

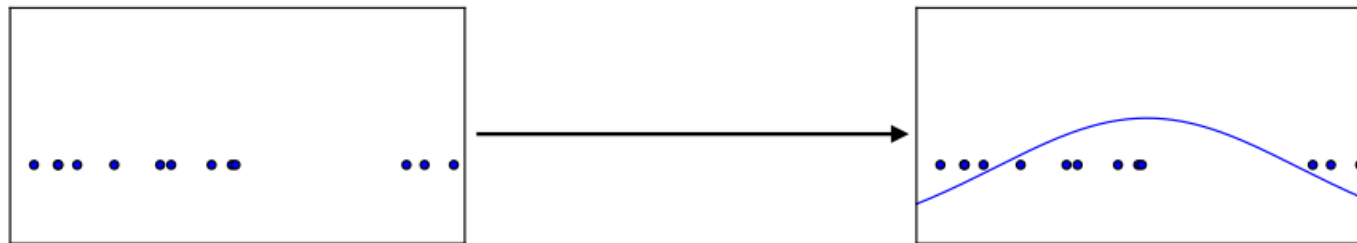
Chapter 20

Deep Generative Models

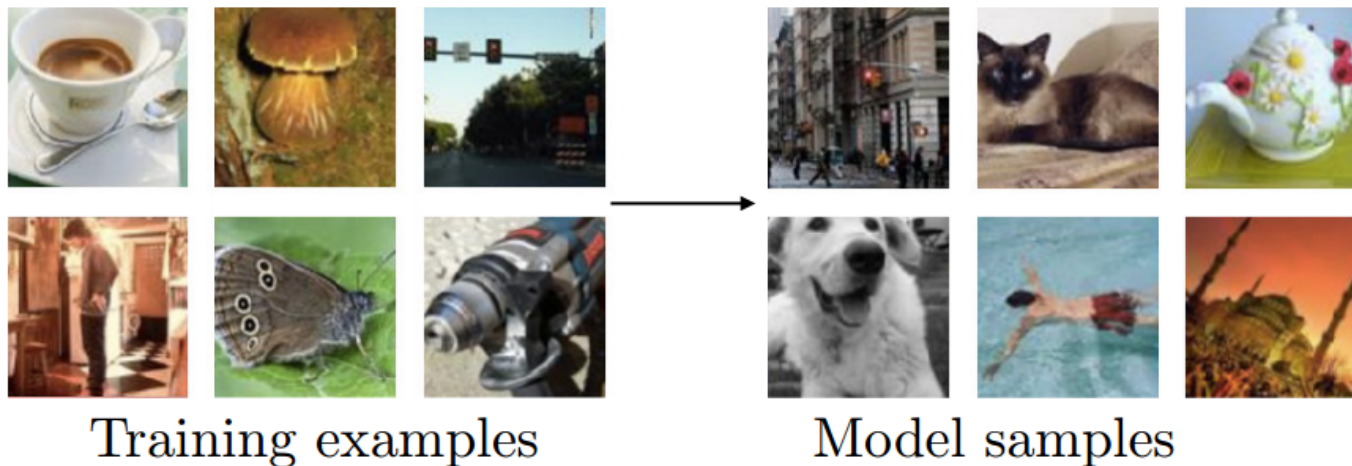
Generative Models

Models that are able to

- Provide an estimate of the probability distribution function, p_{data} , or



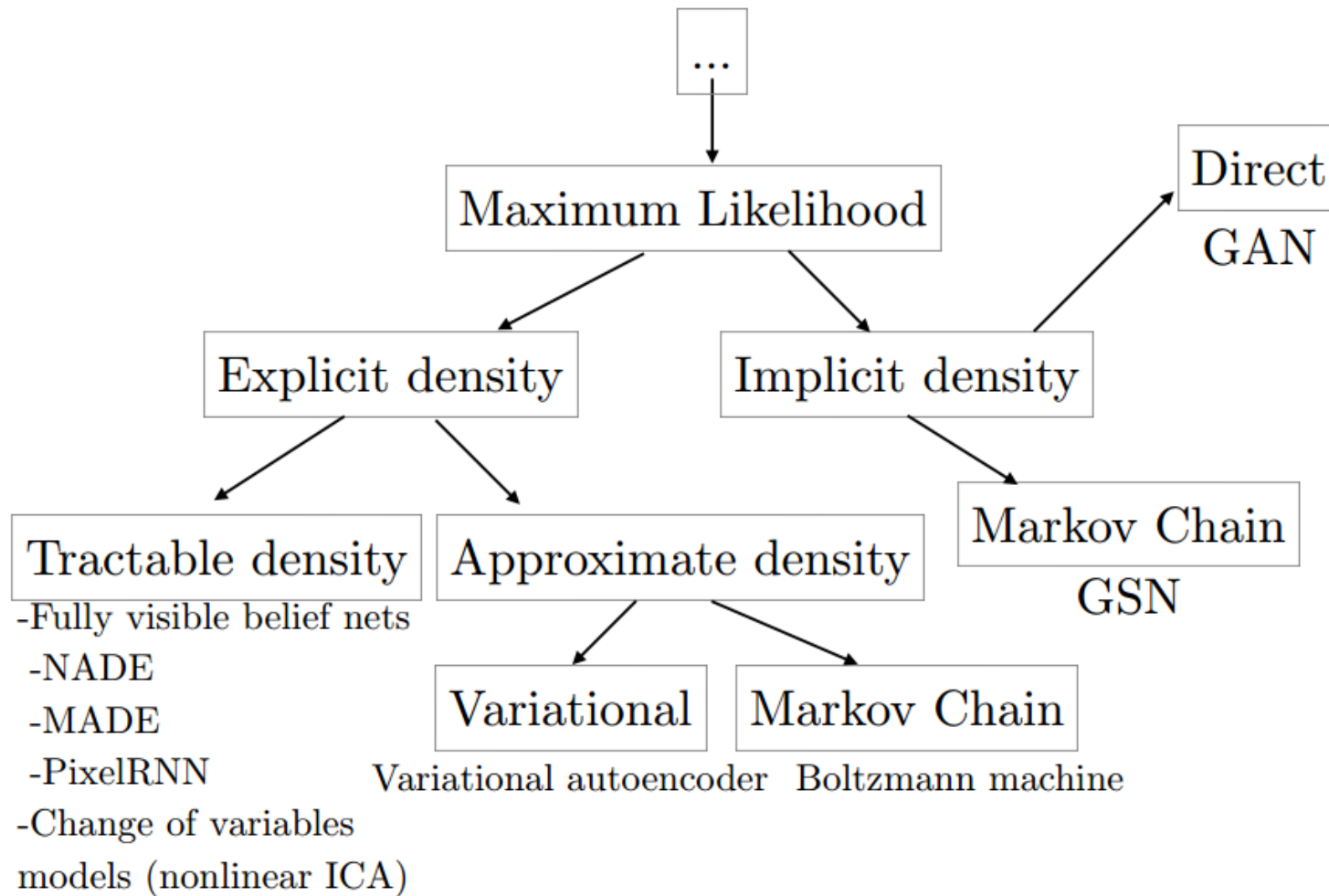
- Generate samples from a (likely implicit) distribution



Why Study Generative Models?

- Manipulation of high-dimensional, multi-modal distributions
- Potential uses in reinforcement learning, such as future state prediction
- Training with missing data (e.g. missing labels) and prediction on them
- Generation of realistic samples
- etc.

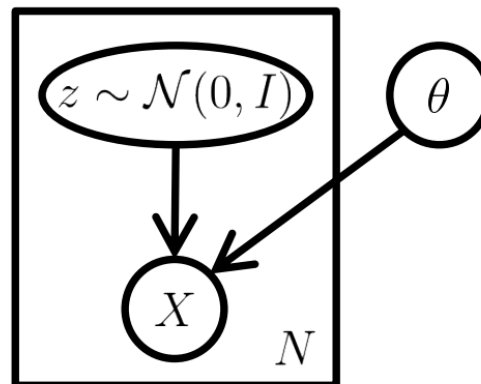
Taxonomy of Generative Models



- Explicit density, $p_{\text{model}}(\mathbf{x}; \boldsymbol{\theta})$
 - Tractable (trained with the ordinary ML)
 - Intractable/approximate (trained with approximate inference and/or MCMC approximations)
- Implicit density
 - Single-step sample generation via a network
 - Multi-step sample generation via Markov chains

