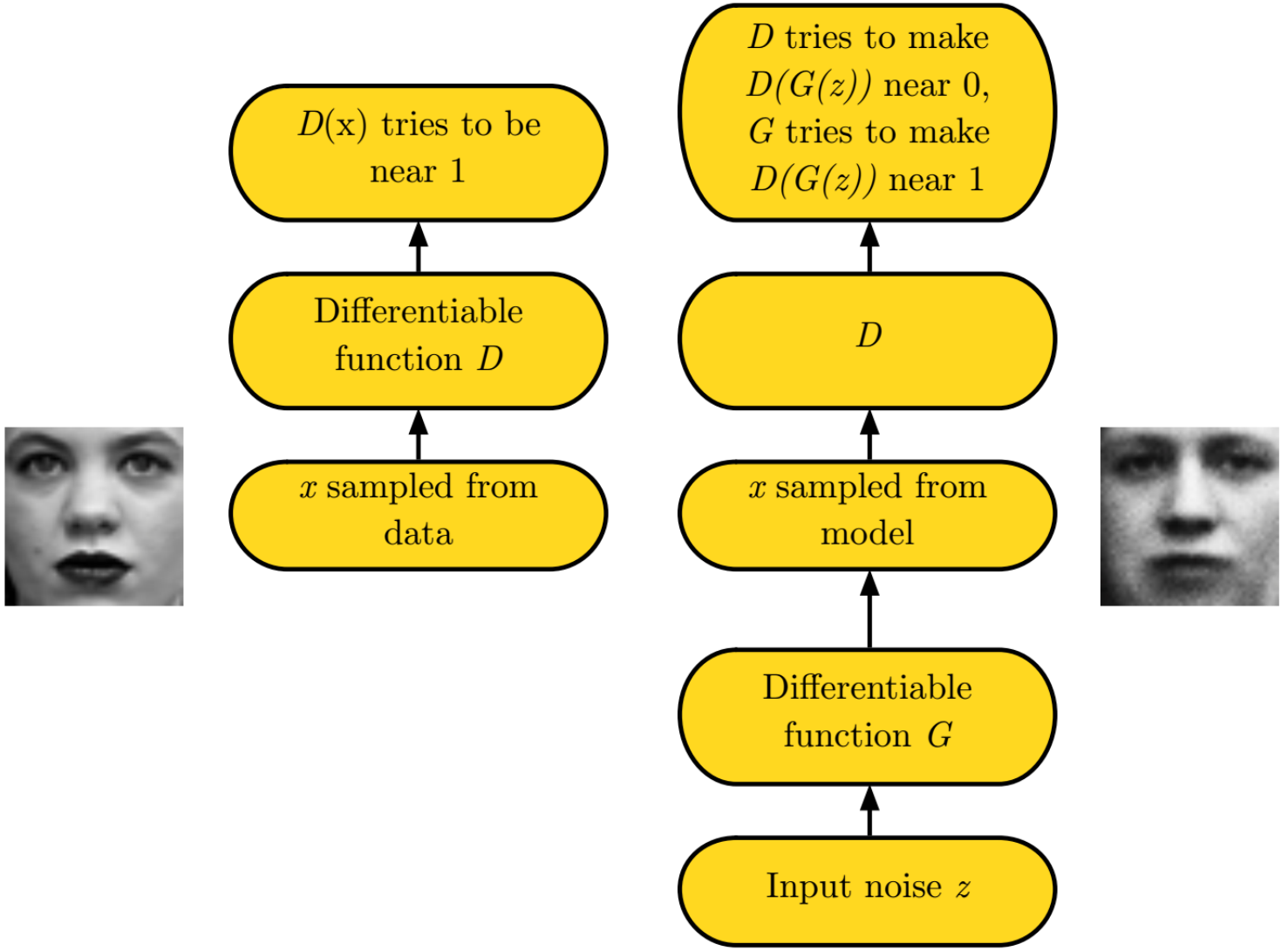
Generative Adversarial Networks



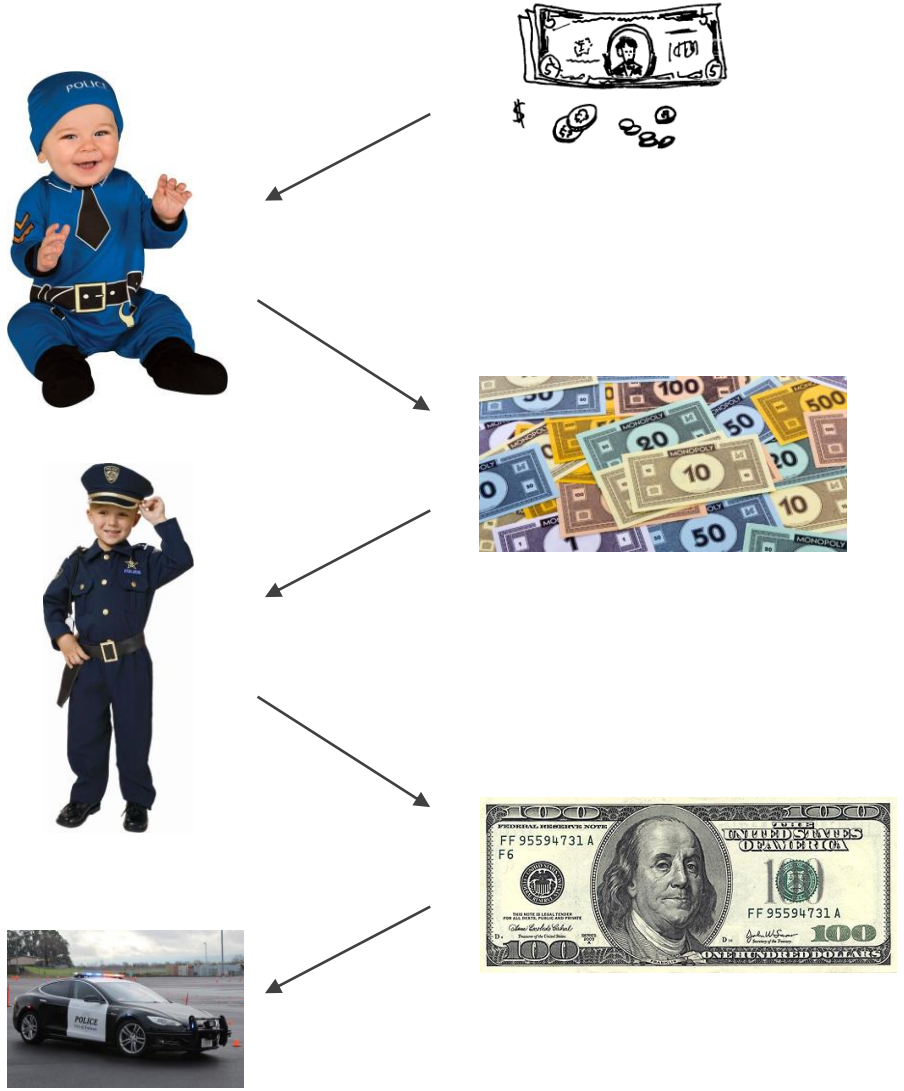
[NIPS 2016 Tutorial: Generative Adversarial Networks]

