Deep Generative Models

Changyou Chen

Department of Computer Science and Engineering
Universitpy at Buffalo, SUNY
changyou@buffalo.edu

April 11, 2019

Extension: Semi-supervised VAE

1. What if we have both labeled and unlabeled data?
2. Implement by conditioning the encoder and decoder to something:
   - Represented by some random variable $\mathbf{y}$, *e.g.*, it could a label.

Generative process

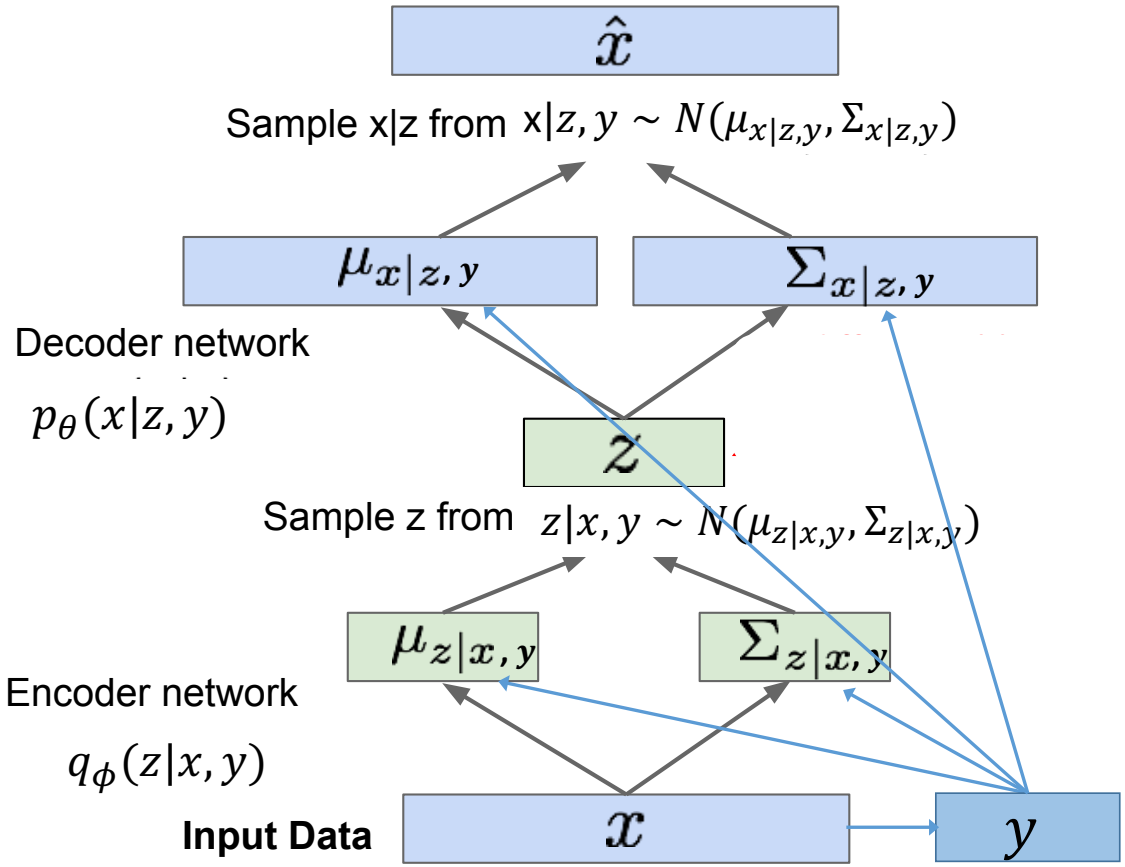
$$p(y) = \text{Cat}(y|\pi), \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}),$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_{\mathbf{x}|\mathbf{z},y}, \Sigma_{\mathbf{x}|\mathbf{z},y}\right)$$

Inference model

$$q_\phi(\mathbf{z}|y, \mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \mu_{\mathbf{z}|\mathbf{x},y}, \Sigma_{\mathbf{z}|\mathbf{x},y}\right), \quad q_\phi(y|\mathbf{x}) = \text{Cat}(y|\pi_\phi(\mathbf{x}))$$