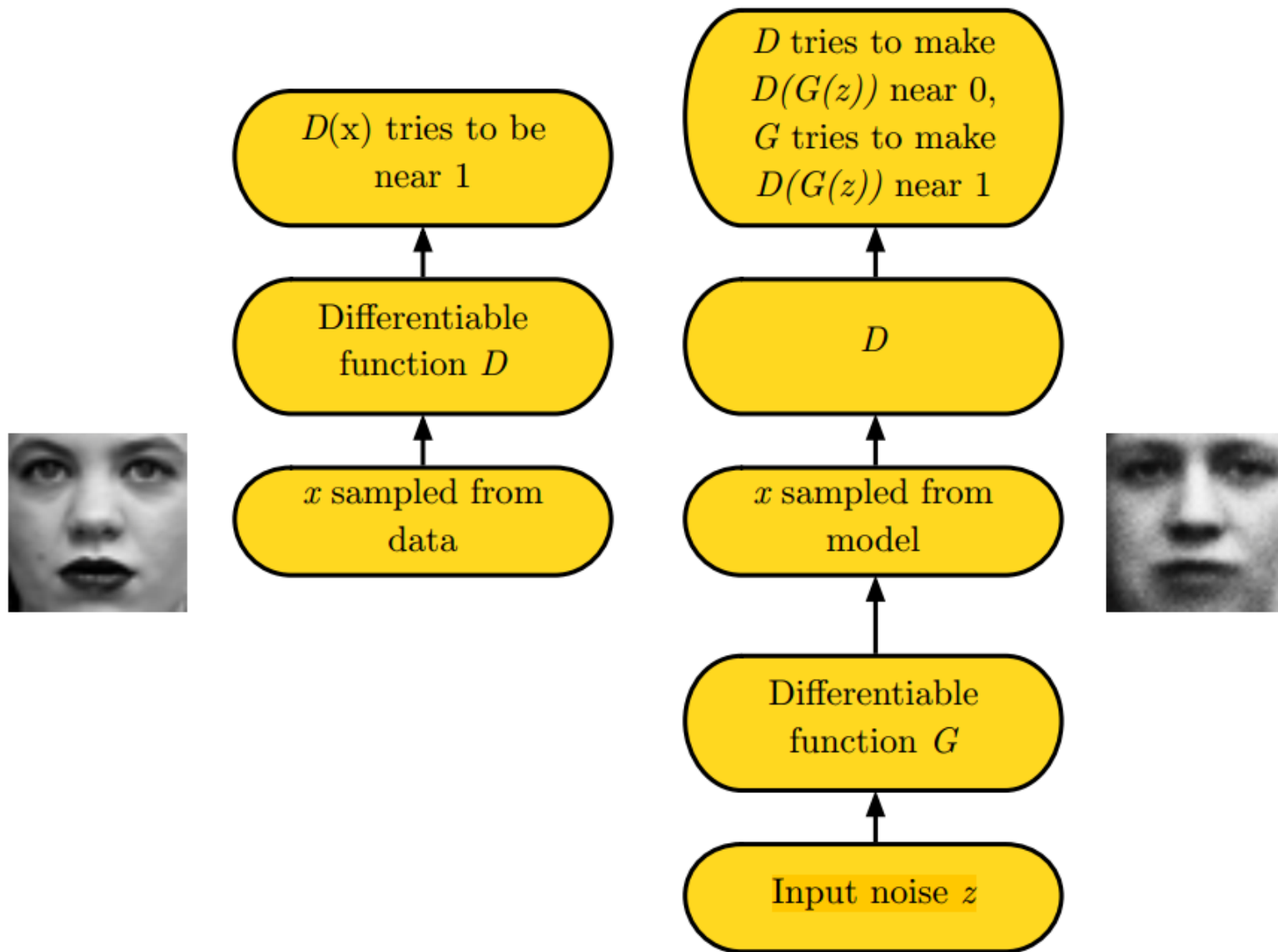
Generative Adversarial Networks (GAN)

- A differentiable **generation** network G , paired with a **discriminator** D for training
- Generator G maps latent noises $z \sim p(z)$ to visible variables x
 - Conceptually, a graphical model with the same structure as VAE
 - $x = G(z)$ can be regarded as a sample drawn from some $p_g(x)$



- **Generator is what we are concerned with**

- Discriminator D divides inputs into real and fake classes
 - An ordinary binary classifier trained supervisedly
 - Inputs are training examples (real) and generated samples (fake)



Training GANs: Two-Player Minimax Game

- $D(\mathbf{x}; \boldsymbol{\theta}^{(D)}), G(\mathbf{z}; \boldsymbol{\theta}^{(G)})$ can be implemented with neural networks, and each has their own cost to minimize

- **Discriminator cost** (cross-entropy cost)

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -E_{\mathbf{x} \sim p_{\text{data}}} \log D(\mathbf{x}) - E_{\mathbf{z} \sim p_{\mathbf{z}}} \log(1 - D(G(\mathbf{z})))$$

where $D(\mathbf{x})$ denotes the probability of \mathbf{x} being real

- **Generator cost**

$$J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

- Note that the sum of all players' costs is zero (zero-sum game)

- The entire game can be summarized with a value function

$$V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) \equiv -J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

and the objective is to find a generator

$$\boldsymbol{\theta}^{(G)*} = \arg \min_{\boldsymbol{\theta}^{(G)}} \max_{\boldsymbol{\theta}^{(D)}} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

Optimization vs. Game

- The solution to an optimization problem is generally a **local minimum** of an objective function in parameter space, e.g.

$$\arg \min_{\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)}} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

where both $\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)}$ are optimized simultaneously

- The solution to a game problem is generally a **saddle point** of an objective function in parameter space, e.g.

$$\arg \min_{\boldsymbol{\theta}^{(G)}} \max_{\boldsymbol{\theta}^{(D)}} V(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$$

where $\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)}$ are optimized in turn by controlling one of them at a time with the other fixed

The Optimal Discriminator

- For a given generator G , the optimal discriminator is seen to be

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})}$$

which can be obtained by having

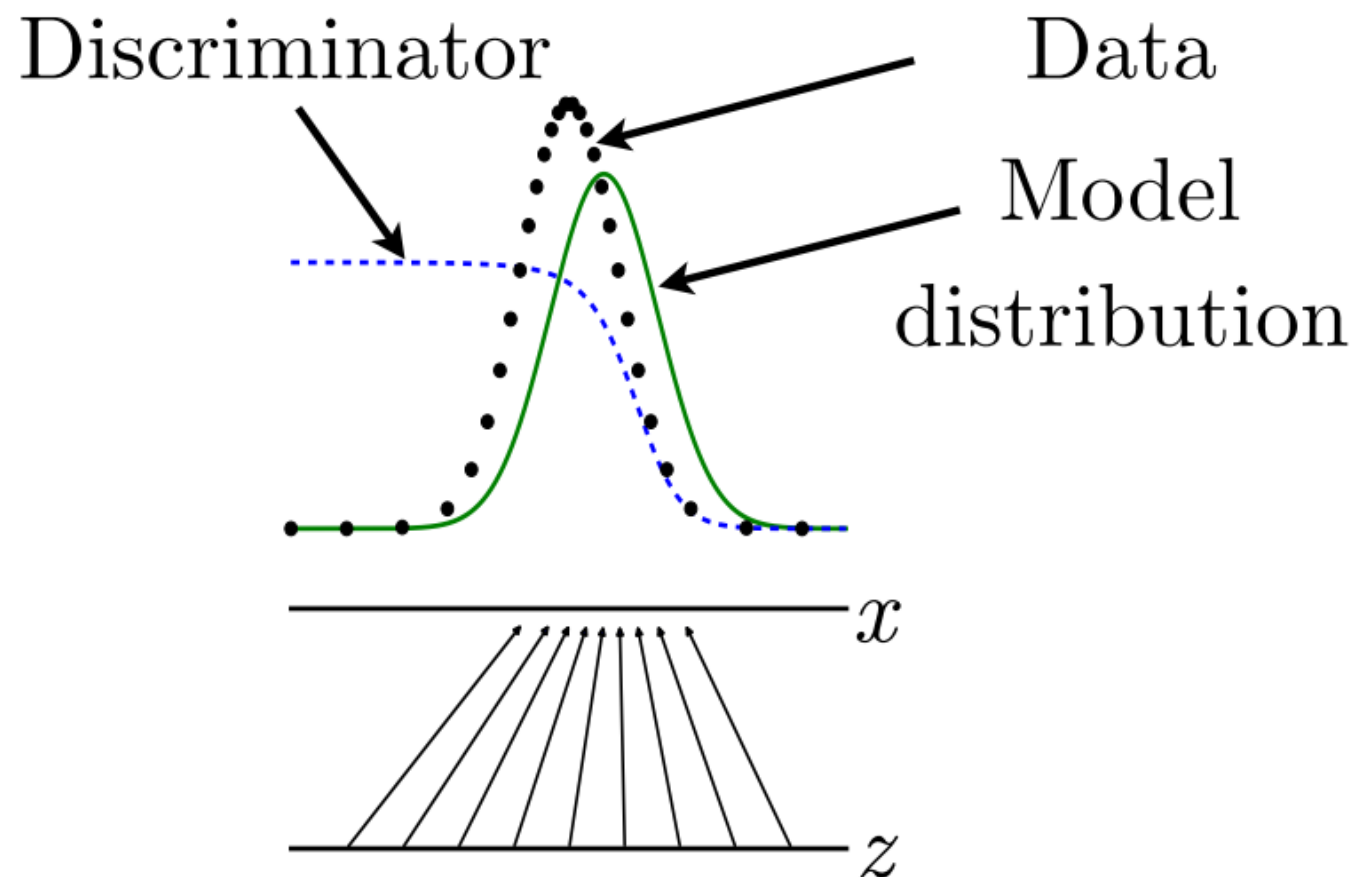
$$\frac{\delta}{\delta D(\mathbf{x})} J^{(D)}(\mathbf{x}) = 0$$

