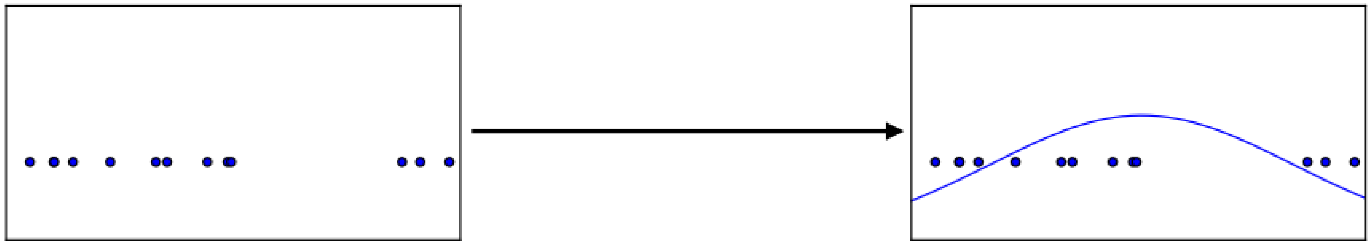
Chapter 20

Deep Generative Models

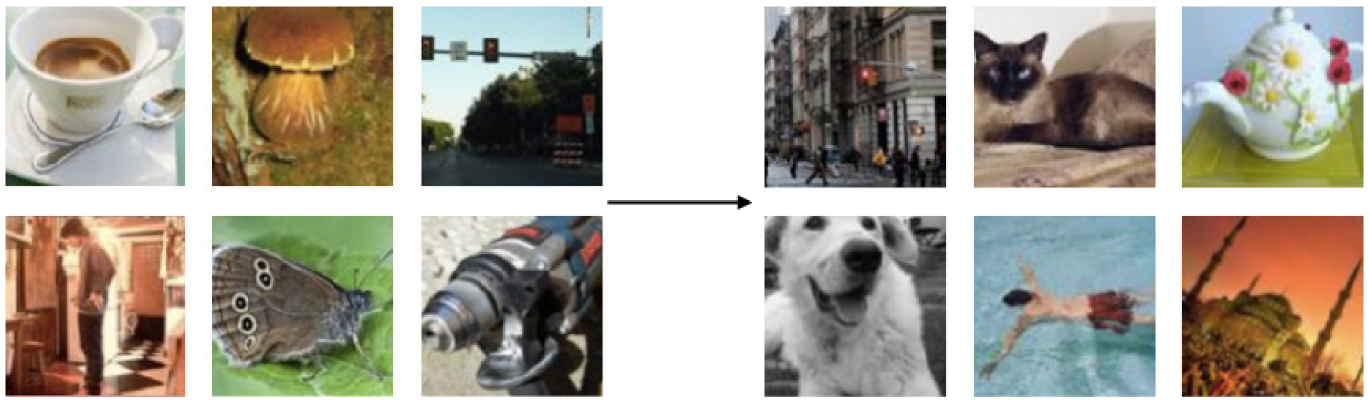
Generative Models

Models that are able to

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



- Generate samples from a (likely implicit) distribution

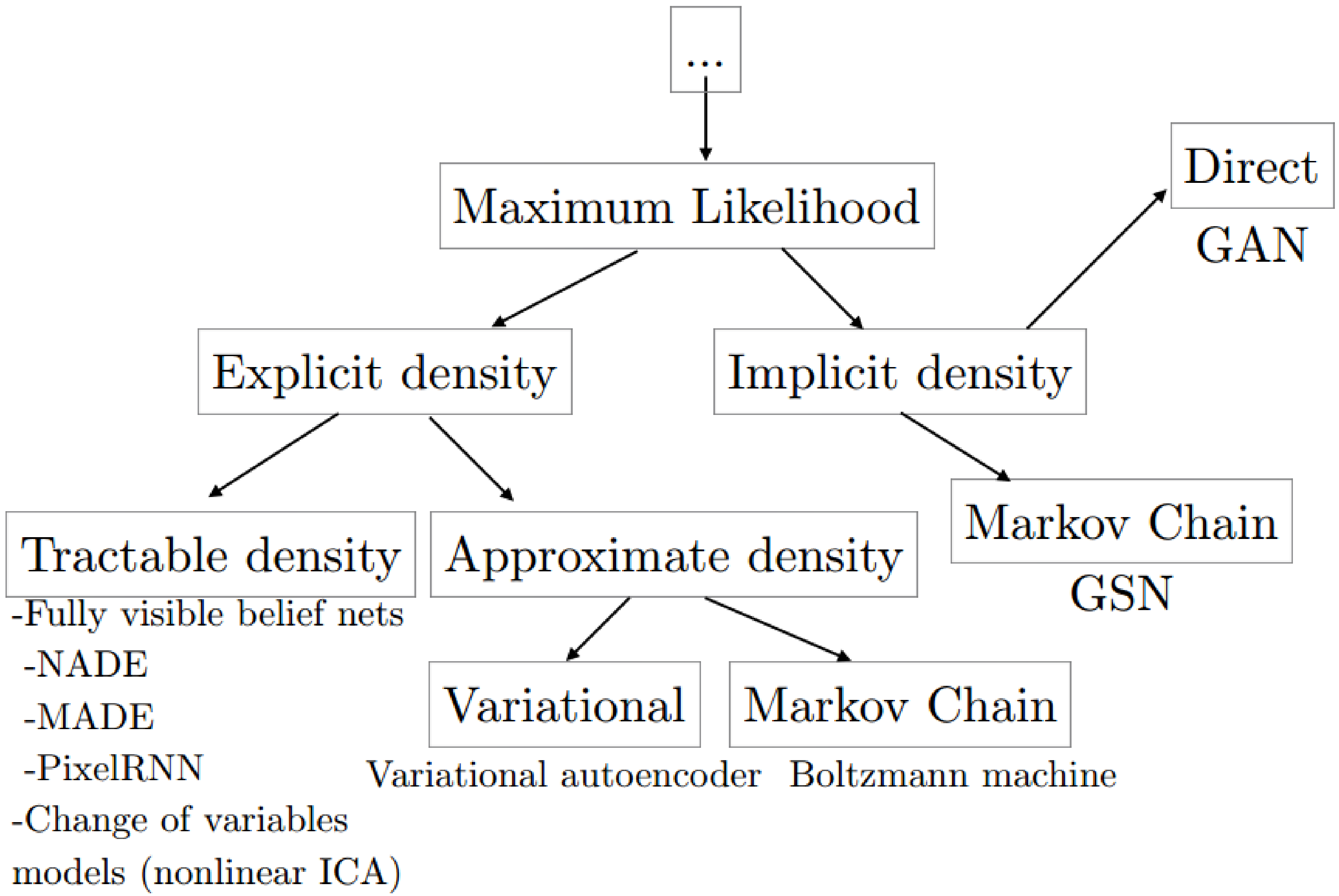


Training examples Model samples

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.

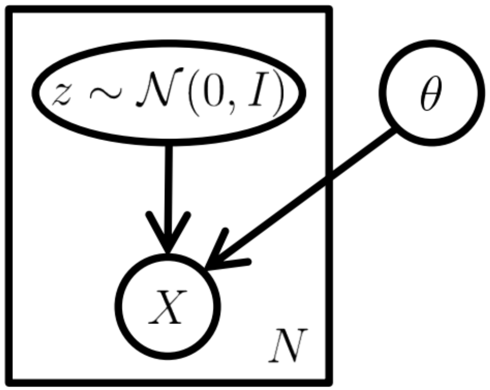
Taxonomy of Generative Models