What is a GAN?

- Generative
  - You can sample novel input samples
  - *E.g.*, you can literally "create" images that never existed
- Adversarial
  - Our generative model G learns adversarially
  - by fooling an discriminative oracle model D
- Network
  - Implemented typically as a (deep) neural network
  - Easy to incorporate new modules
  - Easy to learn via backpropagation

GAN: Intuition

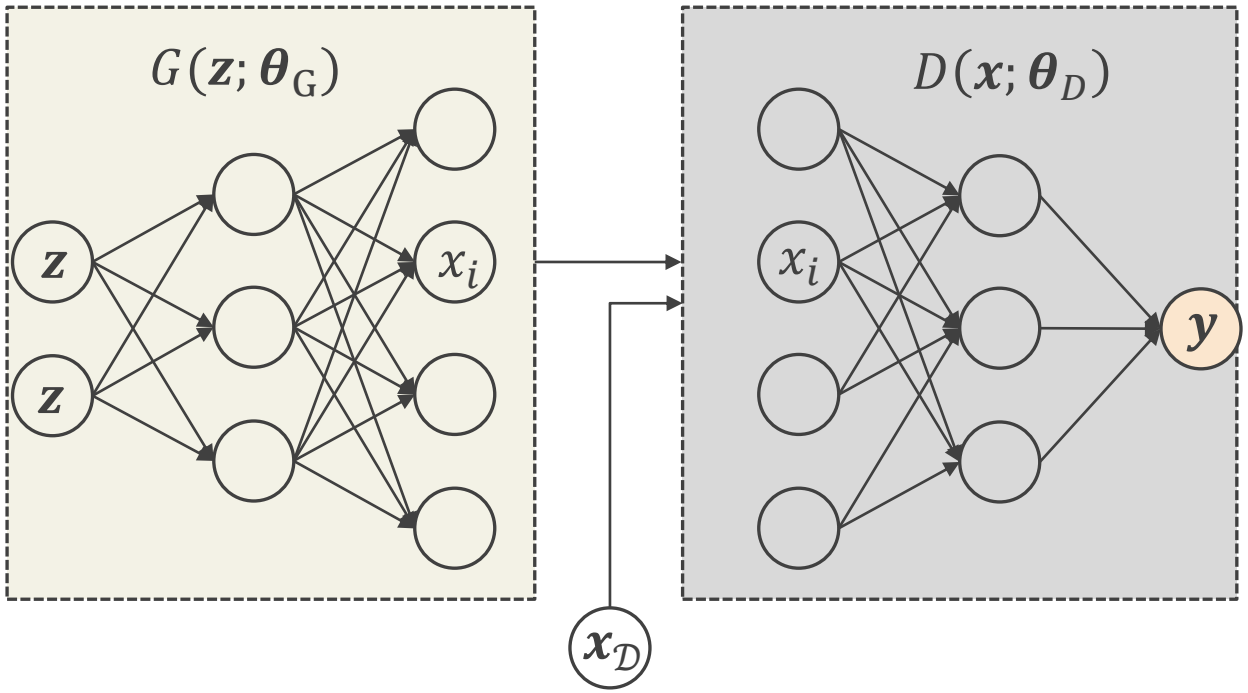
- Police: wants to detect fake money as reliably as possible
- Counterfeiter: wants to make as realistic fake money as possible
- The police forces the counterfeiter to get better
  - and vice versa
- Solution relates to Nash equilibrium



GAN architecture

- The GAN comprises two neural networks
  - Generator network $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}_{\mathrm{G}})$
  - Discriminator network $\mathrm{y} = D(\boldsymbol{x}; \boldsymbol{\theta}_{\mathrm{D}}) = \begin{cases} +1, \text{if } \mathbf{x} \text{ is predicted } 'real' \\ 0, \text{if } \boldsymbol{x} \text{ is predicted } 'fake' \end{cases}$