- When given enough capacity, the discriminator obtains an estimate

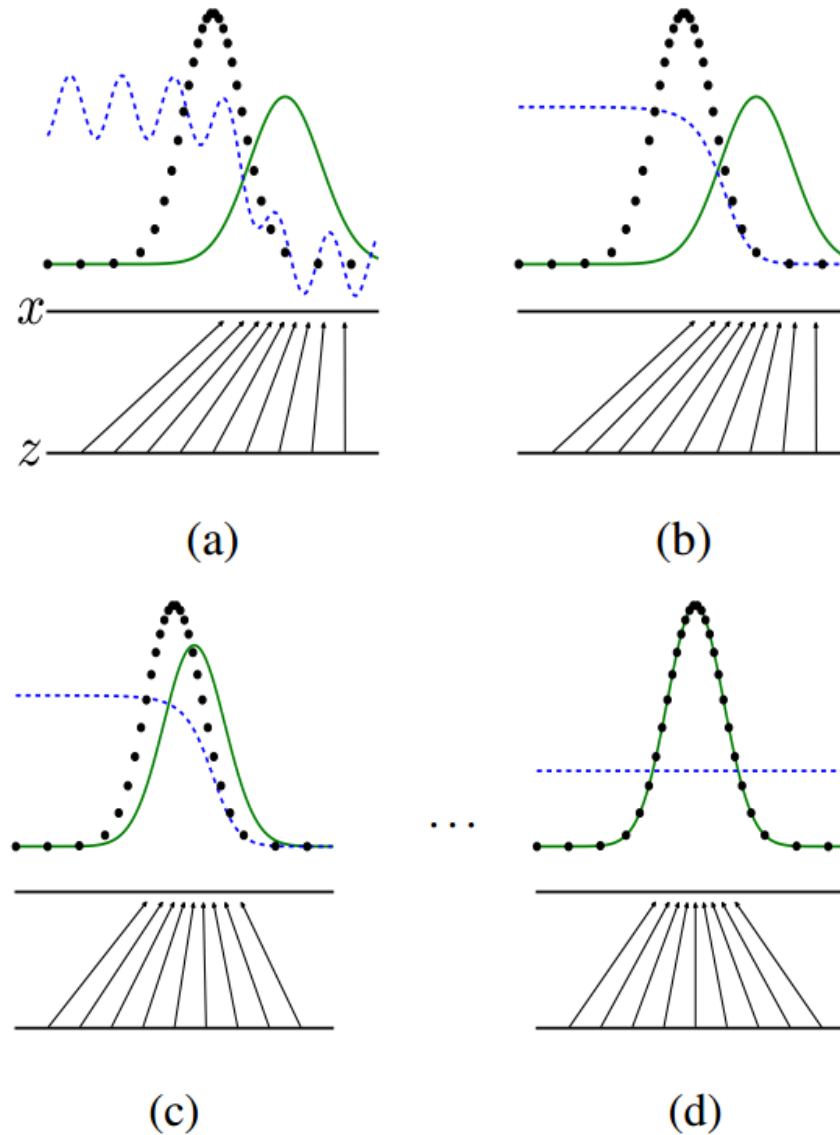
$$\frac{p_{\text{data}}(\mathbf{x})}{p_g(\mathbf{x})}$$

at every \mathbf{x}

- This is the key that sets GANs apart from other generative models



- The generator is to learn a model by following a discriminator uphill



The Optimal Generator

- Given $D_G^*(\mathbf{x})$ and enough capacity, the optimal generator is to minimize the Jensen-Shannon divergence between p_{data} and p_g

$$\begin{aligned}
 & \arg \min_{p_g} E_{\mathbf{x} \sim p_{\text{data}}} \log D_G^*(\mathbf{x}) + E_{\mathbf{x} \sim p_g} \log(1 - D_G^*(\mathbf{x})) \\
 &= \arg \min_{p_g} E_{\mathbf{x} \sim p_{\text{data}}} \log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} + E_{\mathbf{x} \sim p_g} \log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \\
 &= \arg \min_{p_g} -\log(4) + \text{KL} \left(p_{\text{data}} \parallel \frac{p_{\text{data}} + p_g}{2} \right) + \text{KL} \left(p_g \parallel \frac{p_{\text{data}} + p_g}{2} \right) \\
 &= \arg \min_{p_g} -\log(4) + 2 \times \text{JSD}(p_{\text{data}} \parallel p_g)
 \end{aligned}$$

- The minimum is achieved when $p_g = p_{\text{data}}$, i.e. $\text{JSD}(p_{\text{data}} \parallel p_g) = 0$

- Remarks

- The optimization is done w.r.t. p_g directly
- The analysis for the discriminator is done w.r.t. $D(\mathbf{x})$
- Enough capacity in both contexts means that $D_G^*(\mathbf{x})$ and $p_g^*(\mathbf{x})$ can be implemented by $D(\mathbf{x}; \boldsymbol{\theta}^{(D)*})$ and $G(\mathbf{z}; \boldsymbol{\theta}^{(G)*})$, respectively

Implementation

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the generator by descending its stochastic gradient:

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

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

- (Convergence) If G and D have enough capacity, and at each step of Algorithm I, the discriminator is allowed to reach its optimum $D_G^*(x)$ given G , and p_g is updated to improve the criterion (reduce the cost)

$$E_{\mathbf{x} \sim p_{\text{data}}} \log D_G^*(\mathbf{x}) + E_{\mathbf{x} \sim p_g} \log(1 - D_G^*(\mathbf{x}))$$

then p_g converges to p_{data}

- Nothing is said about the convergence when optimization is done based on simultaneous stochastic gradient descent in parameter space

Non-Convergence of Gradient Descent

- Toy problem

$$\min_x \max_y V(x, y) = xy$$

- x, y are optimized based on gradient descent with a tiny learning rate

$$x(t + \Delta t) = x(t) - \Delta t \frac{\partial}{\partial x(t)} V(x(t), y(t))$$

$$y(t + \Delta t) = y(t) + \Delta t \frac{\partial}{\partial y(t)} V(x(t), y(t))$$

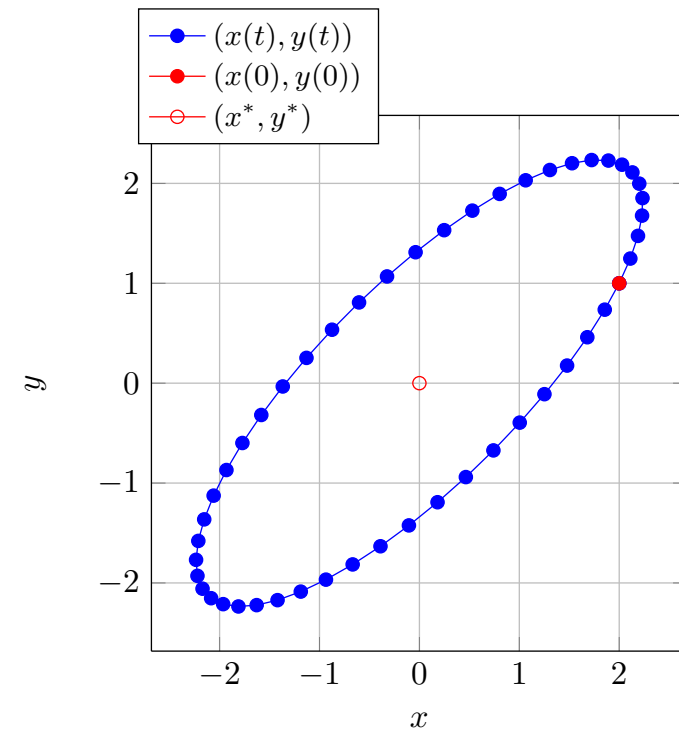
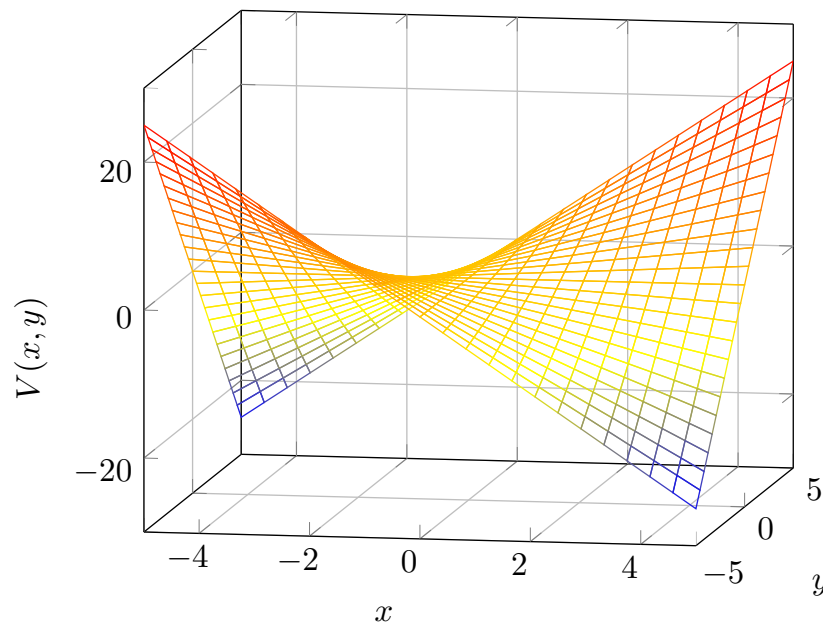
- This amounts to solving

$$\begin{cases} x'(t) = -y(t) \\ y'(t) = x(t) \end{cases} \rightarrow x''(t) = -x(t)$$