- Explicit density, $p_{\text{model}}(\boldsymbol{x};\boldsymbol{\theta})$
  - 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

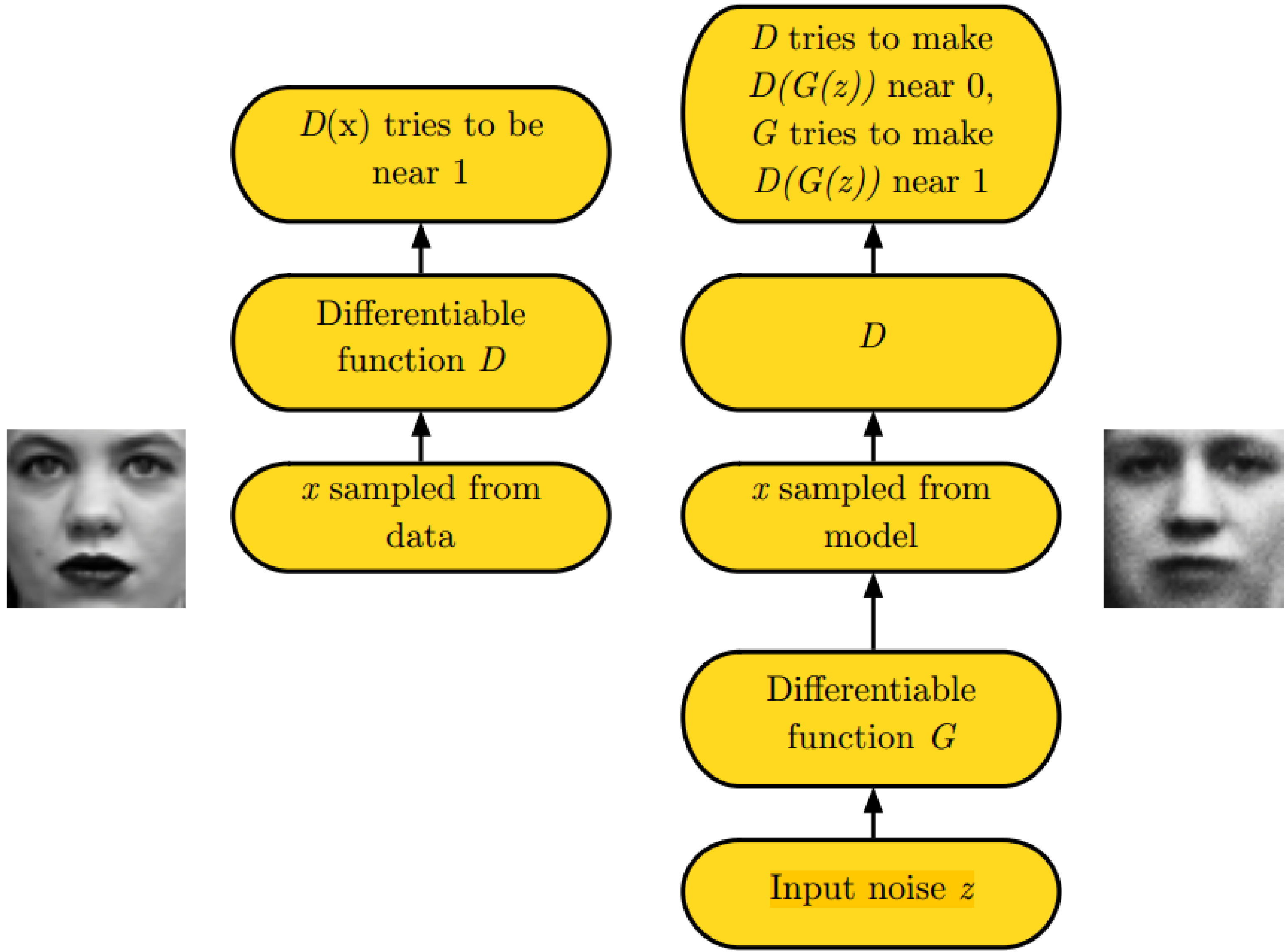
Generative Adversarial Networks (GAN)

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



- 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)


D(x) tries to be near 1
Differentiable function D
x sampled from data
D tries to make $D(G(z))$ near 0, G tries to make $D(G(z))$ near 1
D
x sampled from model
Differentiable function G
Input noise z