Extension: Semi-supervised VAE

1. What if we have both labeled and unlabeled data?
2. Implement by conditioning the encoder and decoder to something:



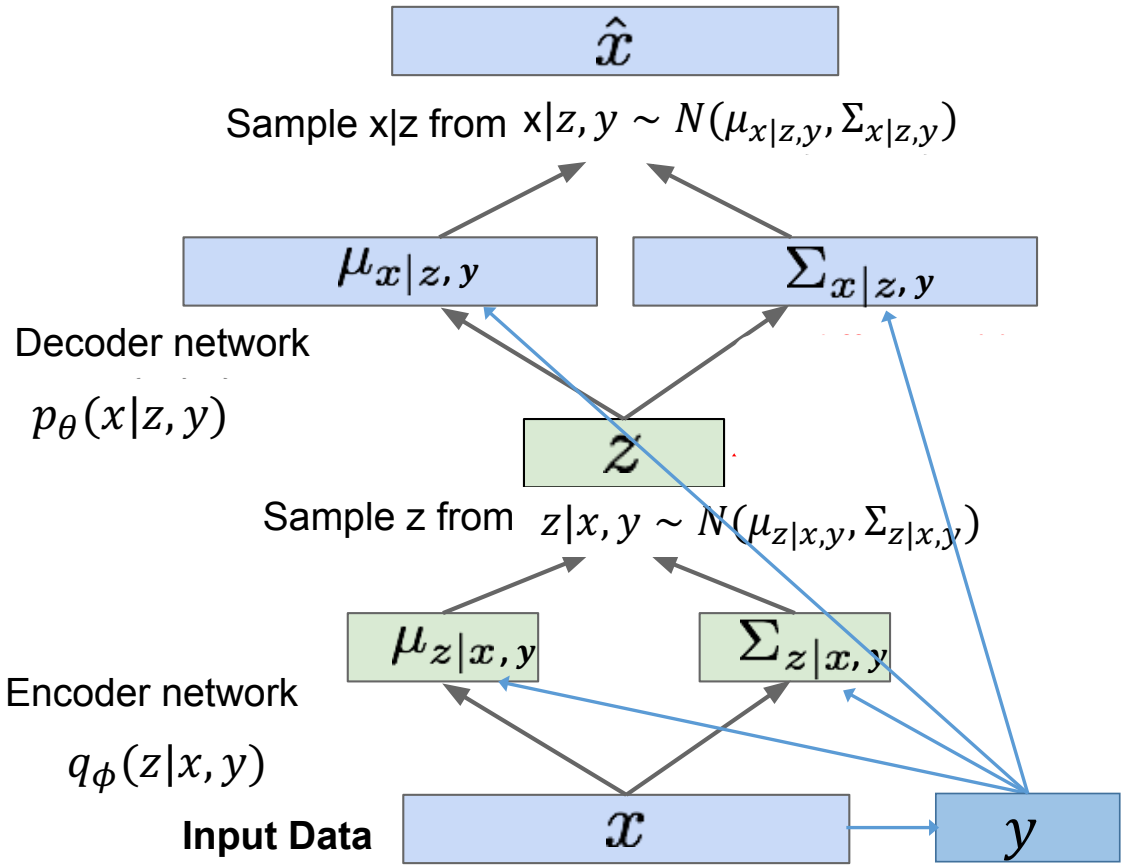
G: $p(y) = \text{Cat}(y|\pi), p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), p_\theta(\mathbf{x}|y,\mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_{\mathbf{x}|\mathbf{z},y}, \Sigma_{\mathbf{x}|\mathbf{z},y}\right)$

I: $q_\phi(\mathbf{z}|y,\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; \mu_{\mathbf{z}|\mathbf{x},y}, \Sigma_{\mathbf{z}|\mathbf{x},y}\right), \quad q_\phi(y|\mathbf{x}) = \text{Cat}(y|\pi_\phi(\mathbf{x}))$

Extension: Semi-supervised VAE

1. What if we have both labeled and unlabeled data?
2. Implement by conditioning the encoder and decoder to something:

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi) = \mathbb{E}_{\mathbf{z}, y|\mathbf{x} \sim q_\phi}\left[\log p_\theta(\mathbf{x}|\mathbf{z}, y)\right] - D_{KL}\left(q_\phi(\mathbf{z}, y|\mathbf{x}) \| p_\theta(\mathbf{z}, y)\right)$$



Extension: Semi-supervised VAE

For data with labels:

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi) = \mathbb{E}_{\mathbf{z}, y|\,\mathbf{x}\sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}\,|\,\mathbf{z}, y)\right] - D_{KL}\left(q_\phi(\mathbf{z}, y|\,\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}, y)\right)$$
$$= \mathbb{E}_{\mathbf{z}, y|\,\mathbf{x}\sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}\,|\,\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}, y|\,\mathbf{x})\right]$$

Extension: Semi-supervised VAE

For data with labels:

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi) = \mathbb{E}_{\mathbf{z}, y|\,\mathbf{x} \sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}\,|\,\mathbf{z}, y)\right] - D_{KL}\left(q_\phi(\mathbf{z}, y|\,\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}, y)\right)$$
$$= \mathbb{E}_{\mathbf{z}, y|\,\mathbf{x} \sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}\,|\,\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}, y|\,\mathbf{x})\right]$$

For data without labels:

$$\mathcal{L}_u(\mathbf{x}, \boldsymbol{\theta}, \phi) = \sum_y \mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi)$$
$$= \sum_y q_\phi(y|\,\mathbf{x})\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}\,|\,\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}, y|\,\mathbf{x})\right]$$

Extension: Semi-supervised VAE

For data with labels:

$$\mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi) = \mathbb{E}_{\mathbf{z}, y|\mathbf{x} \sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y)\right] - D_{KL}\left(q_\phi(\mathbf{z}, y|\mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z}, y)\right)$$
$$= \mathbb{E}_{\mathbf{z}, y|\mathbf{x} \sim q_\phi}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}, y|\mathbf{x})\right]$$

For data without labels:

$$\mathcal{L}_u(\mathbf{x}, \boldsymbol{\theta}, \phi) = \sum_y \mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi)$$
$$= \sum_y q_\phi(y|\mathbf{x})\left[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z}, y) + \log p_{\boldsymbol{\theta}}(y) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}, y|\mathbf{x})\right]$$