which has a solution of the form

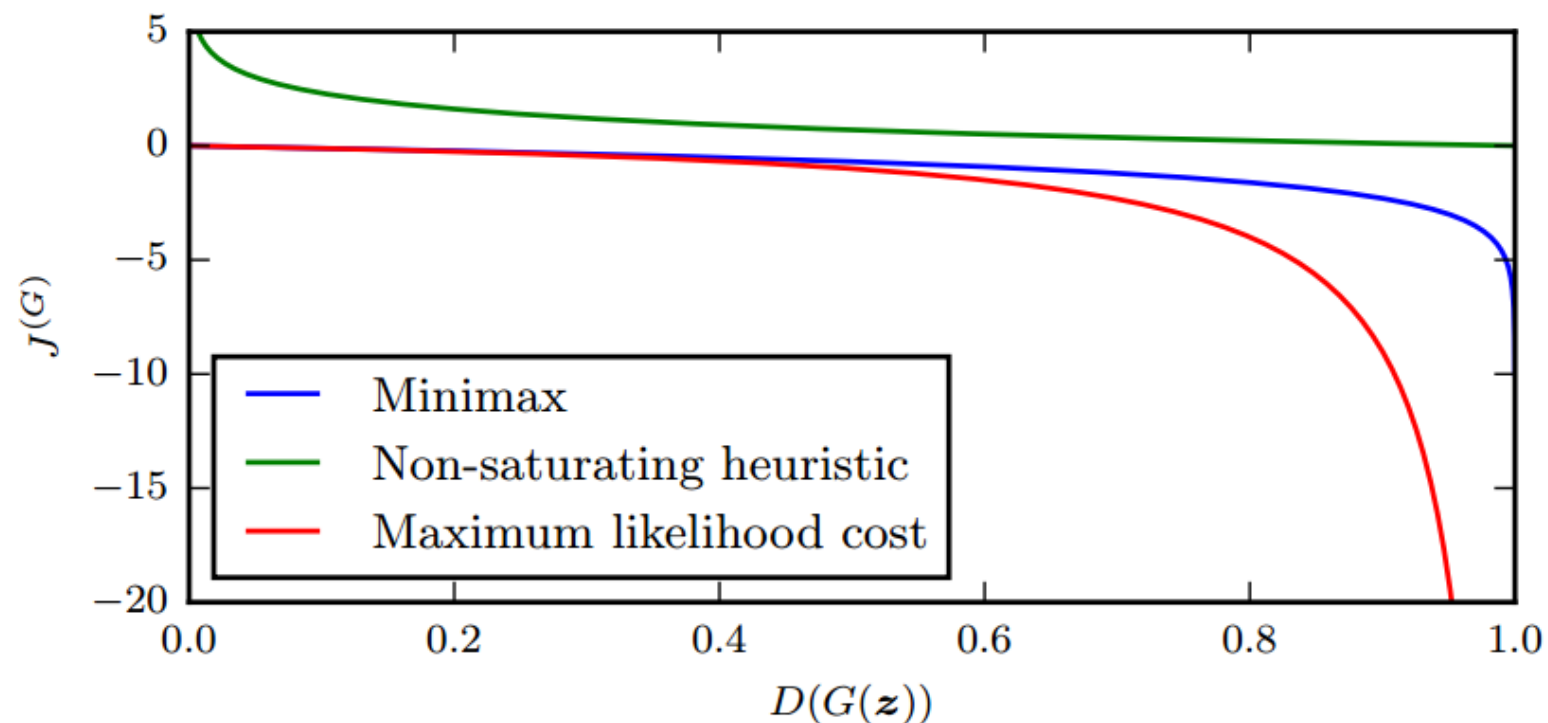
$$x(t) = x(0) \cos(t) + y(0) \sin(t)$$

$$y(t) = x(0) \sin(t) + y(0) \cos(t)$$



Other Games

- Zero-sum game does not perform well in learning generator: gradients of $J^{(G)}$ w.r.t $D(G(z))$ vanish when the discriminator performs well



- Heuristic, non-saturating game (to ensure non-zero gradients)

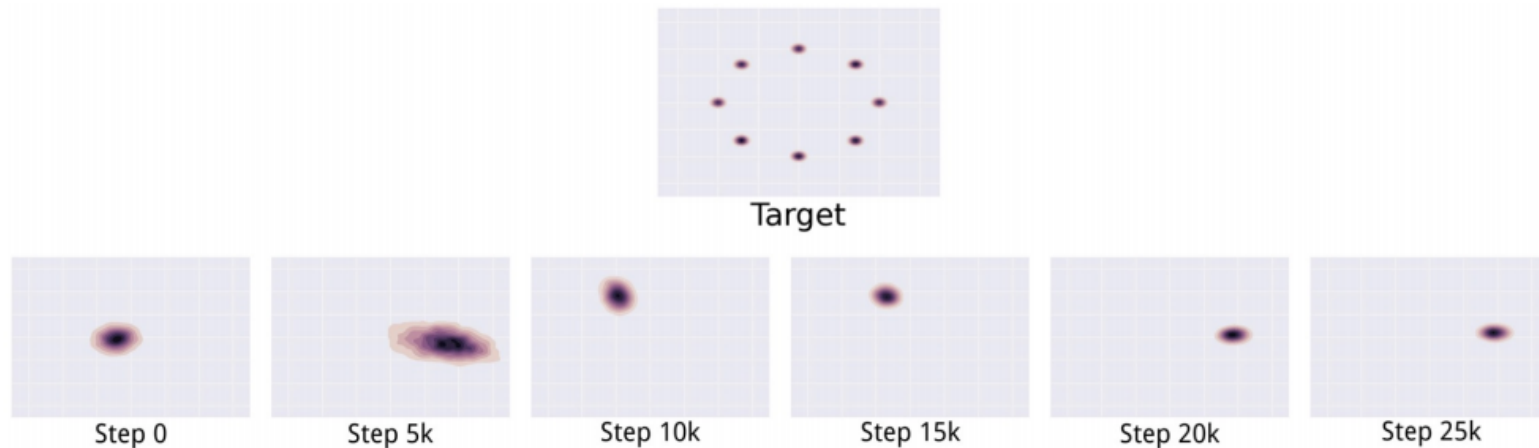
$$J^{(G)} = -E_z \log D(G(z))$$

- Maximum likelihood game (to minimize KL divergence)

$$J^{(G)} = -E_z \exp(\sigma^{-1}(D(G(z))))$$

Mode Collapse Problem

- The generator learns to map different z to the same x



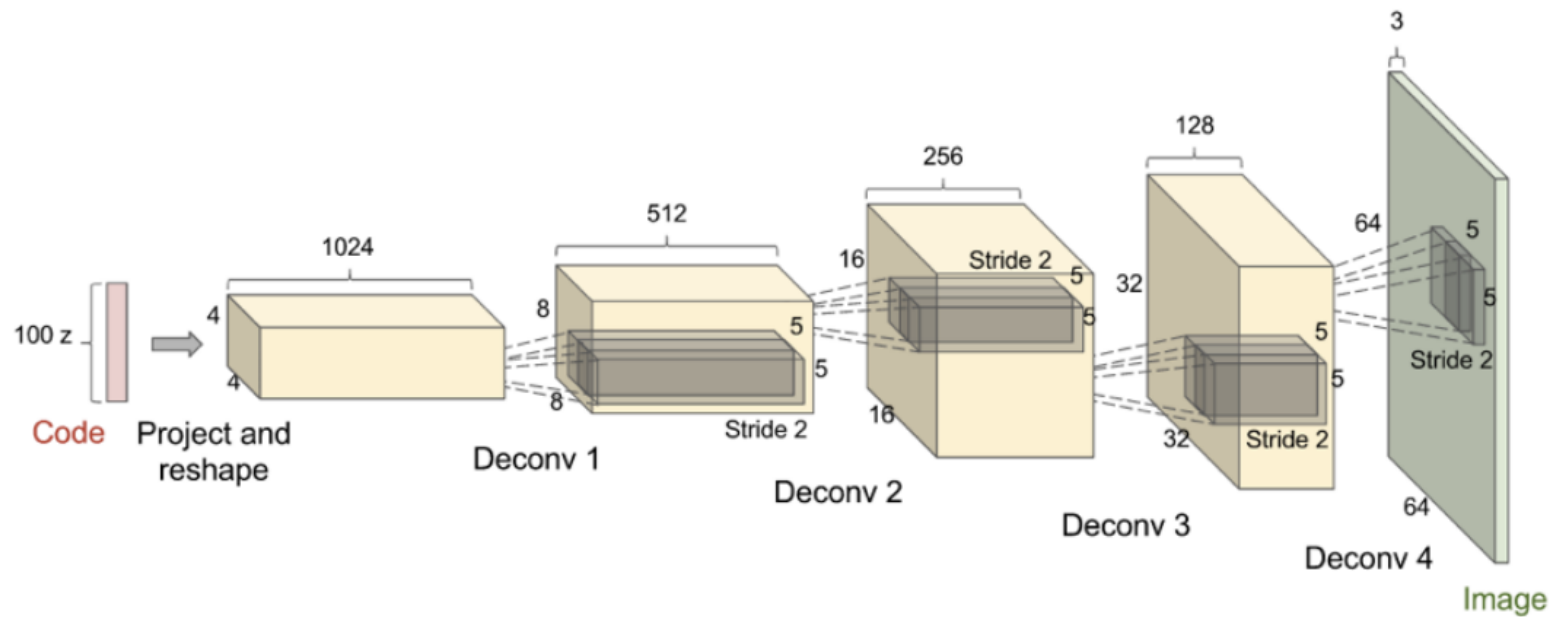
Top: Data distribution (Mixture Gaussian)

Bottom: Learned generator distribution over time

- The generator distribution produces only a single mode at a time and does not converge in this example
- This is acceptable in some applications but not all

DCGAN

- There are many different implementations for generators, such as DCGAN, LPGAN, and more (study by yourself)



Wasserstein GAN (WGAN)

- **Idea:** To adopt the Earth Mover distance (**Wassertein distance**) as the convergence criterion
- We have seen previously that training GAN is to learn a model distribution P_θ that should ideally converge to the data distribution P_r
- The convergence calls for a distance measure $\rho(P_r, P_\theta)$ to indicate how close these two distributions are
- To optimize the parameter θ , it is desirable that $\rho(P_r, P_\theta)$ is a continuous function in θ , or equivalently, the mapping $\theta \mapsto P_\theta$ is continuous (i.e., when $\theta_t \rightarrow \theta$, $P_{\theta_t} \rightarrow P_\theta$)
- The continuity depends on the distance measure