Training GANs: Two-Player Minimax Game

- $D(\boldsymbol{x};\boldsymbol{\theta}^{(D)}), G(\boldsymbol{z};\boldsymbol{\theta}^{(G)})$ are implemented with neural networks, and each has their own cost to minimize
  - **Discriminator cost** (cross-entropy cost)

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

    where $D(\boldsymbol{x})$ denotes the probability of $\boldsymbol{x}$ 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

$$D_G^*(\boldsymbol{x}) = \frac{p_{\mathsf{data}}(\boldsymbol{x})}{p_{\mathsf{data}}(\boldsymbol{x}) + p_{\mathsf{g}}(\boldsymbol{x})}$$

  which can be obtained by having

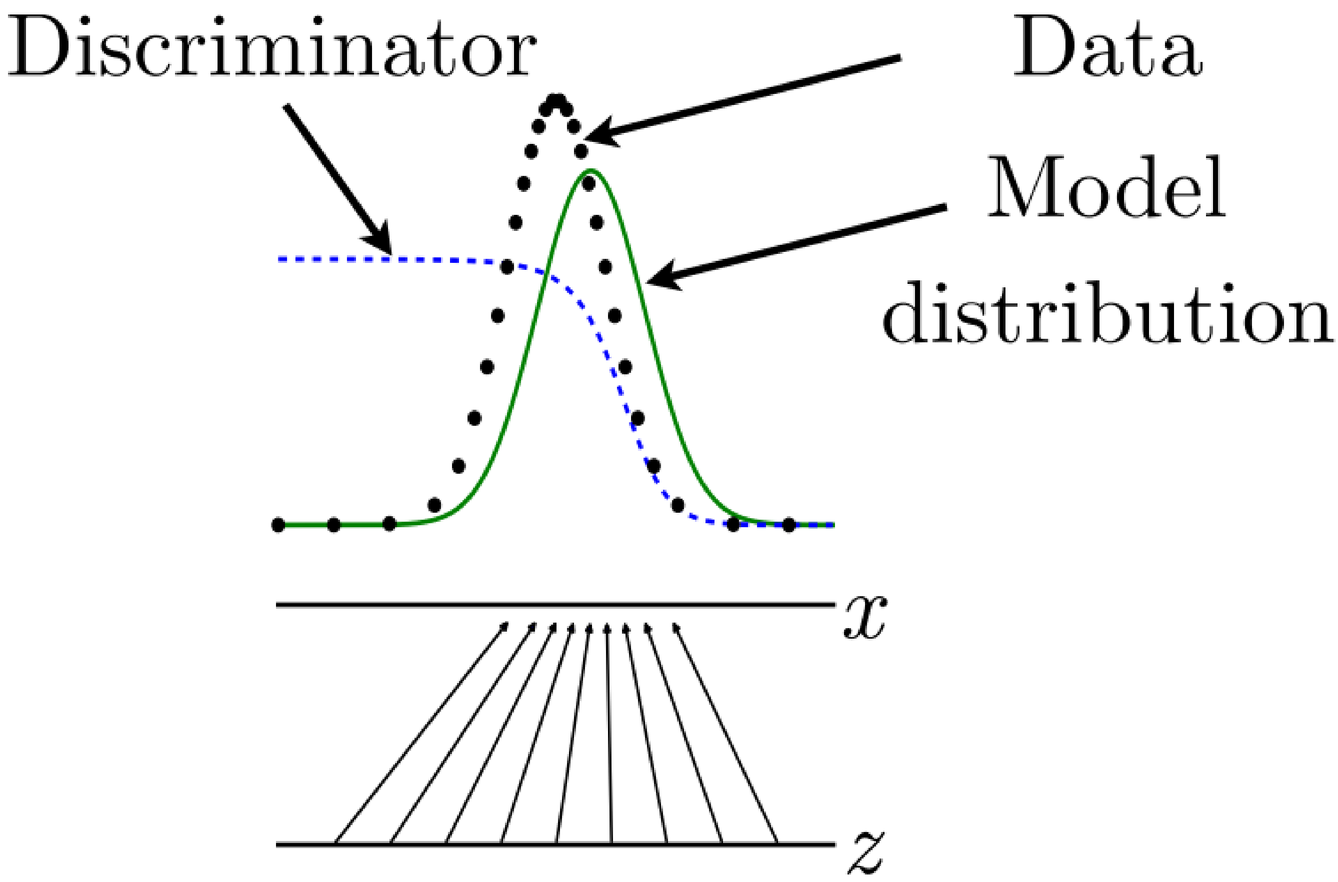
$$\frac{\delta}{\delta D(\boldsymbol{x})} J^{(D)}(\boldsymbol{x}) = 0$$