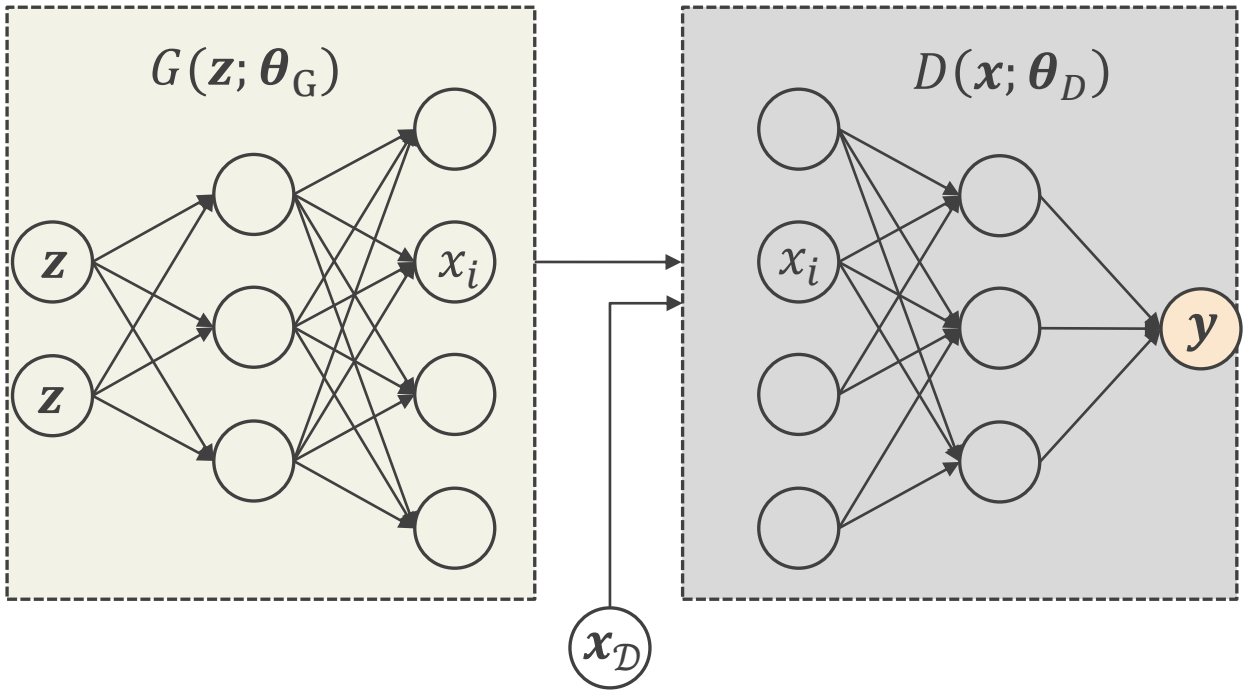
$\boldsymbol{z} \sim \mathcal{N}(0,1)$ or
$\boldsymbol{z} \sim \mathrm{Uniform}(0,1)$
…



GAN has no encoder

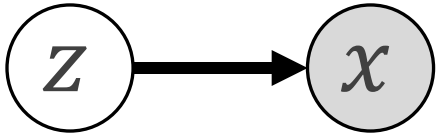
- Note: there is no 'encoder'
  - We cannot learn a representation for an image $\boldsymbol{x}$
  - We cannot compute a likelihood of a specific data point
  - At test time we can only generate new data points

$\boldsymbol{z} \sim \mathcal{N}(0,1)$ or
$\boldsymbol{z} \sim \text{Uniform}(0,1)$
…

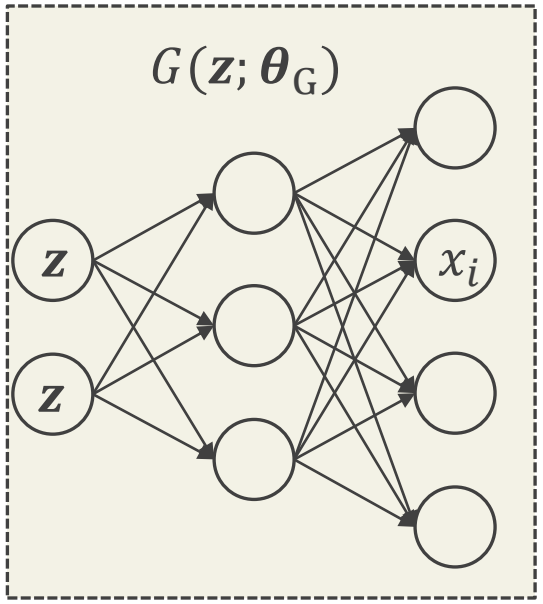


Generator network $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}_G)$

- Any differentiable neural network
- No invertibility requirement → More flexible modelling
- Trainable for any size of z
- Various density functions for the noise variable $\boldsymbol{z}$

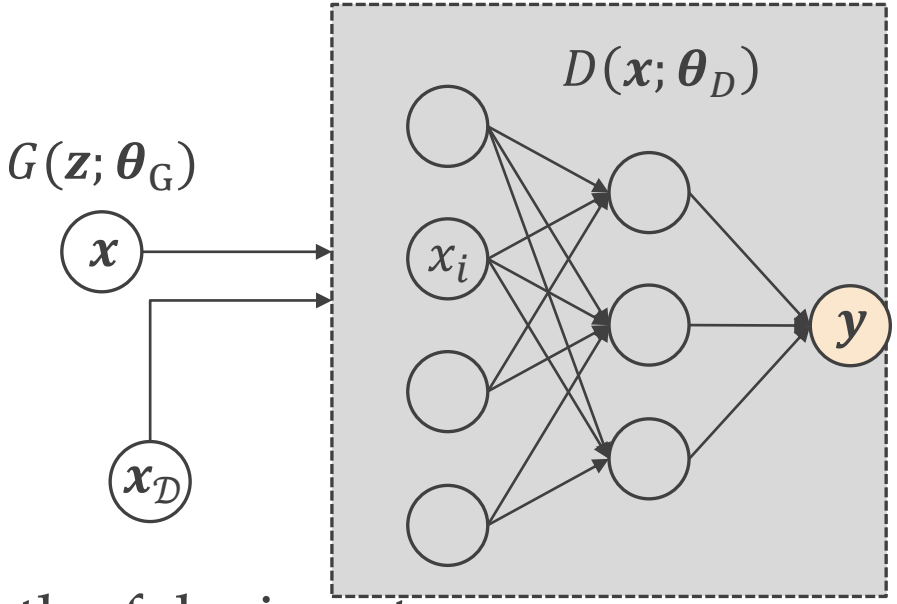


$\boldsymbol{z} \sim \mathcal{N}(0,1)$ or
$\boldsymbol{z} \sim \text{Uniform}(0,1)$
…