Elementary Distances

- The Kullback-Leibler (KL) divergence

$$KL(P_r || P_\theta) = \int P_r(x) \log \frac{P_r(x)}{P_\theta(x)} P_r(x) dx$$

(undefined when there are x 's where $P_r(x) \neq 0$ and $P_\theta(x) = 0$)

- The Jensen-Shannon (JS) divergence

$$JS(P_r || P_\theta) = KL(P_r || P_m) + KL(P_\theta || P_m)$$

where

$$P_m = (P_r + P_\theta)/2$$

- The Earth Mover distance

$$\begin{aligned} W(P_r, P_\theta) &= \inf_{\gamma \in \Pi(P_r, P_\theta)} E_{(x,y) \sim \gamma} [\|x - y\|] \\ &= \inf_{\gamma \in \Pi(P_r, P_\theta)} \sum_{x,y} \gamma(x,y) \|x - y\| \end{aligned}$$

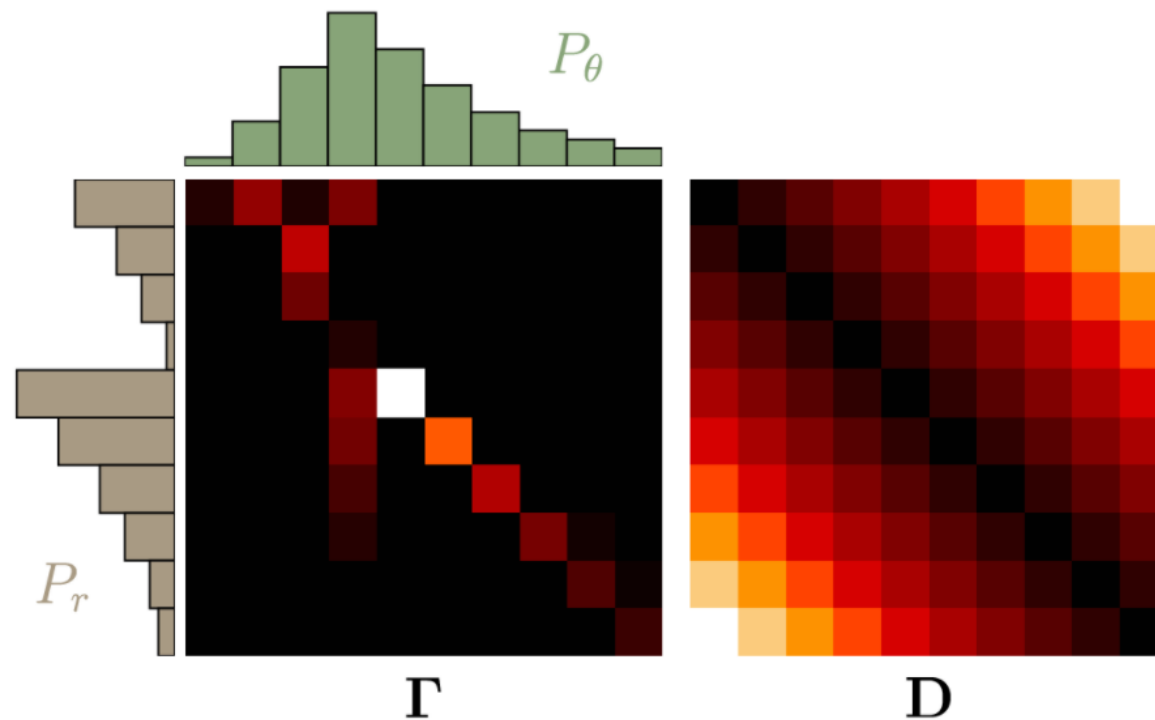
where $\Pi(P_r, P_\theta)$ is the set of all joint distributions $\gamma(x, y)$ whose marginals are P_r and P_θ , respectively; that is,

$$P_r(x) = \sum_y \gamma(x, y)$$

$$P_\theta(y) = \sum_x \gamma(x, y)$$

- Intuitively, $\gamma(x, y)$ indicates how much mass must be transported from y to x in order to transform the distribution P_θ into P_r
- The EM distance is the cost of the optimal transport plan

The Earth Mover (EM) Distance



$$\Gamma = \gamma(\mathbf{x}, \mathbf{y}), \quad D = \|\mathbf{x} - \mathbf{y}\|$$

$$W(P_r, P_\theta) = \inf_{\gamma \in \Pi(P_r, P_\theta)} \langle \Gamma, D \rangle_F$$

Source: <https://vincentherrmann.github.io/blog/wasserstein/>

Comparison of Elementary Distances

- The EM distance is continuous and differentiable almost everywhere, whereas the JS divergence is not
- As such, the EM distance allows the model to learn a probability distribution over low dimensional manifolds by using gradient descent

Example 1 (Learning parallel lines). Let $Z \sim U[0, 1]$ the uniform distribution on the unit interval. Let \mathbb{P}_0 be the distribution of $(0, Z) \in \mathbb{R}^2$ (a 0 on the x-axis and the random variable Z on the y-axis), uniform on a straight vertical line passing through the origin. Now let $g_\theta(z) = (\theta, z)$ with θ a single real parameter. It is easy to see that in this case,

- $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|,$
- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$
- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0, \\ 0 & \text{if } \theta = 0, \end{cases}$

Linear Programming

- The EM distance between two distributions can be solved by linear programming/optimization

$$\min_{\mathbf{x}} z = \mathbf{c}^T \mathbf{x} \quad \text{s.t.} \quad \begin{cases} \mathbf{A}\mathbf{x} = \mathbf{b} \\ \mathbf{x} \geq \mathbf{0} \end{cases}$$

- The dual problem (<https://vincentherrmann.github.io/blog/wasserstein/>)

$$\max_{\mathbf{y}} \tilde{z} = \mathbf{b}^T \mathbf{y} \quad \text{s.t.} \quad \mathbf{A}^T \mathbf{y} \leq \mathbf{c}$$

- (Weak Duality Theorem) \tilde{z} is a lower bound of z

$$z = \mathbf{c}^T \mathbf{x} \geq \mathbf{y}^T \mathbf{A}\mathbf{x} = \mathbf{y}^T \mathbf{b} = \tilde{z}$$

- (Strong Duality Theorem) When we find an optimal solution to the dual problem, $\tilde{z} = z$ (the EM distance)

- The term $Ax = b$

$$\left[\begin{array}{cccc|cccc|c|cccc}
 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 & \dots & 0 & 0 & \dots & 0 \\
 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 & \dots & \dots & 0 & 0 & \dots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & \dots & 1 & 1 & 1 & 1 \\
 \hline
 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & \dots & \dots & 1 & 0 & \dots & 0 \\
 0 & 1 & \dots & 0 & 0 & 1 & \dots & 0 & \dots & \dots & 0 & 1 & \dots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & \dots & \dots & 0 & 0 & 0 & 1
 \end{array} \right] \begin{bmatrix} \gamma(x_1, y_1) \\ \gamma(x_1, y_2) \\ \vdots \\ \gamma(x_2, y_1) \\ \gamma(x_2, y_2) \\ \vdots \\ \vdots \\ \gamma(x_n, y_1) \\ \gamma(x_n, y_2) \\ \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} P_r(x_1) \\ P_r(x_2) \\ \vdots \\ \vdots \\ \frac{P_r(x_n)}{P_\theta(y_1)} \\ P_\theta(y_2) \\ \vdots \\ \vdots \\ P_\theta(y_n) \end{bmatrix}$$

- The vector \mathbf{c}

$$\mathbf{c} = \begin{bmatrix} \|x_1 - y_1\| \\ \|x_1 - y_2\| \\ \vdots \\ \|x_2 - y_1\| \\ \|x_2 - y_2\| \\ \vdots \\ \|x_n - y_1\| \\ \|x_n - y_2\| \\ \vdots \end{bmatrix}$$

- Now we can compute the EM distance using the dual form

$$\max_{\mathbf{y}} \tilde{z} = \mathbf{b}^T \mathbf{y} \quad \text{s.t.} \quad \mathbf{A}^T \mathbf{y} \leq \mathbf{c}$$

- The objective is to find \mathbf{y} such that $\mathbf{b}^T \mathbf{y}$ is maximized

$$\mathbf{b}^T \mathbf{y} = \left[\begin{array}{cccc|cccc} p_r(x_1) & \cdots & p_r(x_n) & p_\theta(y_1) & \cdots & p_\theta(y_n) \end{array} \right] \cdot \left[\begin{array}{c} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \\ -\frac{f(x_1)}{g(x_1)} \\ g(x_2) \\ \vdots \\ g(x_n) \end{array} \right]$$

where the components of \mathbf{y} have been made functions of x_i

- The constraint $\mathbf{A}^T \mathbf{y} \leq \mathbf{c}$ is given by

$$\left[\begin{array}{cccc|cccc}
 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
 1 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
 \hline
 0 & 1 & \cdots & 0 & 1 & 0 & \cdots & 0 \\
 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
 \hline
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 \hline
 0 & 0 & \cdots & 1 & 1 & 0 & \cdots & 0 \\
 0 & 0 & \cdots & 1 & 0 & 1 & \cdots & 0 \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1
 \end{array} \right] \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \\ g(x_1) \\ g(x_2) \\ \vdots \\ g(x_n) \end{bmatrix} \leq \begin{bmatrix} \|x_1 - y_1\| \\ \|x_1 - y_2\| \\ \vdots \\ \|x_2 - y_1\| \\ \|x_2 - y_2\| \\ \vdots \\ \|x_n - y_1\| \\ \|x_n - y_2\| \\ \vdots \end{bmatrix}$$