- When given enough capacity, the discriminator obtains an estimate

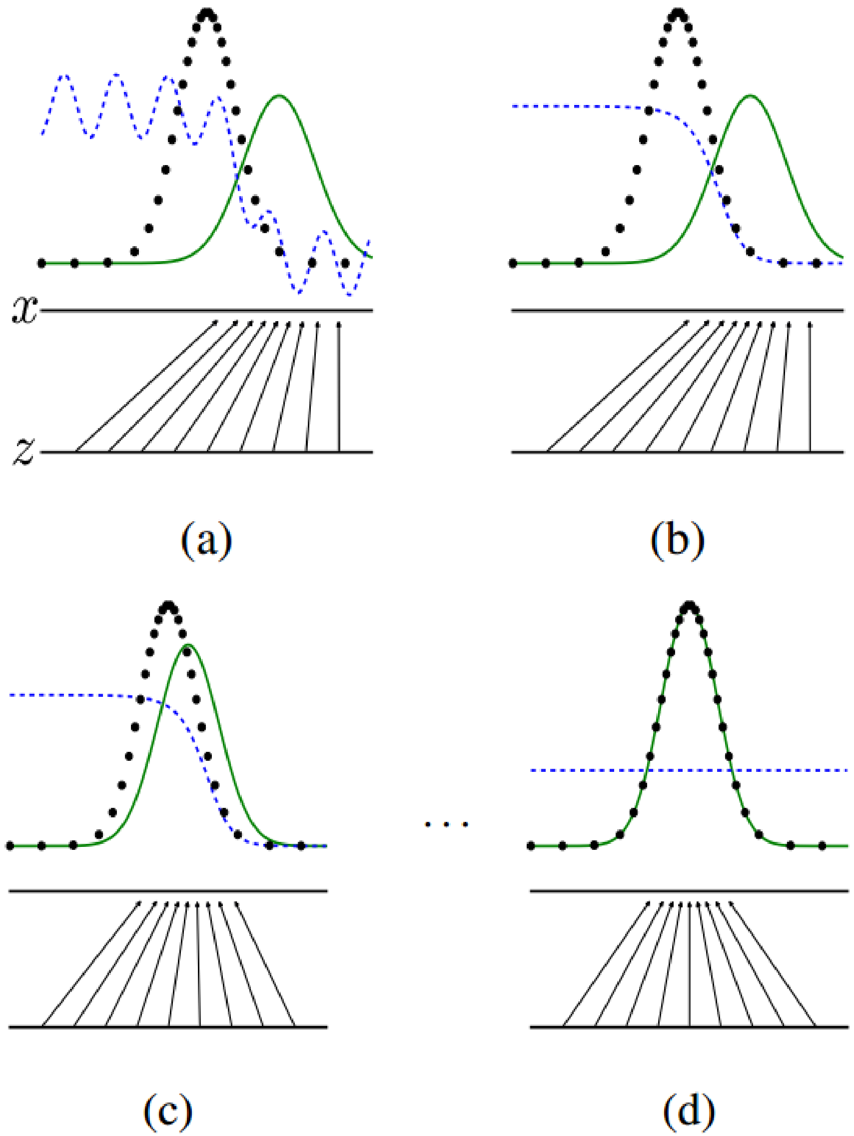
$$\frac{p_{\mathsf{data}(\boldsymbol{x})}}{p_{\mathsf{g}(\boldsymbol{x})}}$$

  at every $\boldsymbol{x}$

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


Discriminator
Data
Model
distribution
x
z

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



The Optimal Generator

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

$$
\begin{aligned}
&\arg\min_{p_{\text{g}}} E_{\boldsymbol{x}\sim p_{\text{data}}} \log D_G^*(\boldsymbol{x}) + E_{\boldsymbol{x}\sim p_{\text{g}}} \log(1 - D_G^*(G(\boldsymbol{x}))) \\
&= \arg\min_{p_{\text{g}}} E_{\boldsymbol{x}\sim p_{\text{data}}} \log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{g}}(\boldsymbol{x})} + E_{\boldsymbol{x}\sim p_{\text{g}}} \log \frac{p_{\text{g}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{g}}(\boldsymbol{x})} \\
&= \arg\min_{p_{\text{g}}} -\log(4) + \mathsf{KL}\left(p_{\text{data}} \,\|\, \frac{p_{\text{data}} + p_{\text{g}}}{2}\right) + \mathsf{KL}\left(p_{\text{g}} \,\|\, \frac{p_{\text{data}} + p_{\text{g}}}{2}\right) \\
&= \arg\min_{p_{\text{g}}} -\log(4) + 2 \times \mathsf{JSD}(p_{\text{data}} \,\|\, p_{\text{g}})
\end{aligned}
$$

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

- Remarks
  - The optimization is done w.r.t. p_{g} directly
  - The analysis for the discriminator is done w.r.t. $D(\boldsymbol{x})$
  - Enough capacity in both contexts means that $D_G^*(\boldsymbol{x})$ and $p_{\text{g}}^*(\boldsymbol{x})$ can be implemented by $D(\boldsymbol{x};\boldsymbol{\theta}^{(D)*})$ and $G(\boldsymbol{z};\boldsymbol{\theta}^{(G)*})$, 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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

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

end for

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

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

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 $D^*_G(\boldsymbol{x})$ given G, and p_{g} is updated to improve the criterion (reduce the cost)

$$E_{\boldsymbol{x}\sim p_{\text{data}}} \log D^*_G(\boldsymbol{x}) + E_{\boldsymbol{x}\sim p_{\text{g}}} \log(1 - D^*_G(G(\boldsymbol{x})))$$

  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

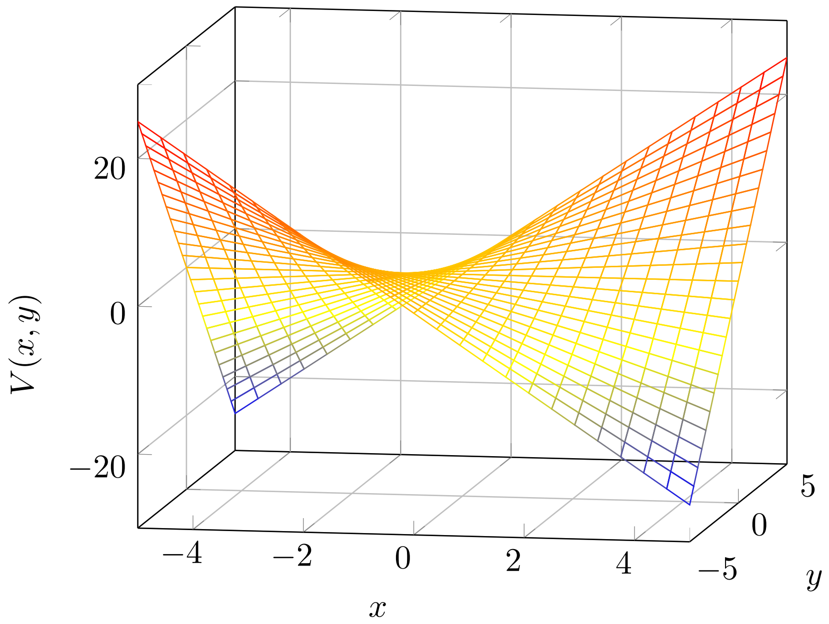
$$x(t + \Delta t) = x(t) - \Delta t \frac{\partial}{\partial x(t)} V(x(t), y(t))$$
$$y(t + \Delta t) = y(t) + \Delta t \frac{\partial}{\partial y(t)} V(x(t), y(t))$$

