generative model that maps input noise $\hat{x} = G(z)$.

* Its goal is to produce samples $\hat{x}$, from the distribution of training data $p(x)$.

* Trained to maximize that the generated model probability will be close to the training data.

2. Discriminator →

* Differentiate between generated and real images.

* $D(x) \in (0, 1)$

3. Generator and discriminator will be trained to fight against each other.

4. Goal → Ideal case the generator will create images so close to real, that discriminator won't be able to differentiate so prob for it will be equivalent to random → $p(D) = 0.5$

* Cost functions →

- Generator and Discriminators will have different cost functions, looking to minimize each of them.

- Based on Game Theory, Equilibrium →

  * There will be a point in which the G will create images so 0.5 of them are taken as real, and D will discriminate properly 0.5 → Balanced cost for 2 of them.

- Discriminator Loss function →

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2} E_{x \sim p_{data}} [\log D(x)] + -\frac{1}{2} E_{\hat{x} \sim p_{model}} [\log (1 - D(\hat{x}))]$$

$$= -\frac{1}{2} E_{x \sim p_{data}} [\log D(x)] - \frac{1}{2} E_{\hat{x} \sim p_{model}} [\log (1 - D(G(z)))]$$

  * Cross-entropy for binary classification.

  * Minimize → $D(x) = 1$ real data ; $D(\hat{x}) = 0$ for generated.

- Optimal minimization for Discriminator ⟶

  * Assuming continuous distribution, easier to make the point.

  * Note: p_{data}, p_{model} are non-zero everywhere, assumption.

$$L^{(D)} = -\frac{1}{2} E_{x \sim p_{data}} [\log D(x)] - \frac{1}{2} E_{x \sim p_{model}} [\log (1 - D(x))] =$$

$$= -\frac{1}{2} \int p_{data}(x) \log D(x) - \frac{1}{2} \int p_{model}(x) \log (1 - D(x)) \Rightarrow$$

$$\frac{\partial L^{(D)}}{\partial D(x)} = 0 \longrightarrow 0 = -\frac{1}{2} \int p_{data}(x) \frac{1}{D(x)} dx + \frac{1}{2} \int p_{model}(x) \frac{1}{1 - D(x)} dx$$

$$D(x) = \frac{p_{data}(x)}{p_{data}(x) + p_{model}(x)}$$

  * If D, G have enough capacity ⟶ $p_{model}(x) = p_{data}(x)$

    Nash equilibrium ⟶ $D(x) = 0.5$

- Zero-sum game ⟶

  * Discriminator ⟶

    - Wants to maximize the objective such that $D(x) \to 1$ and $D(G(z)) \to 0$.

  * Generator ⟶

    - Wants to minimize the objective such that $D(G(z)) \to 1$.

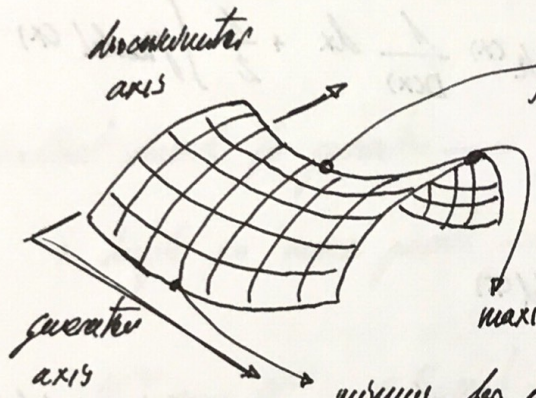
* In this scenario, we can think of the sum of losses as 0 →

$$\mathcal{L}^{(D)} + \mathcal{L}^{(G)} = 0.$$

* Find a way to minimize the maximum loss →

$$\min_{\theta^{(G)}} \max_{\theta^{(D)}} \left[+ \frac{1}{2} E_{x \sim p_{data}(x)} [\log(D(x))] + \frac{1}{2} E_z [\log(1 - D(G(z)))] \right]$$

notice you change from loss.



saddle point.
maximize discriminator,
minimize generator. zero-sum.
equilibrium.

maximum for only discriminator

minimum for only generator.

- Optimization →

* Gradient ascent for discriminator →

$$\theta^{(D)} \leftarrow \arg\max_{\theta^{(D)}} \left[\frac{1}{2} E_{x \sim p_{data}} [\log D(x)] + \frac{1}{2} E_z [\log(1 - D(G(z)))] \right]$$

* Gradient descent generator →

$$\theta^{(G)} \leftarrow \arg\min_{\theta^{(G)}} \frac{1}{2} E_z [\log(1 - D(G(z)))]$$

This term depends on values for $\theta^{(G)}$,

not $E_{x \sim p_{data}} [\log(D(x))]$

- Training algorithm →

- k gradient steps for generator → $k \equiv$ hyperparameter.