- It is seen that

$$f(x_i) + g(x_j) \leq \|x_i - y_j\|, \quad \forall i, j$$

- Because $x_i = y_i$, we further arrive at

$$f(x_i) + g(x_i) \leq 0, \quad \forall i = j$$

$$f(x_i) + g(x_j) \leq \|x_i - x_j\|, \quad \forall i \neq j$$

- For $\mathbf{b}^T \mathbf{y}$ to be maximized, both $f(x_i)$ and $g(x_i)$ need to be as large as possible since the components of \mathbf{b} are all non-negative
- The optimal solution must thus have $f(x_i) + g(x_i) = 0$
- The constraint $\mathbf{A}^T \mathbf{y} \leq \mathbf{c}$ then reduces to requiring

$$f(x_i) - f(x_j) \leq \|x_i - x_j\|$$

$$f(x_j) - f(x_i) \leq \|x_j - x_i\|$$

which suggests f is Lipschitz continuous (with Lipschitz constant 1)

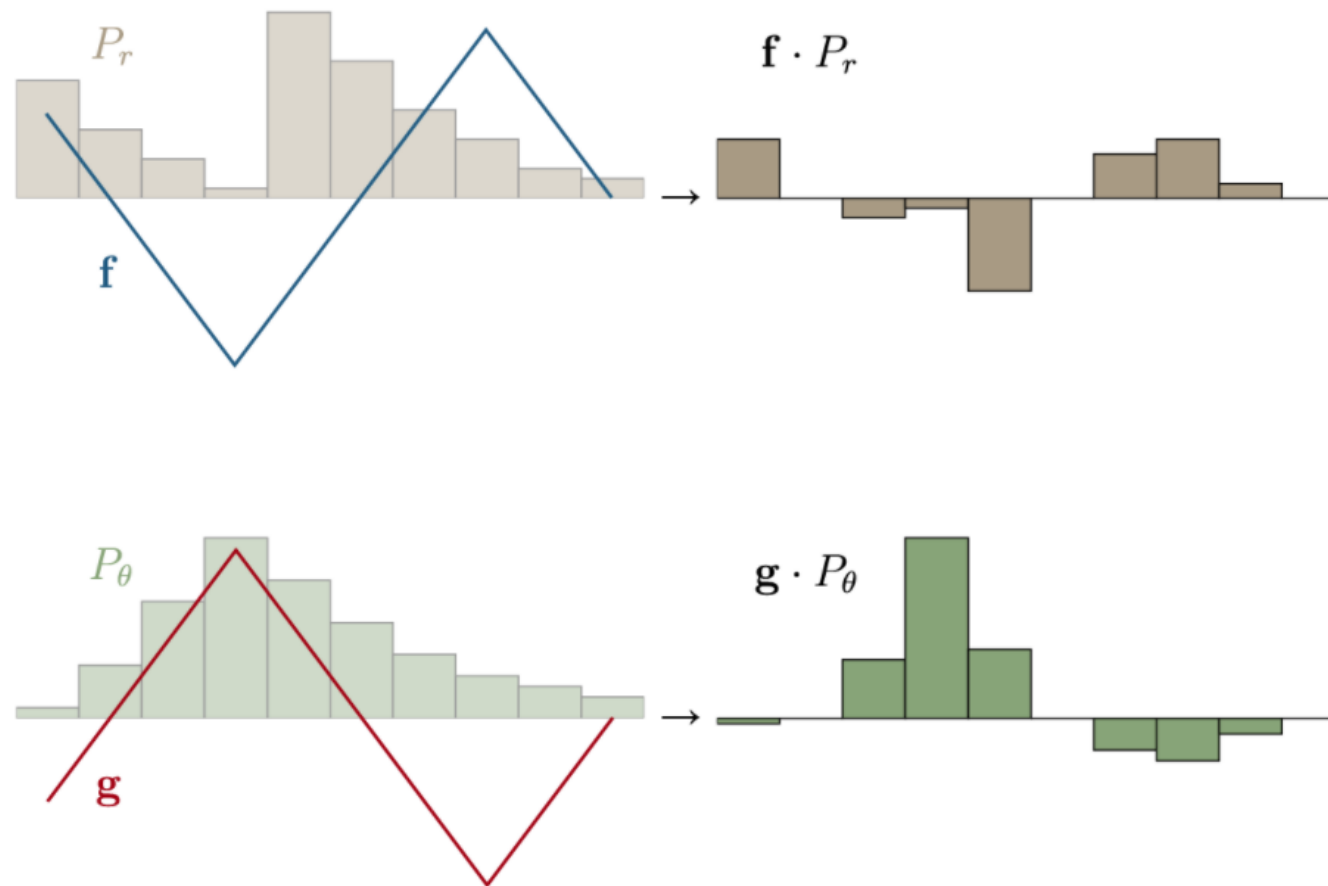
$$\|f(x_i) - f(x_j)\| \leq \|x_i - x_j\|$$

- With this, searching for y to maximize $b^T y$

$$b^T y = \left[\begin{array}{cccc|cccc} p_r(x_1) & \cdots & p_r(x_n) & p_\theta(y_1) & \cdots & p_\theta(y_n) \end{array} \right] \left[\begin{array}{c} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \\ -f(x_1) \\ -f(x_2) \\ \vdots \\ -f(x_n) \end{array} \right]$$

becomes to find a f among all the 1-Lipschitz functions such that

$$\max_{\|f\|_L \leq 1} E_{x \sim P_r} f(x) - E_{x \sim P_\theta} f(x)$$



Source: <https://vincentherrmann.github.io/blog/wasserstein/>

Training WGAN

- Training a generator $x = g_\theta(z)$, $z \sim p(z)$ such that x has a distribution P_θ that converges to P_r in the EM distance

$$\theta^* = \arg \min_{\theta} W(P_r, P_\theta)$$

where the EM distance is evaluated by

$$W(P_r, P_\theta) = \max_{\|f\|_L \leq 1} E_{x \sim P_r} f(x) - E_{z \sim p(z)} f(g_\theta(z))$$

- Assuming $\{f_w\}_{w \in \mathcal{W}}$ is a family of 1-Lipschitz (or k-Lipschitz) functions

$$W(P_r, P_\theta) = \max_{\{f_w\}_{w \in \mathcal{W}}} E_{x \sim P_r} f_w(x) - E_{z \sim p(z)} f_w(g_\theta(z))$$

- The final objective becomes

$$\theta^* = \arg \min_{\theta} \max_{w \in \mathcal{W}} E_{x \sim P_r} f_w(x) - E_{z \sim p(z)} f_w(g_\theta(z))$$

where f_w (critic) and g_θ (generator) can be neural networks

- Under mild conditions, $W(P_r, P_\theta)$ is differentiable w.r.t. θ

$$\nabla_\theta W(P_r, P_\theta) = -E_{z \sim p(z)} \nabla_\theta f_w(g_\theta(z))$$

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.

```

1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSPProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSPProp}(\theta, g_\theta)$ 
12: end while

```

GAN vs. WGAN

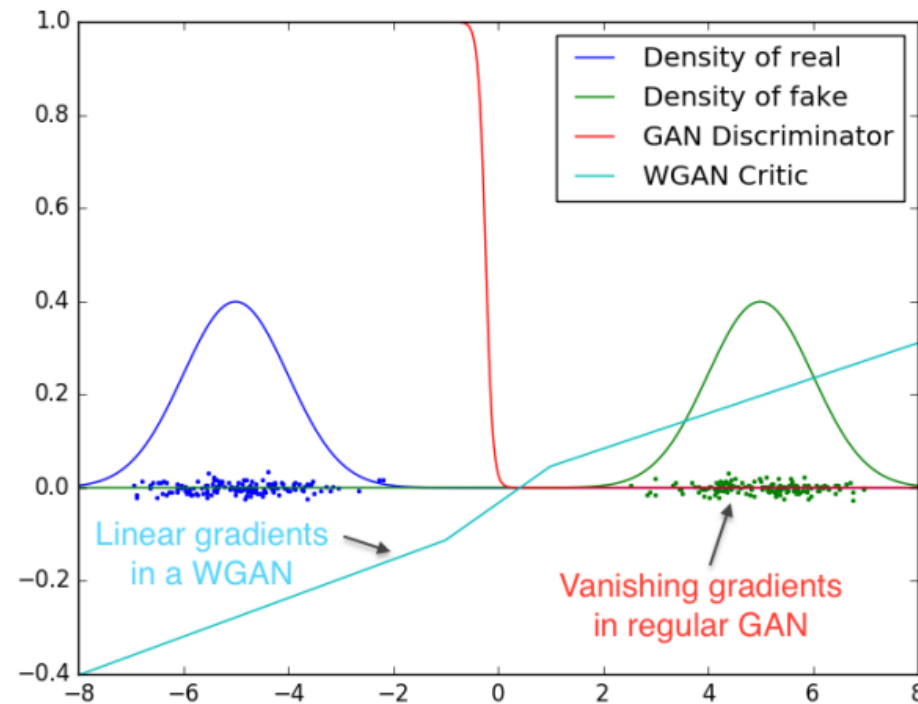


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

InfoGAN

- **Idea:** To learn unsupervisedly **disentangled representations** for data through GAN
 1. Separate input noise into two parts z, c
 2. Learn a generator such that its output $x = G(z, c)$ correlates highly with c , by maximizing the mutual information $I(c; G(z, c))$
- Objective function

$$\arg \min_{\theta^{(G)}} \max_{\theta^{(D)}} V(\theta^{(D)}, \theta^{(G)}) - \underbrace{\lambda I(c; G(z, c; \theta^{(G)}))}_{\text{mutual information}}$$

where $\lambda > 0$

Mutual Information

- (Definition) The mutual information between two random variables \mathbf{x}, \mathbf{y} is given by

$$\begin{aligned} I(\mathbf{x}; \mathbf{y}) &= \text{KL}(p(\mathbf{x}, \mathbf{y}) || p(\mathbf{x})p(\mathbf{y})) \\ &= E\left[\log \frac{p(\mathbf{x}, \mathbf{y})}{p(\mathbf{x})p(\mathbf{y})}\right] \\ &= H(\mathbf{x}) - H(\mathbf{x}|\mathbf{y}) \\ &= H(\mathbf{y}) - H(\mathbf{y}|\mathbf{x}) \end{aligned}$$

where $I(\mathbf{x}; \mathbf{y}) = 0$ if and only if \mathbf{x}, \mathbf{y} are independent