Discriminator network $\boldsymbol{y} = D(\boldsymbol{x}; \boldsymbol{\theta}_{\mathrm{D}})$

- Any differentiable neural network
- Receives as inputs
  - either real images from the training set
  - or generated images from the generator
  - usually a mix of both in mini-batches



- The discriminator must recognize the real from the fake inputs
- The discriminator loss

$$J_D(\boldsymbol{\theta}_D, \boldsymbol{\theta}_G) = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}}\left[\log\left(1 - D\left(G(\boldsymbol{z})\right)\right)\right]$$

$$= -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{model}}[\log(1 - D(\boldsymbol{x}))]$$

Generator & Discriminator: Implementation

- The discriminator is just a standard neural network
- The generator looks like an inverse discriminator

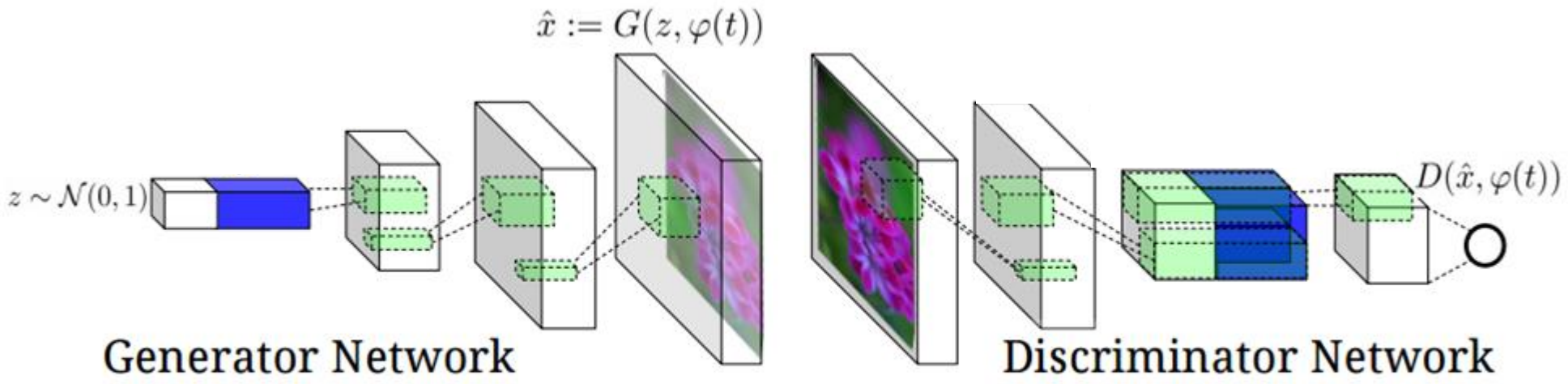
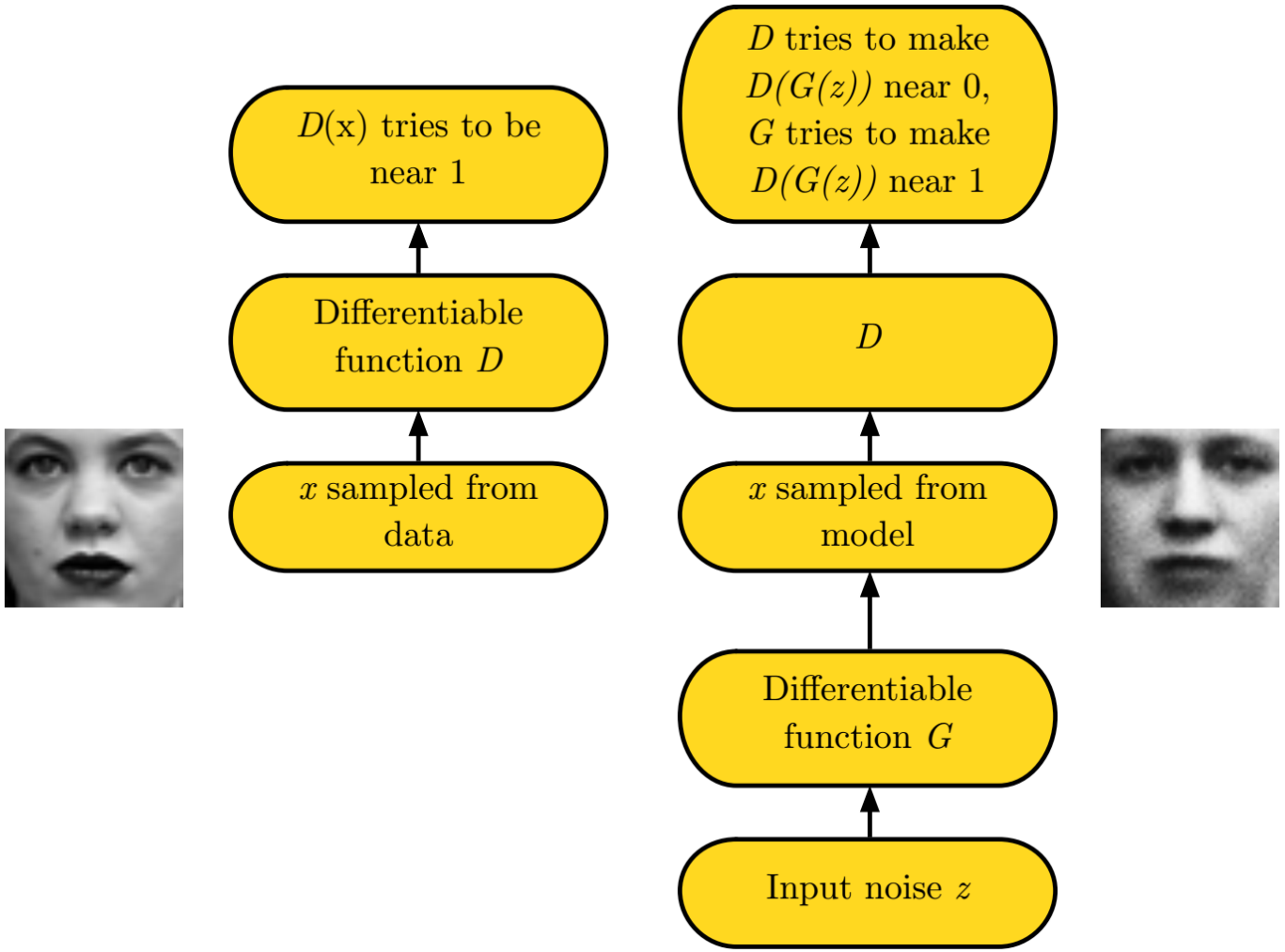