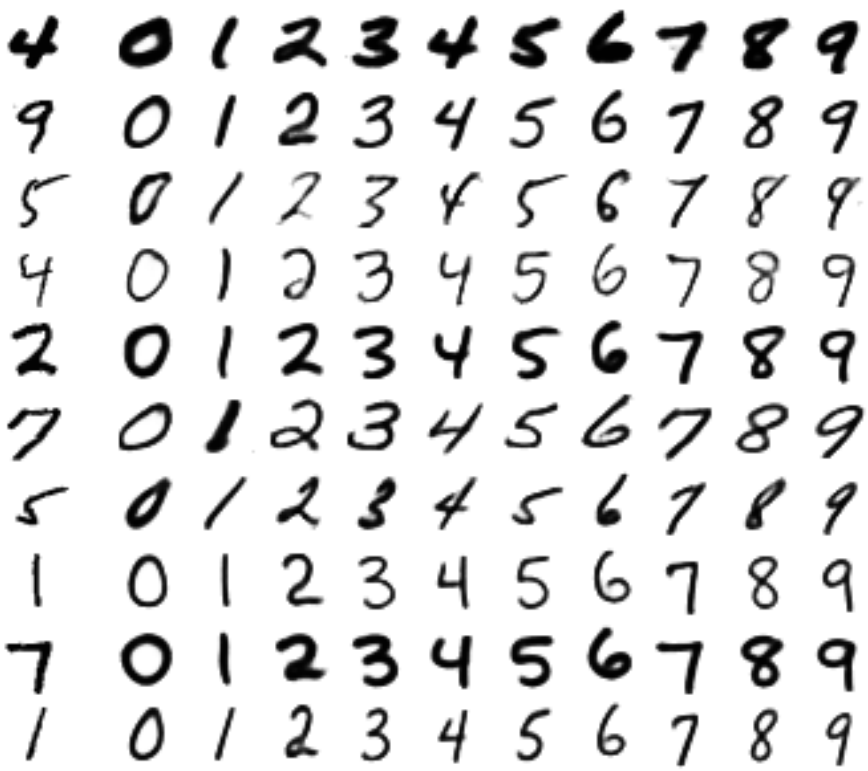
Total loss:

$$\tilde{L} = \mathcal{L}(\mathbf{x}, y, \boldsymbol{\theta}, \phi) + \mathcal{L}_u(\mathbf{x}, \boldsymbol{\theta}, \phi)$$

Extension: Semi-supervised VAE

(a) Handwriting styles for MNIST obtained by fixing the class label and varying the 2D latent variable $\mathbf{z}$

Extension: Semi-supervised VAE



(b) MNIST analogies

(c) SVHN analogies

Analogical reasoning with generative semi-supervised models using a high-dimensional $\mathbf{z}$-space. The leftmost columns show images from the test set. The other columns show analogical fantasies of $\mathbf{x}$ by the generative model, where the latent variable $\mathbf{z}$ of each row is set to the value inferred from the test-set image on the left by the inference network. Each column corresponds to a class label $\mathbf{y}$.

Questions

Can we use the VAE framework to do image segmentation? If so, how?

Summary

Pros:

- Principled approach to generative models.
- Allows inference of $q(\mathbf{z} \mid \mathbf{x})$, may be useful feature representation for other tasks.

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN.
- Samples blurrier and lower quality compared to state-of-the-art (GANs).

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, *e.g.*, normalizing flows.
- Incorporating structure in latent variables.

Summary of VAE

Pros:

- Principled approach to generative models.
- Allows inference of $q(\mathbf{z} \mid \mathbf{x})$, may be useful feature representation for other tasks.

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN.
- Samples blurrier and lower quality compared to state-of-the-art (GANs).

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, *e.g.*, normalizing flows.
- Incorporating structure in latent variables.

Generative Adversarial Networks (GAN)