- $I(\mathbf{x}; \mathbf{y})$ indicates the reduction of uncertainty about \mathbf{x} (respectively, \mathbf{y}) after observing \mathbf{y} (respectively, \mathbf{x})

Variational Mutual Information Maximization

- Evaluating $I(c; x)$ with $x = G(z, c)$ needs to know the posterior $p(c|x)$, which is intractable

$$\begin{aligned} I(c; x) &= H(c) - H(c|x) \\ &= H(c) + E_{x \sim G(z, c)} E_{c' \sim p(c|x)} [\log p(c'|x)] \end{aligned}$$

- One way out of this difficulty is to introduce a variational function $q(c|x)$ for approximating $p(c|x)$

$$\begin{aligned} I(c; x) &= H(c) + E_{x \sim G(z, c)} E_{c' \sim p(c|x)} [\log p(c'|x) - \log q(c'|x)] \\ &\quad + E_{x \sim G(z, c)} E_{c' \sim p(c|x)} [\log q(c'|x)] \end{aligned}$$

- It is readily seen that

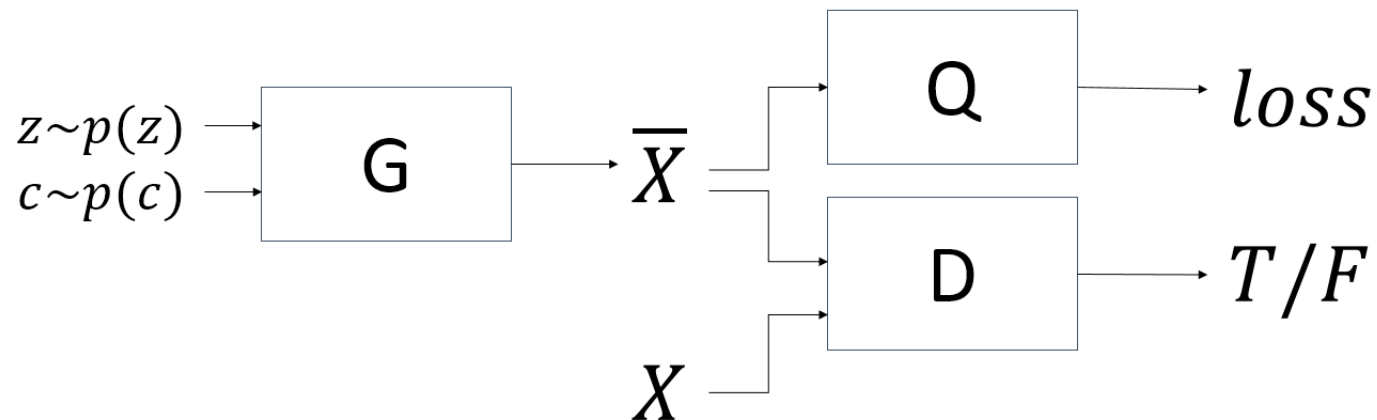
$$\begin{aligned} I(c; x) &= H(c) + E_{x \sim G(z, c)} [\text{KL}(p(c'|x) || q(c'|x))] \\ &\quad + E_{x \sim G(z, c)} E_{c' \sim p(c|x)} [\log q(c'|x)] \end{aligned}$$

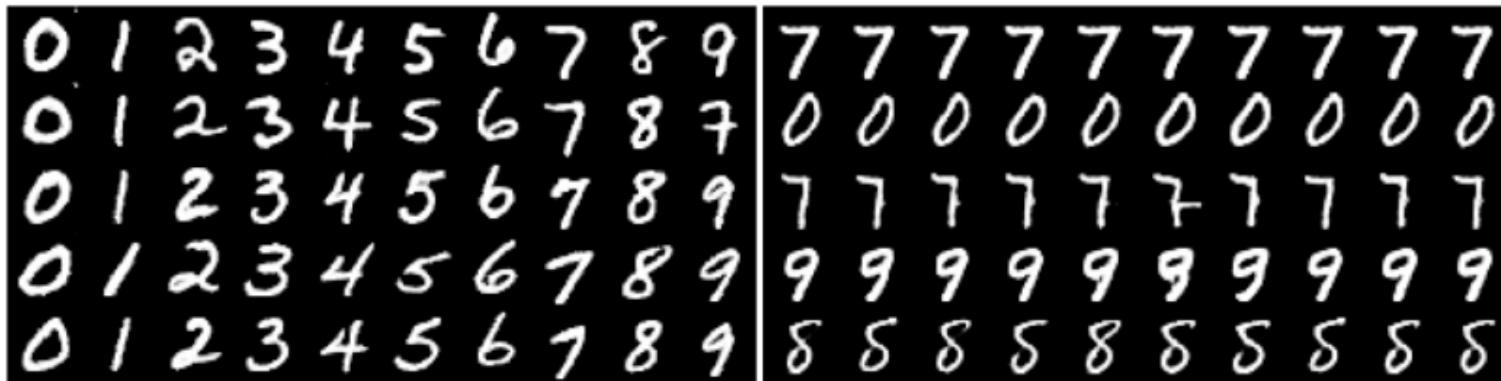
$$\begin{aligned}
&\geq H(c) + \underbrace{E_{x \sim G(z, c)} E_{c' \sim p(c|x)} [\log q(c'|x)]}_{\text{InfoNCE}} \\
&= H(c) + \underbrace{E_{c \sim p(c), x \sim G(z, c)} [\log q(c|x)]}_{\text{InfoNCE}}
\end{aligned}$$

- The objective function of InfoGAN then becomes

$$\arg \min_{\theta^{(G)}, \theta^{(Q)}} \max_{\theta^{(D)}} V(\theta^{(D)}, \theta^{(G)}) - \underbrace{\lambda (E_{c \sim p(c), x \sim G(z, c)} [\log q(c|x; \theta^{(Q)})])}_{\text{InfoNCE}}$$

- The term $E_{c \sim p(c), x \sim G(z, c)} [\log q(c|x; \theta^{(Q)})]$ can be evaluated by using the reparameterization trick (i.e., by inputting different z samples while fixing c and requiring that the output of Q be c again)



(a) Varying c_1 on InfoGAN (Digit type)(b) Varying c_1 on regular GAN (No clear meaning)

Deep Boltzmann Machines (DBM)

- An energy-based generative model with an explicit density over **binary** visible \mathbf{v} and hidden $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}$ variables

$$p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}; \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(-E(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}; \boldsymbol{\theta}))$$

where

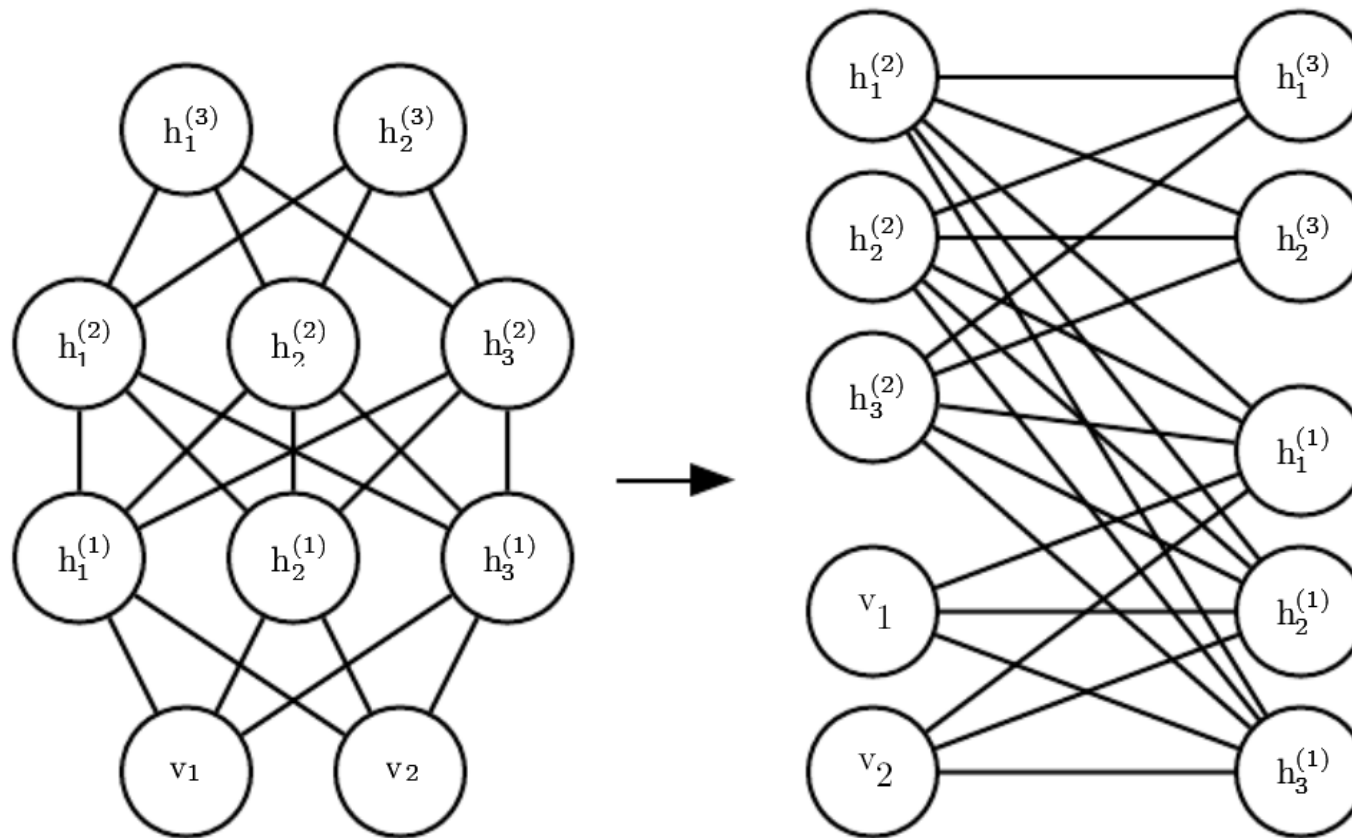
$$\begin{aligned} E(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}; \boldsymbol{\theta}) \\ = -\mathbf{v}^T \mathbf{W}^{(1)} \mathbf{h}^{(1)} - \mathbf{h}^{(1)T} \mathbf{W}^{(2)} \mathbf{h}^{(2)} - \mathbf{h}^{(2)T} \mathbf{W}^{(3)} \mathbf{h}^{(3)} \end{aligned}$$