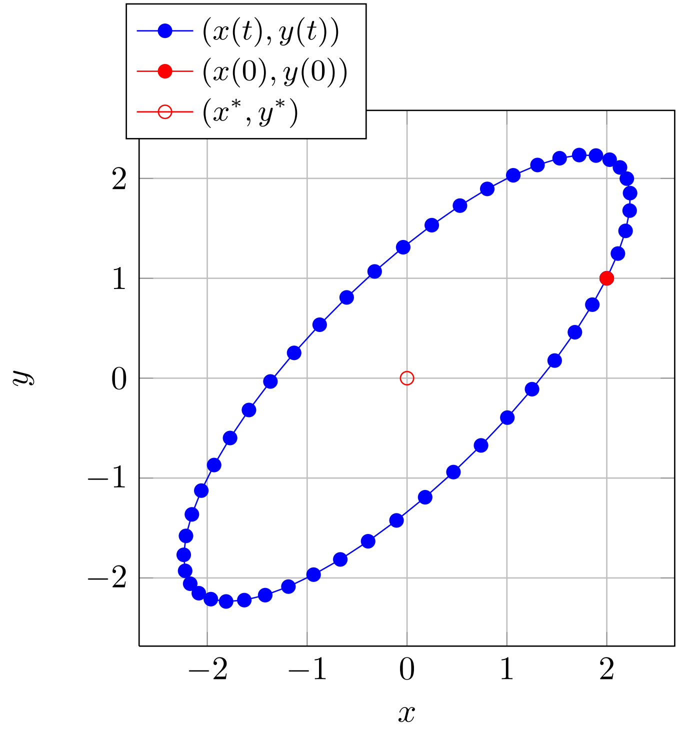
- This amounts to solving

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

which has a solution of the form

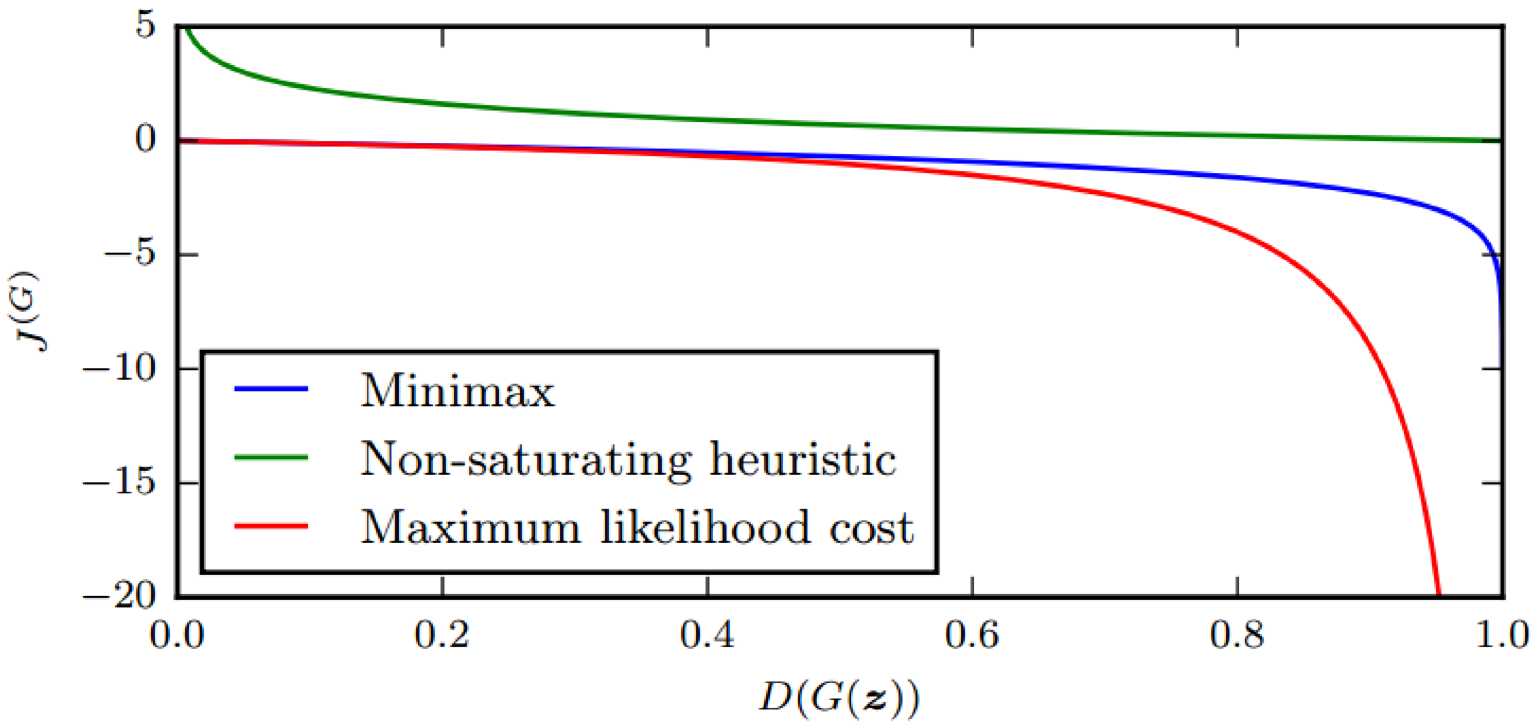
$$x(t) = x(0)\cos(t) + y(0)\sin(t)$$
$$y(t) = x(0)\sin(t) + y(0)\cos(t)$$