Generative Models

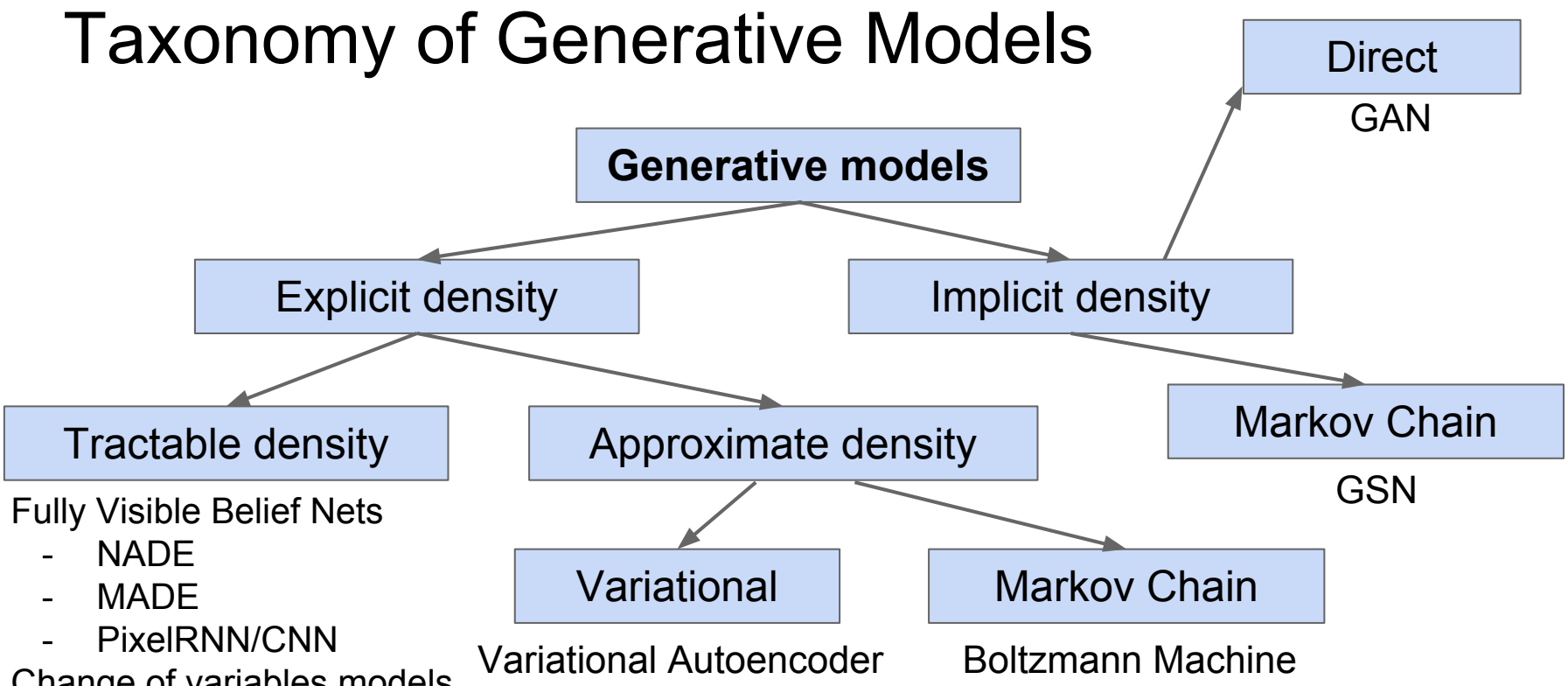


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Motivation

1. PixelCNNs define tractable density function; Perform learning via maximum likelihood estimator:

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i | x_1, \cdots, x_{i-1})$$

2. VAEs introduce latent variable $\mathbf{z}$ into the model; the likelihood is an intractable integration:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x} \,|\, \mathbf{z}) \mathrm{d}\, \mathbf{z}$$

3. Learning/inference has to resorted to some approximations, *e.g.*, by optimizing on a lower bound of $p_\theta(\mathbf{x})$.
4. What if we give up on explicitly modeling density, and just want ability to sample?
5. GANs do not explicitly model a density function; Instead, it learns to generate samples from the training distribution via a game-theoretic approach.

Motivation

1. PixelCNNs define tractable density function; Perform learning via maximum likelihood estimator:

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i | x_1, \cdots, x_{i-1})$$

2. VAEs introduce latent variable $\mathbf{z}$ into the model; the likelihood is an intractable integration:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x} \,|\, \mathbf{z}) \mathrm{d}\, \mathbf{z}$$

3. Learning/inference has to resorted to some approximations, *e.g.*, by optimizing on a lower bound of $p_\theta(\mathbf{x})$.
4. What if we give up on explicitly modeling density, and just want ability to sample?
5. GANs do not explicitly model a density function; Instead, it learns to generate samples from the training distribution via a game-theoretic approach.

Motivation

1. PixelCNNs define tractable density function; Perform learning via maximum likelihood estimator:

$$p_\theta(\mathbf{x}) = \prod_{i=1}^{n} p_\theta(x_i | x_1, \cdots, x_{i-1})$$

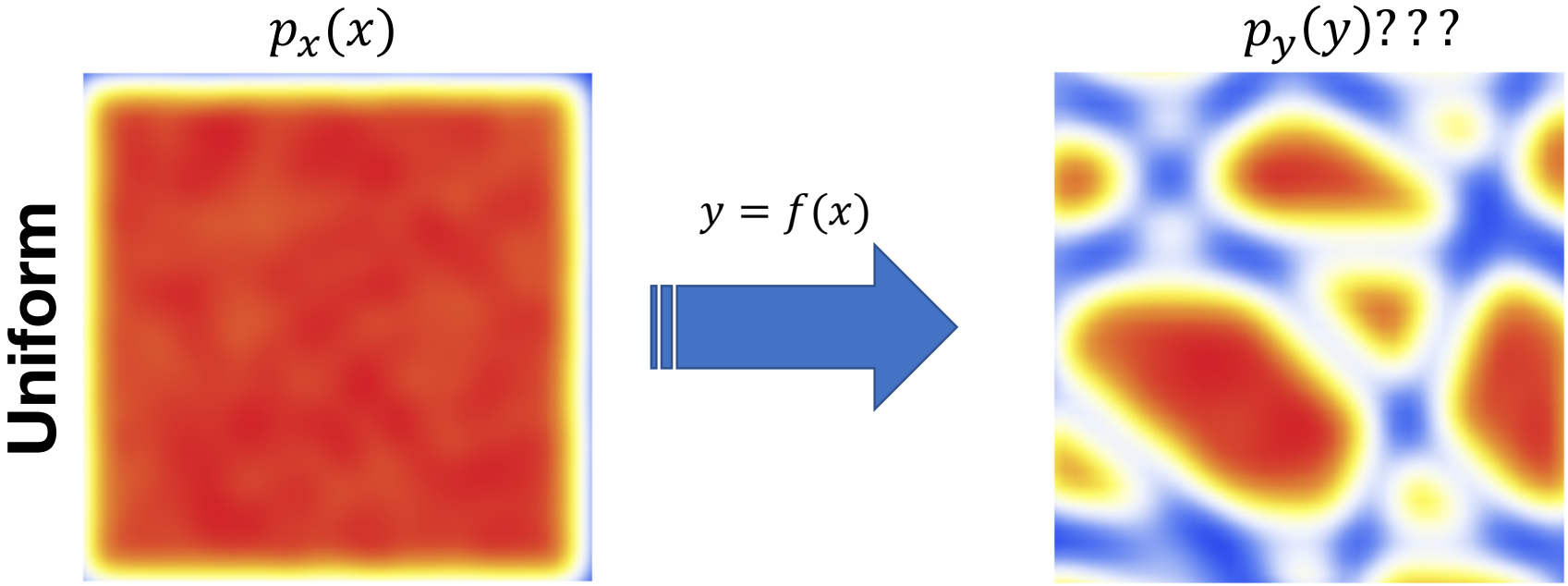
2. VAEs introduce latent variable $\mathbf{z}$ into the model; the likelihood is an intractable integration:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x} \,|\, \mathbf{z}) \mathrm{d}\, \mathbf{z}$$

3. Learning/inference has to resorted to some approximations, *e.g.*, by optimizing on a lower bound of $p_\theta(\mathbf{x})$.
4. What if we give up on explicitly modeling density, and just want ability to sample?
5. GANs do not explicitly model a density function; Instead, it learns to generate samples from the training distribution via a game-theoretic approach.