Figure 2. Our text-conditional convolutional GAN architecture. Text encoding $\varphi(t)$ is used by both generator and discriminator. It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

Network Architecture

GAN: Pipeline

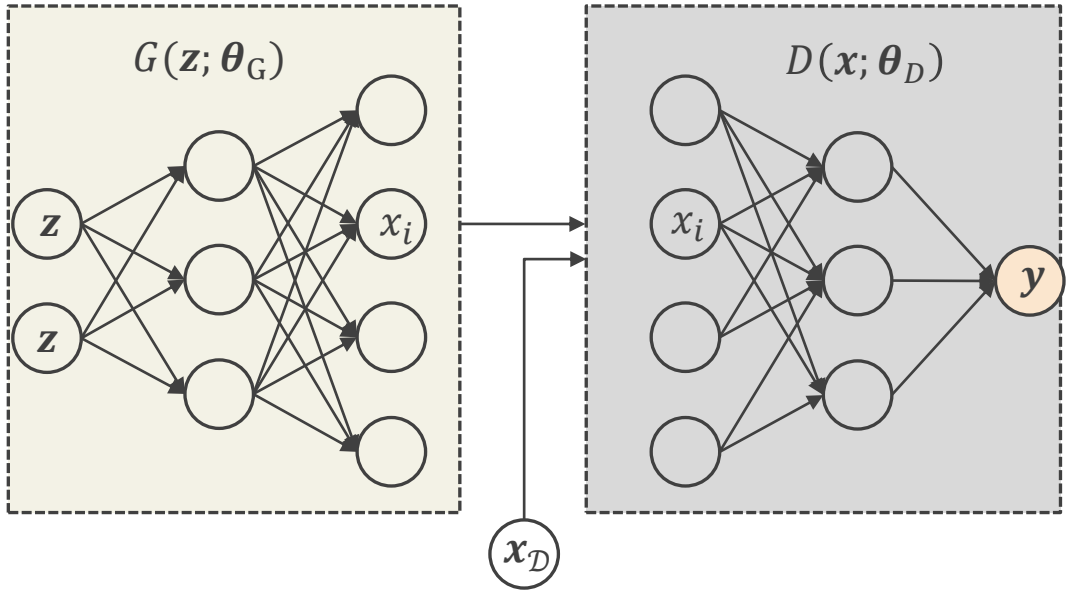


NIPS 2016 Tutorial: Generative Adversarial Networks

Learning objectives

- Not obvious how to use maximum likelihood
- If we take the output of the generator, how to train the discriminator?
- Even then, how do we know if a completely new $\boldsymbol{x}$ is likely or not?
  - Remember, we have no encoder → no target to compare against

$\boldsymbol{z} \sim \mathcal{N}(0,1)$ or
$\boldsymbol{z} \sim \text{Uniform}(0,1)$
…



Minimax Game

- For the simple case of zero-sum game

$$J_G = -J_D$$

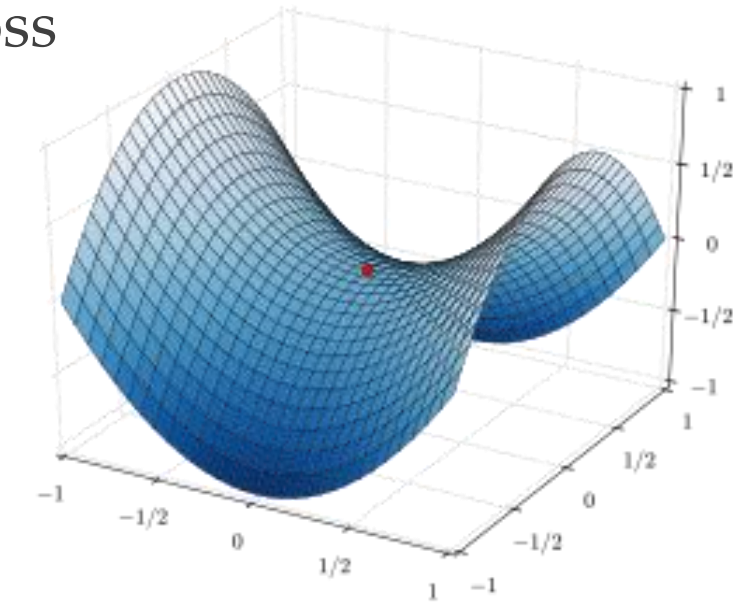
  - The lower the generator loss, the higher the discriminator loss
  - Symmetric definitions
- Our learning objective then becomes

$$V = -J_D(\boldsymbol{\theta}_\mathrm{D}, \boldsymbol{\theta}_\mathrm{G})$$

- $D(\boldsymbol{x}) = 1 \rightarrow$ The discriminator believes that x is a true image
- $D(G(\boldsymbol{z})) = 1 \rightarrow$ The discriminator believes that $G(z)$ is a true image

Minimax Game

- Learning stops after a while
  - As training iterations increase the discriminator improves: $\frac{dJ_D}{d\boldsymbol{\theta}_\mathrm{D}} \rightarrow 0$
  - Then, the generator, preceding the discriminator, vanish
- Equilibrium is a saddle point of the discriminator loss
- Final loss resembles Jenssen-Shannon divergence
- This allows for easier theoretical analysis