Other Games

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



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

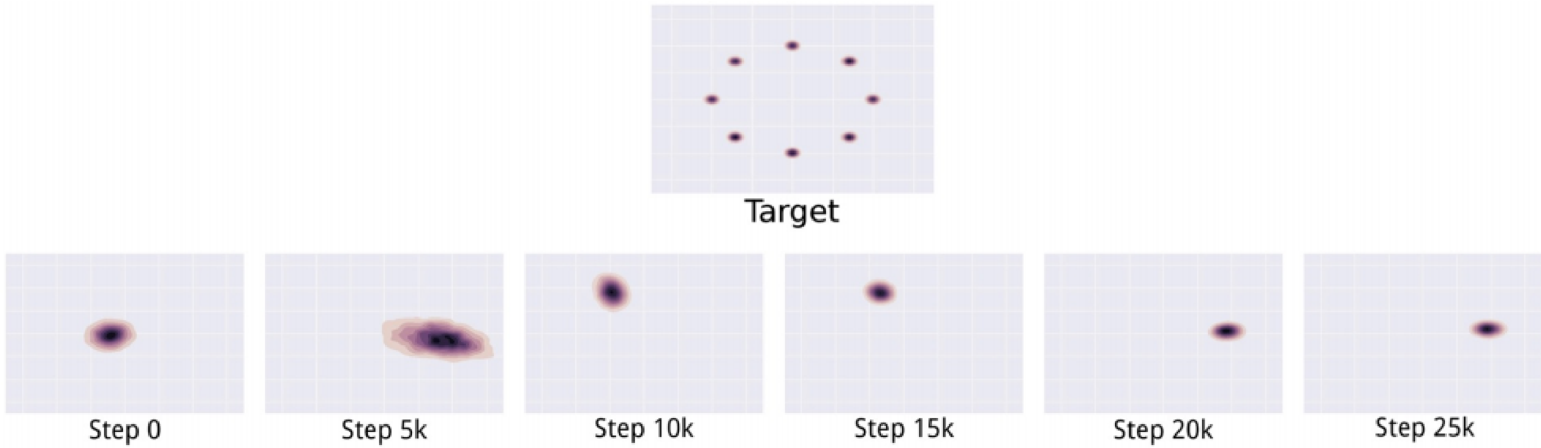
$$J^{(G)} = -E_z \log D(G(z))$$

- 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 $\boldsymbol{z}$ to the same $\boldsymbol{x}$



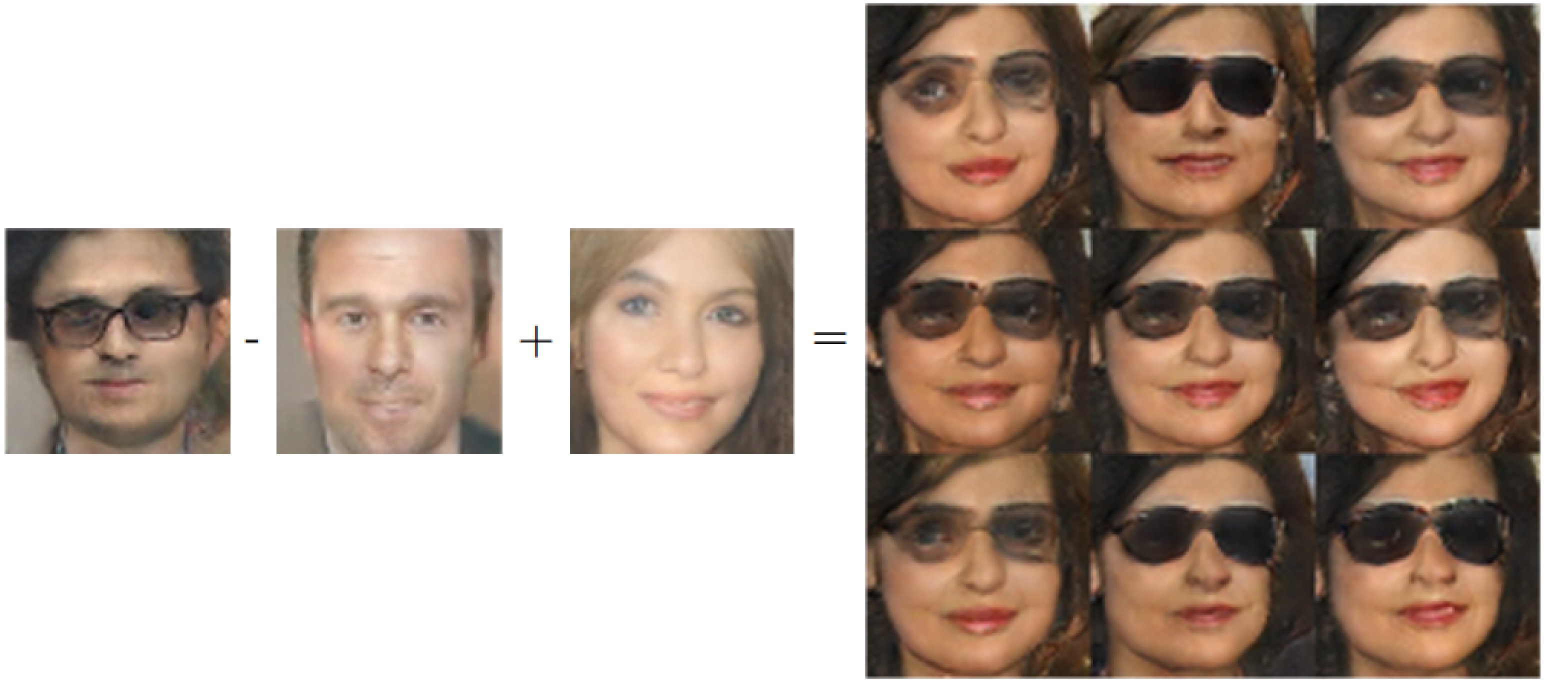
Top: Data distribution (Mixture Gaussian)