How to sample from a complex, high-dimensional training distribution?

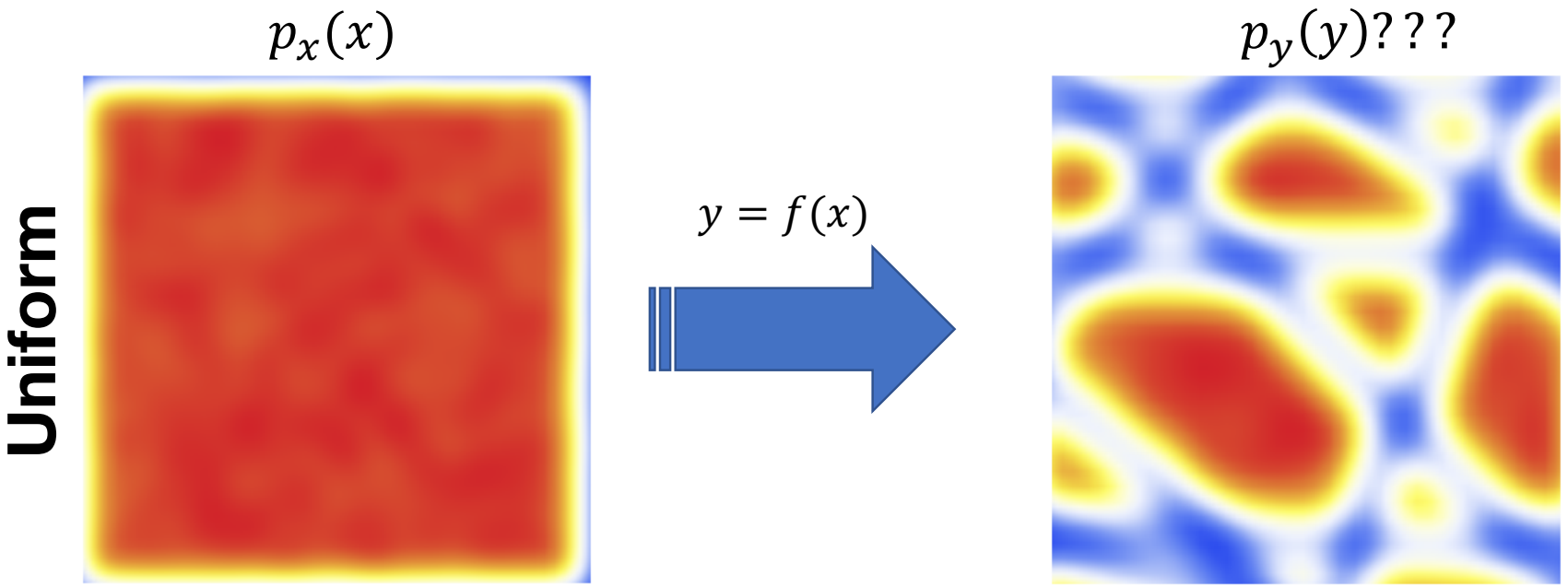
Solution: First sample from a simple distribution, *e.g.* random noise, then learn a transformation to the training distribution.



Transformation of random variables

Let $p_x(x)$ be the distribution of a random variable x, and $x = f^{-1}(y)$. Then the probability distribution of y can be written as

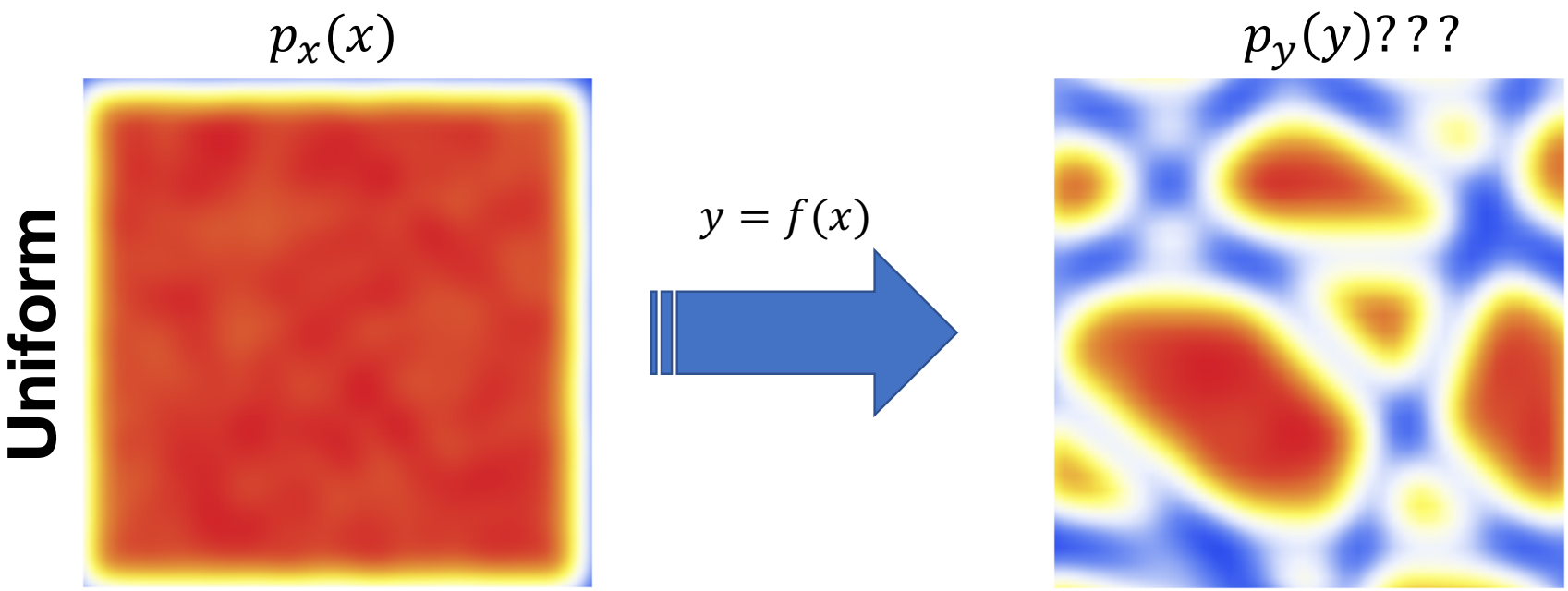
$$p_y(y) = p_x(f^{-1}(y)) \left| \frac{\mathrm{d}}{\mathrm{d}y} f^{-1}(y) \right| .$$