$m \equiv$ mini-batch input data

* Sample m noise points → $p(z) \Rightarrow \{z^{(1)}, z^{(2)}, z^{(3)}, \dots, z^{(m)}\}$

* Sample m data points → $p_{data}(x) \rightarrow \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

* Gradient (ascent) update → ascent

$$\theta^{(D)} \leftarrow \theta^{(D)} + \varepsilon \nabla_{\theta^{(D)}} \left(\frac{1}{m} \sum_{i=1}^{m} \left[\log(D(x^{(i)})) + \log(1 - D(G(z^{(i)}))) \right] \right)$$

- Gradient descent for generator →

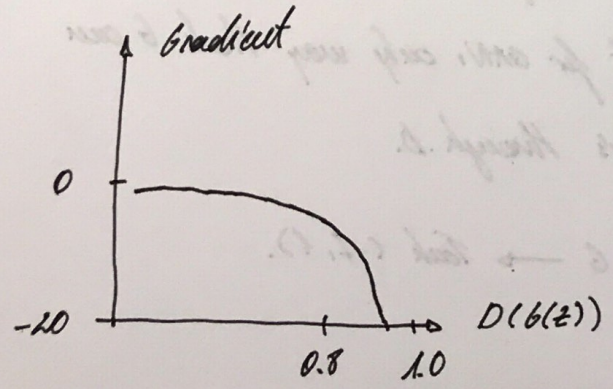
* Sample m noise points → $p(z) \rightarrow \{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$

$$\theta^{(G)} \leftarrow \theta^{(G)} - \varepsilon \nabla_{\theta^{(G)}} \left(\frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z^{(i)}))) \right)$$

- Ian Goodfellow → Common practice $k = 1$.

- Gradient problem on $\log(1 - D(G(z)))$ →

$$\frac{\partial}{\partial D(G(z))} \left(\log(1 - D(G(z))) \right) = - \frac{1}{1 - D(G(z))}$$

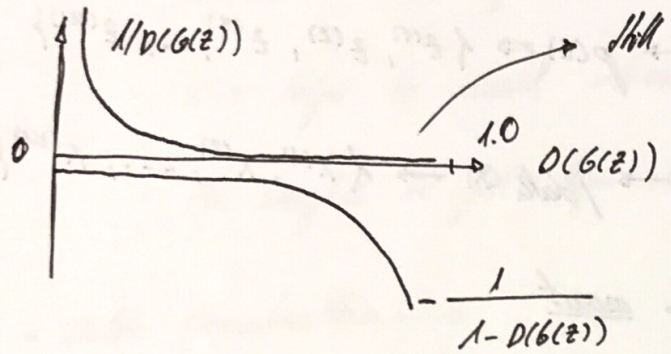


* $D(G(z)) = 0 \rightarrow \nabla$ short 0.

* At the beginning discriminator differentiate well, the update is really small.

* Instead alternative cost function for gradient →

Gradient $-\mathbb{E}_z[\log(D(G(z))]$ → Grad. max $-1/D(G(z))$



→ Still obtain minimum when $D(G(z)) \simeq 1$

* Algorithm changes our generator update →

$$\theta^{(G)} \leftarrow \theta^{(G)} + \varepsilon \nabla_{\theta^{(G)}} \left[\frac{1}{m} \sum_{i=1}^{m} \log D(G(z^{(i)})) \right]$$

* No longer a zero-sum.
* In practice, this is achieved by flipping labels.

- Base GAN →

Discriminator: Natural + Sigmoid ↑ Natural + LReLU ↑

Generator: Natural + Tanh ↑ Natural + LReLU

* At least one hidden layer →
  the ensures that they can represent any probability distribution, if given enough hidden units.

* Leaky ReLU →
  - Help to pass gradient through architecture.
  - Important for GAN, only way that G can learn is through D.

* Output of G → Tanh (-1, 1).

- Training GANs →

* One-sided label smoothing →

- Provide D labels for real data as 1-x, fake 0.

- 1-x helps to avoid extreme predictions. Prevent the weights to get high values → Regularization.

* Batch Normalization →

- Apply every layer, except output generator and input of discriminator.

- Batch normalize →

1. Read data for mini-batch

2. Generated data for mini-batch.

* Noise input →

- Use a gaussian distribution, not an uniform.

* Avoid sparse gradients → ReLU, Maxpool.