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

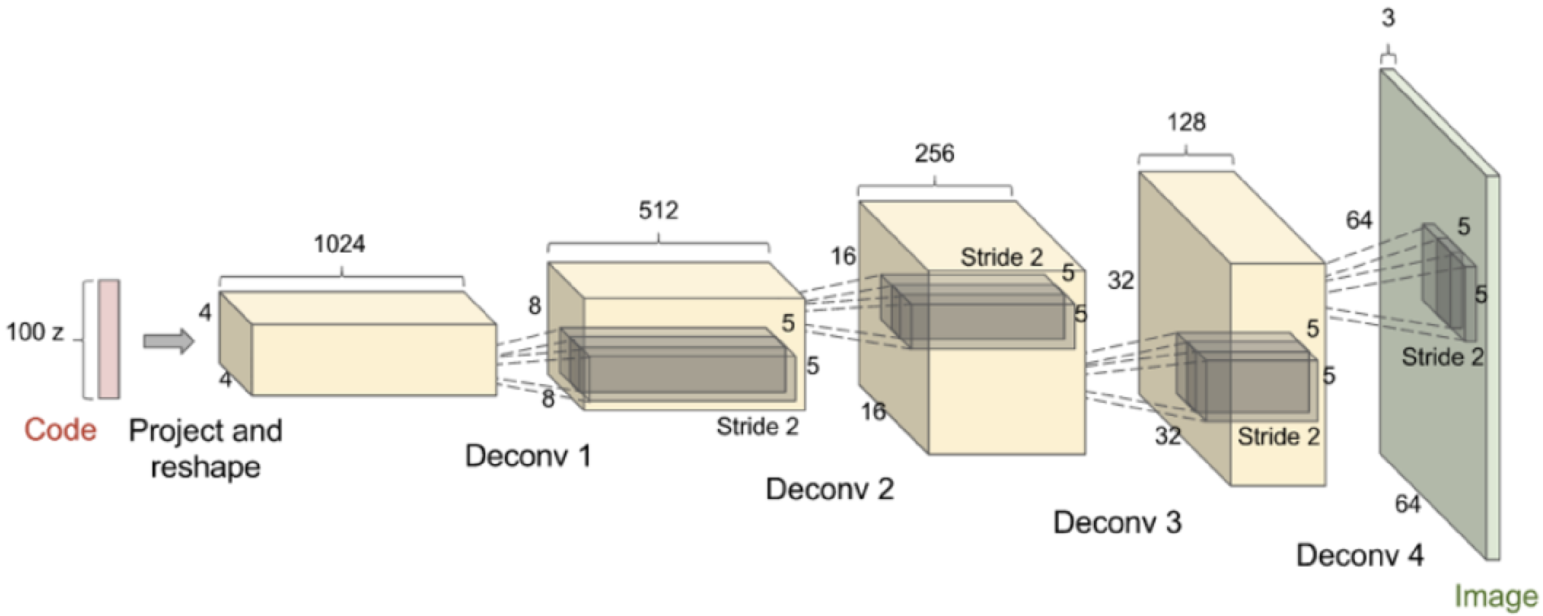
Learned Representation

- The generator can learn a distributed representation that disentangles high-level concepts, e.g. gender vs. wearing glasses



DCGAN

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



Deep Boltzmann Machines (DBM)

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

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

where

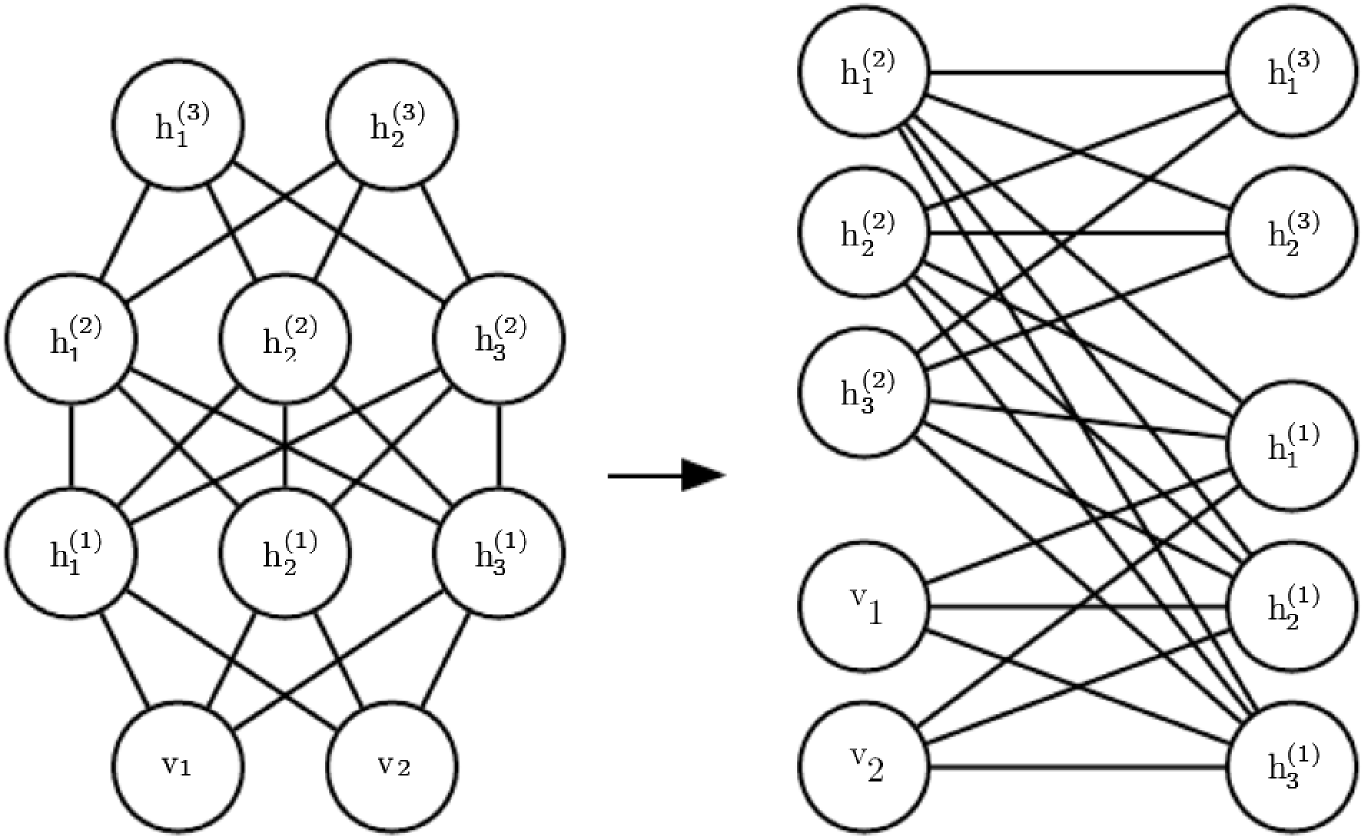
$$\begin{aligned} & E(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \boldsymbol{h}^{(3)}; \boldsymbol{\theta}) \\ &= -\boldsymbol{v}^T \boldsymbol{W}^{(1)} \boldsymbol{h}^{(1)} - \boldsymbol{h}^{(1)T} \boldsymbol{W}^{(2)} \boldsymbol{h}^{(2)} - \boldsymbol{h}^{(2)T} \boldsymbol{W}^{(3)}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