Generative Adversarial Networks

In generative adversarial networks (GANs), we only know the transformation $y = f(x)$, but not the inverse transformation $x = f^{-1}(y)$:

- $\frac{\mathrm{d}}{\mathrm{d}y} f^{-1}(y)$ is unknown $\Rightarrow$ $p_y(y)$ is unknown $\Rightarrow$ implicit modeling

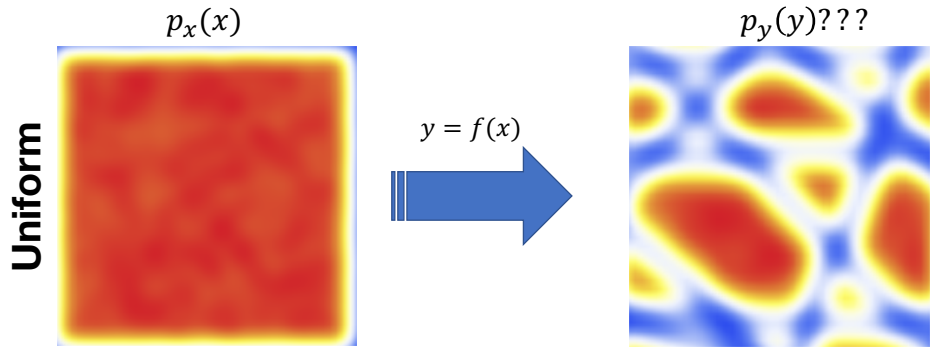


Generative Adversarial Networks

How to sample from a complex, high-dimensional training data distribution?

Solution: First sample from a simple distribution, *e.g.* random noise, then learn a transformation to the data distribution.

- How to represent this complex transformation?
- Use a neural network!



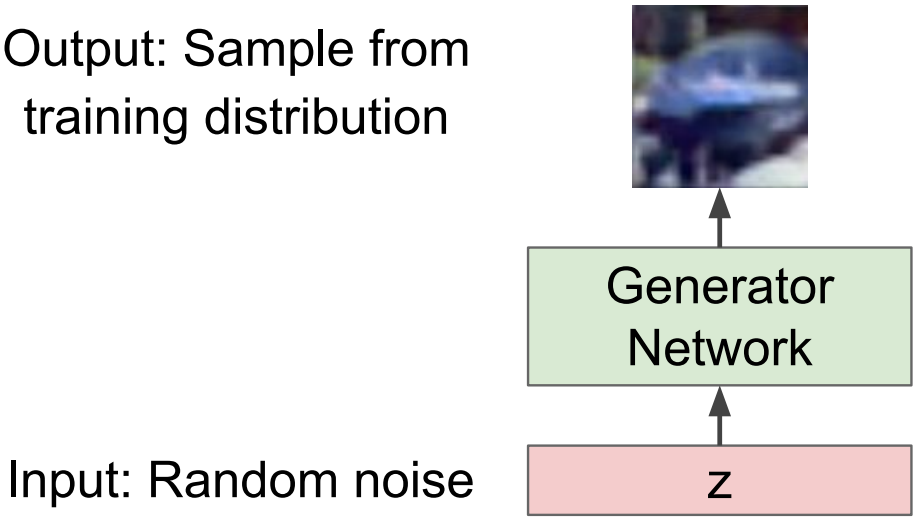
Goodfellow *et al.*, NIPS 2014

Generative Adversarial Networks

How to sample from a complex, high-dimensional training data distribution?

Solution: First sample from a simple distribution, *e.g.* random noise, then learn a transformation to the data distribution.