NIPS 2016 Tutorial: Generative Adversarial Networks

Heuristic non-saturating game

- Discriminator loss

$$J_D = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log D(\boldsymbol{x}) \; -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_z} \log(1 - D(G(\boldsymbol{z})))$$

- Generator loss

$$J_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z} \log(D(G(\boldsymbol{z}))$$

- Equilibrium not any more describable by single loss
  - The discriminator maximizes the log-likelihood of the discriminator correctly discovering real $\log D(\boldsymbol{x})$ and fake $\log(1 - D(G(\boldsymbol{z})))$ samples
  - The generator $G(\boldsymbol{z})$ maximizes the log-likelihood of the discriminator $\log(D(G(\boldsymbol{z}))$ being wrong. Doesn't case if D gets confused with real samples
- Heuristically motivated; generator can still learn even when discriminator successfully rejects all generator samples

Modifying GANs for max-likelihood

- Discriminator loss

$$J_D = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{data}} \log D(\boldsymbol{x}) \; -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}} \log(1 - D(G(\boldsymbol{z})))$$

- Generator loss

$$J_G = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log(\sigma^{-1}(D\big(G(\boldsymbol{z})\big))$$

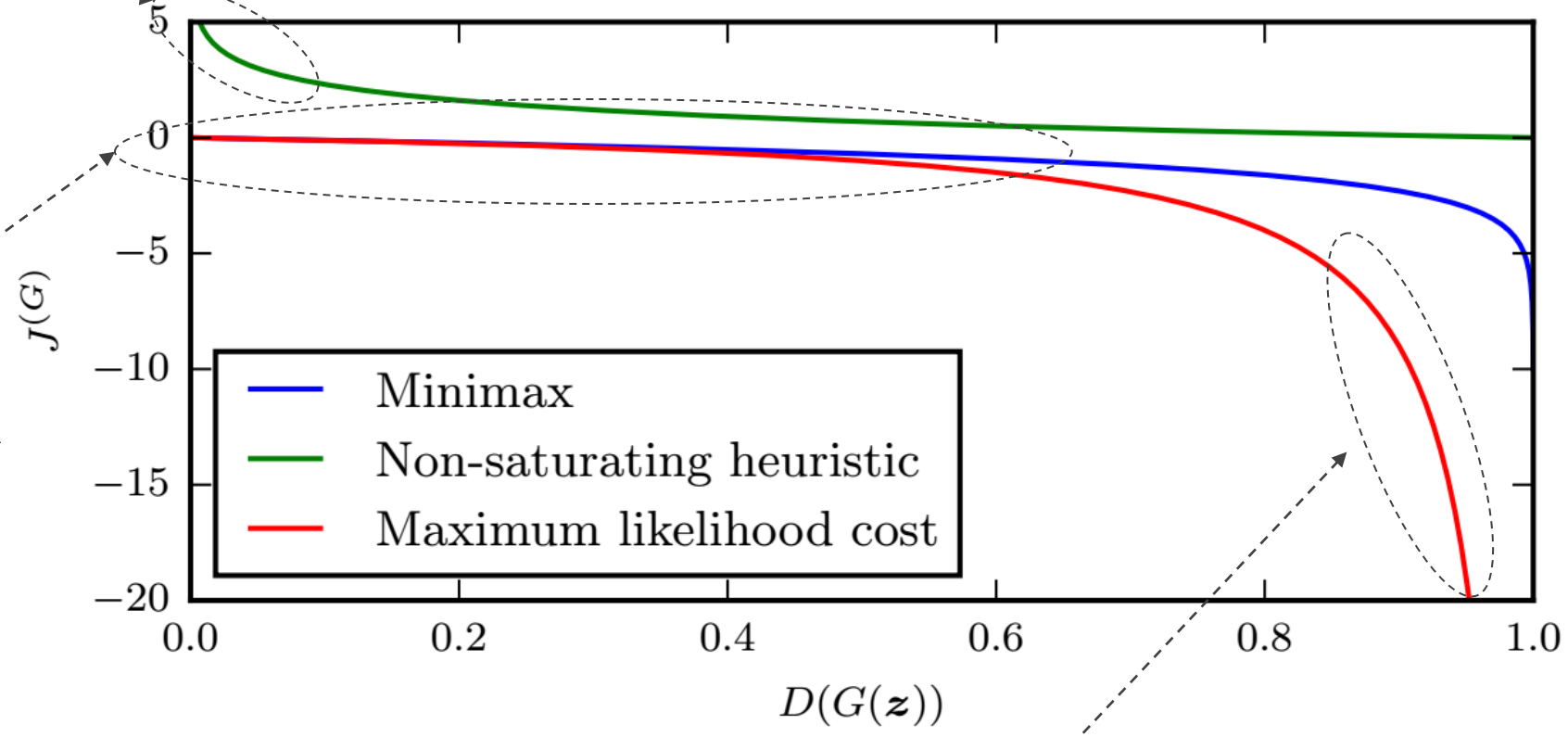
- The discriminator remains the same
- The generator is activated by an inverse sigmoid
  - When discriminator is optimal $\frac{dJ_D}{d\boldsymbol{\theta}_D} \rightarrow 0$
  - the generator gradient matches that of maximum likelihood

On distinguishability criteria for estimating generative models

Comparison of Generator Losses

The heuristic loss yields good generator gradients when the discriminator is too good. And smaller gradients as the discriminator gets more confused.

$$J_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z}\log(D(G(\boldsymbol{z}))$$



When the discriminator detects fake samples accurately (low $D(G(\boldsymbol{z}))$) the generator has a flat loss curve with both the minimax and the ML losses
→ no gradients in early steps