and

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

- 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

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

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

$$p(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}|\boldsymbol{v}) \neq \prod_j p(h^{(1)}_j|\boldsymbol{v}) \prod_k p(h^{(2)}_k|\boldsymbol{v})$$

- Approximate inference needs to be sought

DBM Mean Field Inference

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

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

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

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

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

- Carrying out the expectation (needs some work)

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

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 $p(\boldsymbol{h}|\boldsymbol{v})$ and the intractable partition function $Z(\boldsymbol{\theta})$
- Combined variational inference, learning, and MCMC is necessary
- The objective then becomes to find $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}$ 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 $p(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}; \boldsymbol{\theta} + \epsilon\Delta_{\boldsymbol{\theta}})$ to burn in, starting from samples from $p(\boldsymbol{v}, \boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}; \boldsymbol{\theta})$.
Initialize three matrices, $\tilde{\boldsymbol{V}}$, $\tilde{\boldsymbol{H}}^{(1)}$ and $\tilde{\boldsymbol{H}}^{(2)}$ 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 $\boldsymbol{V}$.
  Initialize matrices $\hat{\boldsymbol{H}}^{(1)}$ and $\hat{\boldsymbol{H}}^{(2)}$, possibly to the model's marginals.
  while not converged (mean field inference loop) **do**
    $\hat{\boldsymbol{H}}^{(1)} \leftarrow \sigma\left(\boldsymbol{V}\boldsymbol{W}^{(1)} + \hat{\boldsymbol{H}}^{(2)}\boldsymbol{W}^{(2)\top}\right)$.
    $\hat{\boldsymbol{H}}^{(2)} \leftarrow \sigma\left(\hat{\boldsymbol{H}}^{(1)}\boldsymbol{W}^{(2)}\right)$.
  end while
  $\Delta_{\boldsymbol{W}^{(1)}} \leftarrow \frac{1}{m}\boldsymbol{V}^\top\hat{\boldsymbol{H}}^{(1)}$
  $\Delta_{\boldsymbol{W}^{(2)}} \leftarrow \frac{1}{m}\hat{\boldsymbol{H}}^{(1)\top}\hat{\boldsymbol{H}}^{(2)}$
  for $l = 1$ to k (Gibbs sampling) **do**
    Gibbs block 1:
    $\forall i, j, \tilde{V}_{i,j}$ sampled from $P(\tilde{V}_{i,j} = 1) = \sigma\left(\boldsymbol{W}_{j,:}^{(1)}\left(\tilde{\boldsymbol{H}}_{i,:}^{(1)}\right)^\top\right)$.
    $\forall i, j, \tilde{H}_{i,j}^{(2)}$ sampled from $P(\tilde{H}_{i,j}^{(2)} = 1) = \sigma\left(\tilde{\boldsymbol{H}}_{i,:}^{(1)}\boldsymbol{W}_{:,j}^{(2)}\right)$.
    Gibbs block 2:
    $\forall i, j, \tilde{H}_{i,j}^{(1)}$ sampled from $P(\tilde{H}_{i,j}^{(1)} = 1) = \sigma\left(\tilde{\boldsymbol{V}}_{i,:}\boldsymbol{W}_{:,j}^{(1)} + \tilde{\boldsymbol{H}}_{i,:}^{(2)}\boldsymbol{W}_{j,:}^{(2)\top}\right)$.
  end for
  $\Delta_{\boldsymbol{W}^{(1)}} \leftarrow \Delta_{\boldsymbol{W}^{(1)}} - \frac{1}{m}\boldsymbol{V}^\top\tilde{\boldsymbol{H}}^{(1)}$
  $\Delta_{\boldsymbol{W}^{(2)}} \leftarrow \Delta_{\boldsymbol{W}^{(2)}} - \frac{1}{m}\tilde{\boldsymbol{H}}^{(1)\top}\tilde{\boldsymbol{H}}^{(2)}$
  $\boldsymbol{W}^{(1)} \leftarrow \boldsymbol{W}^{(1)} + \epsilon\Delta_{\boldsymbol{W}^{(1)}}$ (this is a cartoon illustration, in practice use a more effective algorithm, such as momentum with a decaying learning rate)
  $\boldsymbol{W}^{(2)} \leftarrow \boldsymbol{W}^{(2)} + \epsilon\Delta_{\boldsymbol{W}^{(2)}}$
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)