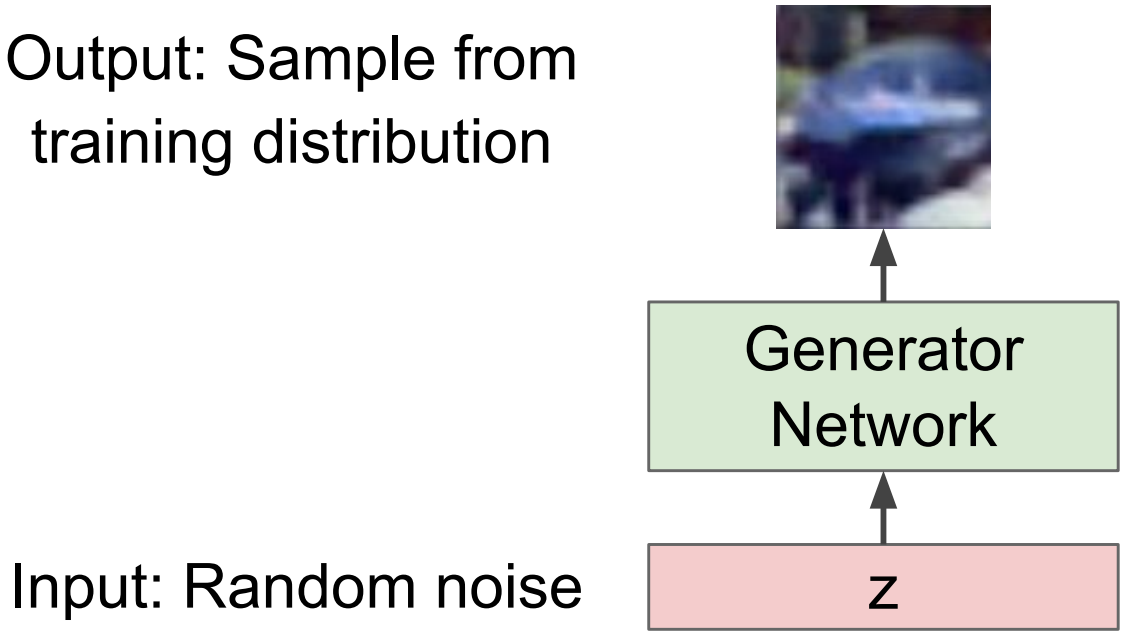
- How to represent this complex transformation?
- Use a neural network!



Goodfellow *et al.*, NIPS 2014

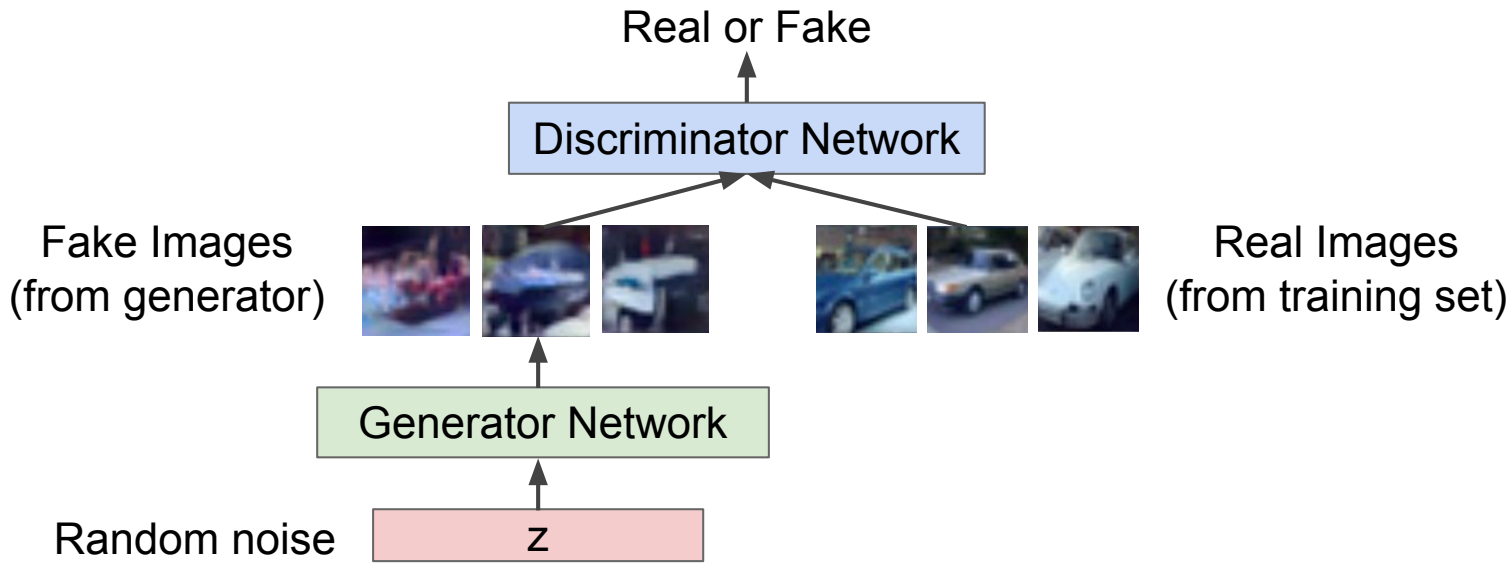
How to Do the Training?

- Since the transformation by neural network is too complicated ($f^{-1}(y)$ thus $p_y(y)$ are unknown), we don't have an explicit distribution for the output.
- Maximum likelihood unavailable.
- How to do training?



Training GANs: A Two-player Game

1. Different from the maximum likelihood principle, GAN is trained by adopting a two-player game approach from game theory.
2. There are two players, called generator and discriminator, each represented by a neural network:
   - the generator network: generate real-looking images and try to fool the discriminator, so that the discriminator can not distinguish between the images from the generator and the training data.
   - the discriminator network: try to tell apart images from the generator and training data.



Training GANs: A Two-player Game

1. Different from the maximum likelihood principle, GAN is trained by adopting a two-player game approach from game theory.
2. There are two players, called generator and discriminator, each represented by a neural network:
   - the generator network: generate real-looking images and try to fool the discriminator, so that the discriminator can not distinguish between the images from the generator and the training data.
   - the discriminator network: try to tell apart images from the generator and training data.