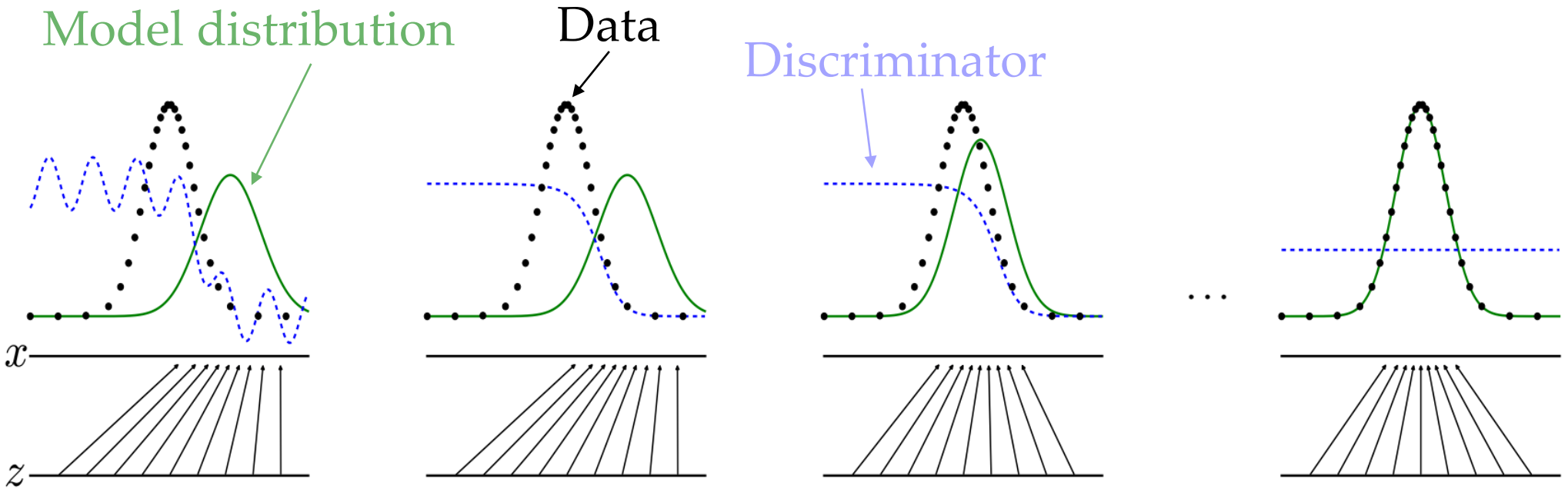
The ML cost variant generates gradients mostly from the "good generations"
→ all gradients from few samples
→ high variance, unreliable gradients (?)

Optimal discriminator

- Optimal $D(\boldsymbol{x})$ for any $p_{data}(\boldsymbol{x})$ and $p_{model}(\boldsymbol{x})$ is always

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

- Estimating this ratio with supervised learning (discriminator) is the key



Why is this the optimal discriminator?

- $L(D,G) = \int_{\boldsymbol{x}} p_r(\boldsymbol{x}) \log D(\boldsymbol{x}) + p_g(\boldsymbol{x}) \log\big(1 - D(\boldsymbol{x})\big)\, d\boldsymbol{x}$
  - Minimize $\mathcal{L}(D,G)$ w.r.t. $D \rightarrow \frac{d\mathcal{L}}{dD} = 0$ and ignore the integral (sample over x)
  - The function $x \rightarrow a \log x + b \log(1-x)$ attains max in $[0,1]$ at $\frac{a}{a+b}$
- The optimal discriminator

$$D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

  - And at **optimality** $p_g(\boldsymbol{x}) \rightarrow p_r(\boldsymbol{x})$, thus

$$D^*(\boldsymbol{x}) = \frac{1}{2}$$

$$L(G^*, D^*) = -2 \log 2$$

Great blog: https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

GANs and Jensen-Shannon divergence

- Expanding the JS divergence for the optimal discriminator $D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})+p_g(\boldsymbol{x})}$

$$D_{JS}(p_r||p_g) = \frac{1}{2}D_{KL}(p_r||\frac{p_r+p_g}{2}) + \frac{1}{2}D_{KL}(p_g||\frac{p_r+p_g}{2})$$

$$= \frac{1}{2}\left(\log 2 + \int_x p_r(x)\log\frac{p_r(x)}{p_r(x)+p_g(x)}\,dx + \log 2 + \int_x p_g(x)\log\frac{p_g(x)}{p_r(x)+p_g(x)}\,dx\right)$$

$$= \frac{1}{2}(\log 4 + L(G,D^*))$$

- So, $L(G,D^*) = 2D_{JS}(p_r \parallel p_g) - 2\log 2$
  - And we just found out that $L(G^*,D^*) = -2\log 2 \Rightarrow D_{JS}(p_r||p_g) = 0$

https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

Is the divergence important?

- Does the divergence make a difference?
- Is there a difference between KL-divergence, Jensen-Shannon divergence, …

$$D_{KL}(p_r||p_g) = \int_x p_r \log \frac{p_r}{p_g} dx$$

$$D_{JS}(p_r||p_g) = \frac{1}{2} D_{KL}(p_r|| \frac{p_r + p_g}{2}) + \frac{1}{2} D_{KL}(p_g|| \frac{p_r + p_g}{2})$$