and

$$\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}\}$$

- Note that bias terms are omitted for simplicity

- Graphical model for DBM, where odd layers can be separated from even layers to reveal a bipartite structure



- As a result, variables in odd layers are conditionally independent given even layers and vice versa; this enables block Gibbs sampling

- Likewise, it is seen that variables in a layer are conditionally independent given the neighbouring layers
- In the case of two hidden layers, we have

$$p(v_i = 1 | \mathbf{h}^{(1)}) = \sigma(\mathbf{W}_{i,:}^{(1)} \mathbf{h}^{(1)})$$

$$p(h_i^{(1)} = 1 | \mathbf{v}, \mathbf{h}^{(2)}) = \sigma(\mathbf{v}^T \mathbf{W}_{:,i}^{(1)} + \mathbf{W}_{i,:}^{(2)} \mathbf{h}^{(2)})$$

$$p(h_i^{(2)} = 1 | \mathbf{h}^{(1)}) = \sigma(\mathbf{h}^{(1)T} \mathbf{W}_{:,i}^{(2)})$$

- However, the posterior distribution of all hidden layers given the visible layer does not factorize because of interactions between layers

$$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)} | \mathbf{v}) \neq \prod_j p(h_j^{(1)} | \mathbf{v}) \prod_k p(h_k^{(2)} | \mathbf{v})$$

- Approximate inference needs to be sought

DBM Mean Field Inference

- To construct a factorial $Q(\mathbf{h}|\mathbf{v})$ for approximating $p(\mathbf{h}|\mathbf{v})$

$$p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}|\mathbf{v}) \approx Q(\mathbf{h}|\mathbf{v}) = \prod_j q(h_j^{(1)}|\mathbf{v}) \prod_k q(h_k^{(2)}|\mathbf{v})$$

- In the present case, all hidden variables $h_j^{(1)}, h_k^{(2)}$ are binary; these $q(h|\mathbf{v})$ must have a functional form of the Bernoulli distribution, i.e.

$$q(h_j^{(1)}|\mathbf{v}) = (\hat{h}_j^{(1)})^{h_j^{(1)}} (1 - \hat{h}_j^{(1)})^{(1-h_j^{(1)})}, \forall j$$

$$q(h_k^{(2)}|\mathbf{v}) = (\hat{h}_k^{(2)})^{h_k^{(2)}} (1 - \hat{h}_k^{(2)})^{(1-h_k^{(2)})}, \forall k$$

where $\hat{h}_j^{(1)}, \hat{h}_k^{(2)} \in [0, 1]$ are the corresponding parameters

- Carrying out the expectation (needs some work)

$$\tilde{q}_j(h_j|\mathbf{v}) = \exp(E_{q_{-j}}(\log p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}; \boldsymbol{\theta})))$$

yields the following fixed-point update equations

$$\hat{h}_j^{(1)} = \sigma \left(\sum_i v_i W_{i,j}^{(1)} + \sum_k W_{j,k}^{(2)} \hat{h}_k^{(2)} \right), \forall j$$

$$\hat{h}_k^{(2)} = \sigma \left(\sum_j W_{j,k}^{(2)} \hat{h}_j^{(1)} \right), \forall k$$

DBM Parameter Learning

- DBM learning has to confront both the intractable inference $p(\mathbf{h}|\mathbf{v})$ and the intractable partition function $Z(\boldsymbol{\theta})$
- Combined variational inference, learning, and MCMC is necessary
- The objective then becomes to find $\mathbf{W}^{(1)}, \mathbf{W}^{(2)}$ that minimize

$$\mathcal{L}(Q, \boldsymbol{\theta}) = \sum_i \sum_j v_i W_{i,j}^{(1)} \hat{h}_j^{(1)} + \sum_j \sum_k \hat{h}_j^{(1)} W_{j,k}^{(2)} \hat{h}_k^{(2)} - \log Z(\boldsymbol{\theta}) + H(Q)$$

which can be done via gradient descent

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \varepsilon \nabla_{\boldsymbol{\theta}} \mathcal{L}(Q, \boldsymbol{\theta})$$

(study Algorithm 20.1)

- In general, layer-wise pre-training is needed to arrive at a good model

Set ϵ , the step size, to a small positive number

Set k , the number of Gibbs steps, high enough to allow a Markov chain of $p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}; \boldsymbol{\theta} + \epsilon \Delta_{\boldsymbol{\theta}})$ to burn in, starting from samples from $p(\mathbf{v}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}; \boldsymbol{\theta})$.

Initialize three matrices, $\tilde{\mathbf{V}}$, $\tilde{\mathbf{H}}^{(1)}$ and $\tilde{\mathbf{H}}^{(2)}$ each with m rows set to random values (e.g., from Bernoulli distributions, possibly with marginals matched to the model's marginals).

while not converged (learning loop) **do**

Sample a minibatch of m examples from the training data and arrange them as the rows of a design matrix \mathbf{V} .

Initialize matrices $\hat{\mathbf{H}}^{(1)}$ and $\hat{\mathbf{H}}^{(2)}$, possibly to the model's marginals.

while not converged (mean field inference loop) **do**

$$\hat{\mathbf{H}}^{(1)} \leftarrow \sigma \left(\mathbf{V} \mathbf{W}^{(1)} + \hat{\mathbf{H}}^{(2)} \mathbf{W}^{(2)\top} \right).$$

$$\hat{\mathbf{H}}^{(2)} \leftarrow \sigma \left(\hat{\mathbf{H}}^{(1)} \mathbf{W}^{(2)} \right).$$

end while

$$\Delta \mathbf{W}^{(1)} \leftarrow \frac{1}{m} \mathbf{V}^\top \hat{\mathbf{H}}^{(1)}$$

$$\Delta \mathbf{W}^{(2)} \leftarrow \frac{1}{m} \hat{\mathbf{H}}^{(1)\top} \hat{\mathbf{H}}^{(2)}$$

for $l = 1$ to k (Gibbs sampling) **do**

Gibbs block 1:

$$\forall i, j, \tilde{V}_{i,j} \text{ sampled from } P(\tilde{V}_{i,j} = 1) = \sigma \left(\mathbf{W}_{j,:}^{(1)} \left(\tilde{\mathbf{H}}_{i,:}^{(1)} \right)^\top \right).$$

$$\forall i, j, \tilde{H}_{i,j}^{(2)} \text{ sampled from } P(\tilde{H}_{i,j}^{(2)} = 1) = \sigma \left(\tilde{\mathbf{H}}_{i,:}^{(1)} \mathbf{W}_{:,j}^{(2)} \right).$$

Gibbs block 2:

$$\forall i, j, \tilde{H}_{i,j}^{(1)} \text{ sampled from } P(\tilde{H}_{i,j}^{(1)} = 1) = \sigma \left(\tilde{\mathbf{V}}_{i,:} \mathbf{W}_{:,j}^{(1)} + \tilde{\mathbf{H}}_{i,:}^{(2)} \mathbf{W}_{j,:}^{(2)\top} \right).$$

end for

$$\Delta \mathbf{W}^{(1)} \leftarrow \Delta \mathbf{W}^{(1)} - \frac{1}{m} \mathbf{V}^\top \tilde{\mathbf{H}}^{(1)}$$

$$\Delta \mathbf{W}^{(2)} \leftarrow \Delta \mathbf{W}^{(2)} - \frac{1}{m} \tilde{\mathbf{H}}^{(1)\top} \tilde{\mathbf{H}}^{(2)}$$

$\mathbf{W}^{(1)} \leftarrow \mathbf{W}^{(1)} + \epsilon \Delta \mathbf{W}^{(1)}$ (this is a cartoon illustration, in practice use a more effective algorithm, such as momentum with a decaying learning rate)

$$\mathbf{W}^{(2)} \leftarrow \mathbf{W}^{(2)} + \epsilon \Delta \mathbf{W}^{(2)}$$

end while

Topics Not Covered

- Optimization for training deep models (Chapter 8)
- Representation learning (Chapter 15)
- Back-prop through random operations (REINFORCE, Chapter 20)
- BM for real-valued data (Chapter 20)
- Generative Stochastic Networks (Chapter 20)
- Deep Belief Networks (Chapter 20)
- Other generative models (Chapter 20)