KL vs JS

- JS is symmetric
- KL is not

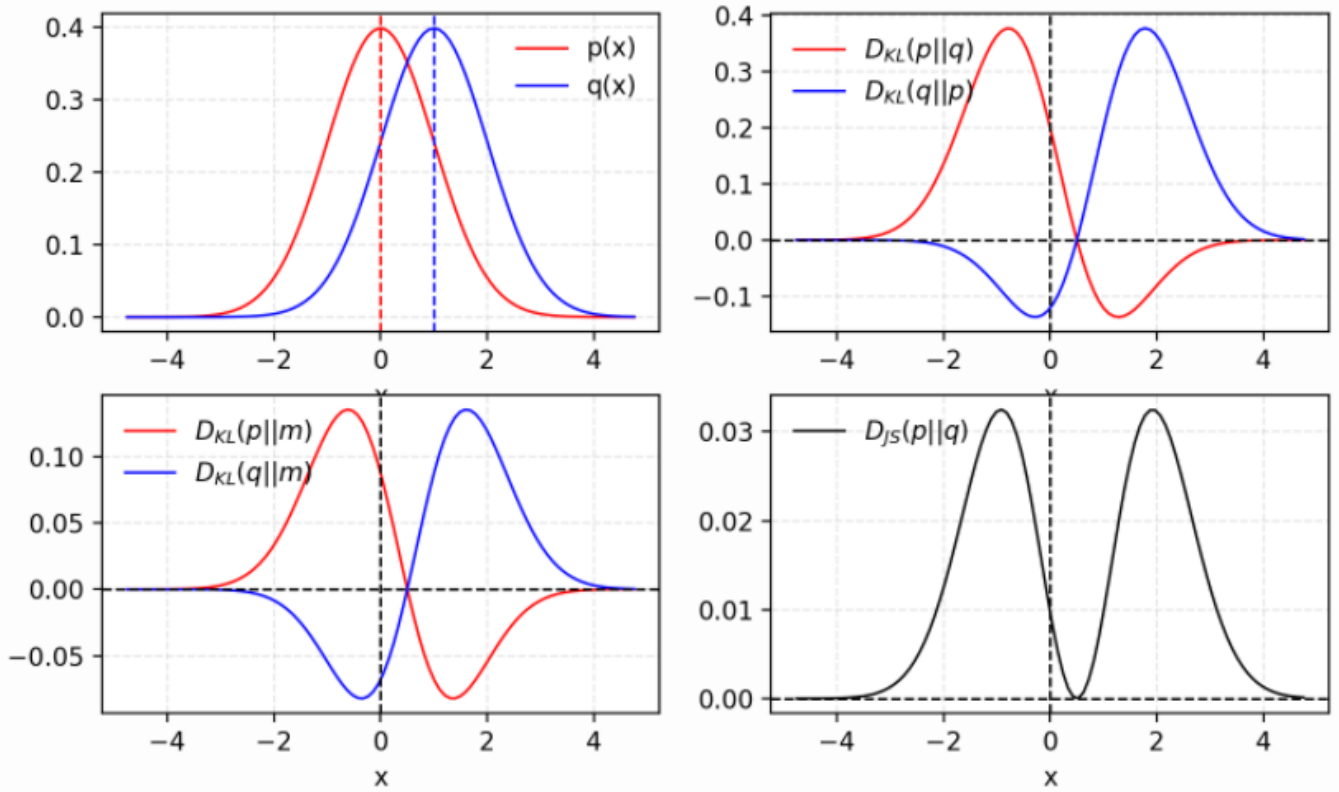
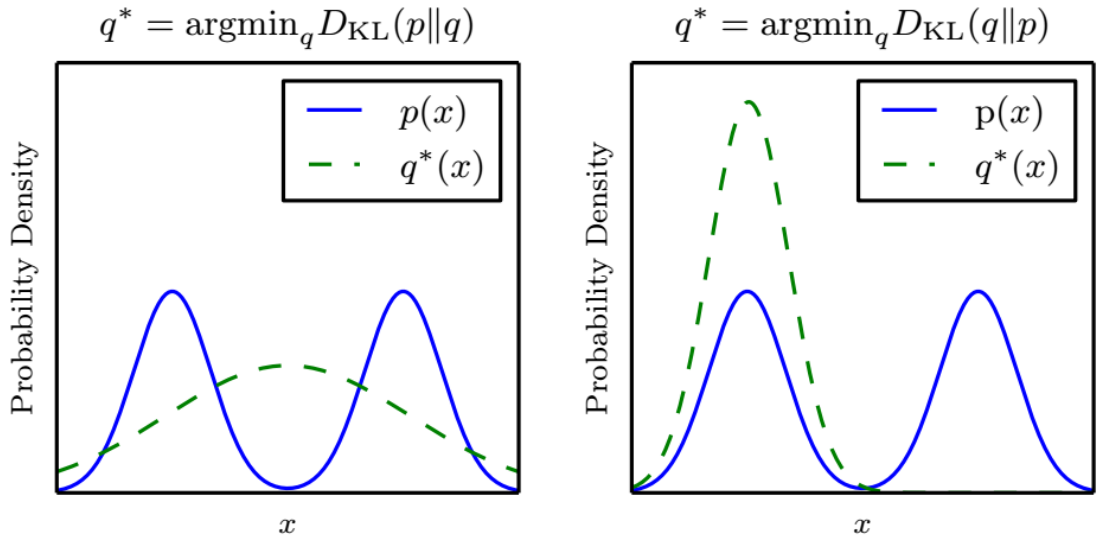


Fig. 1. Given two Gaussian distribution, p with mean=0 and std=1 and q with mean=1 and std=1. The average of two distributions is labelled as $m = (p + q)/2$. KL divergence D_{KL} is asymmetric but JS divergence D_{JS} is symmetric.

https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html

Is the divergence important?

- $D_{KL}(p(x)||q^*(x))$ → high probability everywhere that the data occurs
- $D_{KL}(q^*(x)||p(x))$ → low probability wherever the data does not occur
- Backward KL is 'zero forcing' the learned model → makes model "conservative"
  - $D_{KL}(q^*(x)||p(x)) = \int q^*(x) \log \frac{q^*(x)}{p(x)}$
  - $q^*(x) \to 0$ where $\frac{q^*(x)}{p(x)}$ cannot be good
  - Avoid areas where $p(x) \to 0$



p_r is what we get from our data and cannot change
p_g is what we learn with our model

General observations

- GANs are designed to deliver good generations
- With the original GANs we have no encoder
  - No latent variable model
  - No representation for a given image x
- With GANs we do not write down the densities explicitly
  - We can do generations
  - But cannot easily do inferences, compute conditionals or marginals

Training procedure

- Use SGD-like algorithm of choice
  - Adam Optimizer is a good choice
- Use two mini-batches simultaneously
  - The first mini-batch contains real examples from the training set
  - The second mini-batch contains fake generated examples from the generator
- Optional: run k-steps of one player (e.g. discriminator) for every step of the other player (e.g. generator)

Great blog: https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html