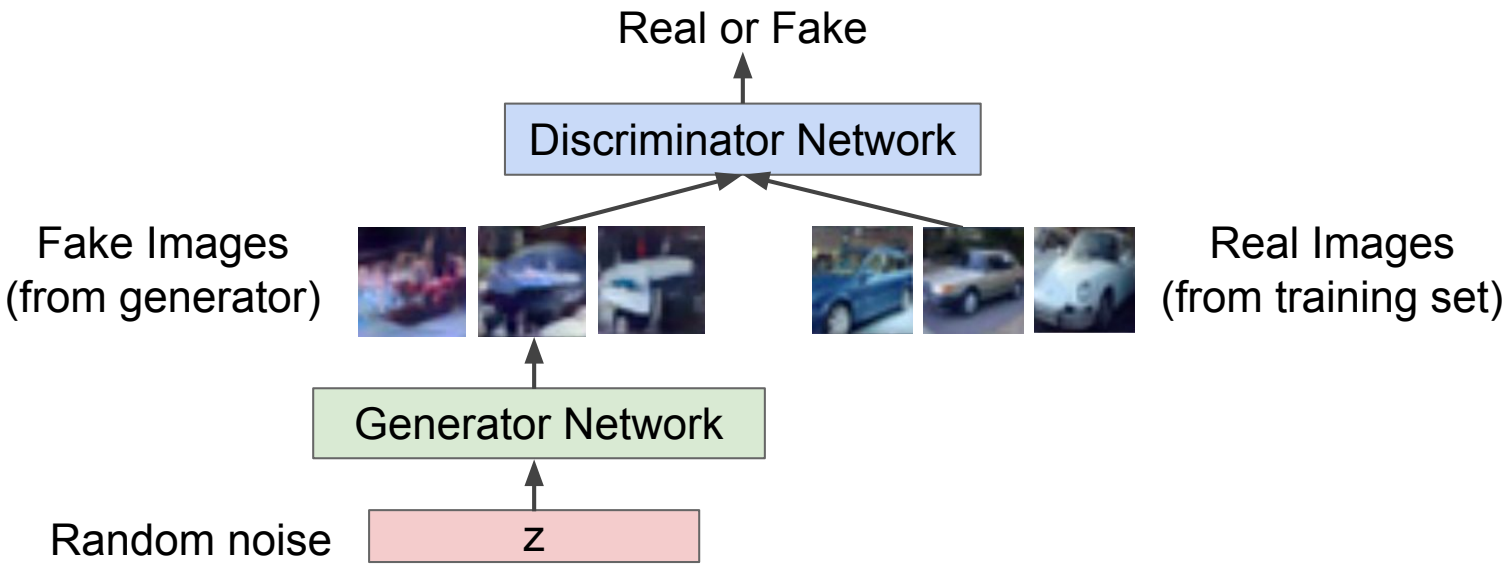
Why does this work?

Intuitively, the generator and discriminator will reach a balance such that the generator will generate images that the discriminator can discriminate half of them:

- The generated images look real!

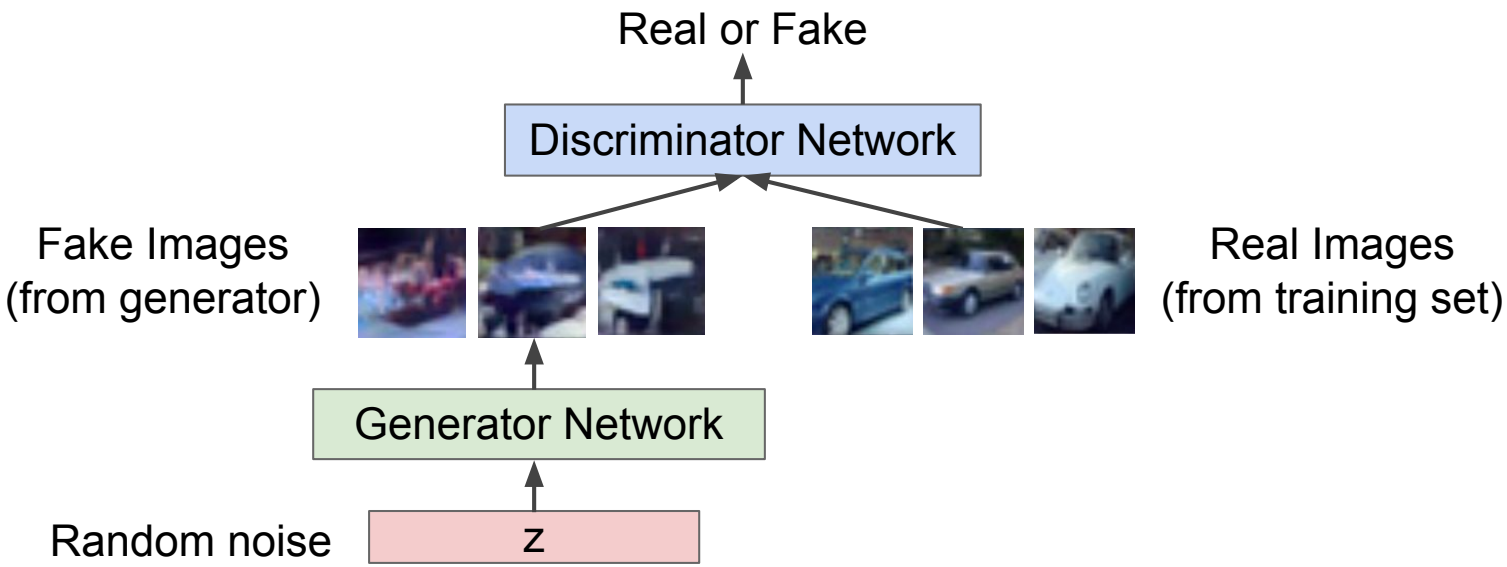
Training GANs: A Two-player Game

1. Different from the maximum likelihood principle, GAN is trained by adopting a two-player game approach from game theory.
2. There are two players, called generator and discriminator, each represented by a neural network:
   - the generator network: $\Rightarrow$ a generative model.
   - the discriminator network: $\Rightarrow$ a classifier.



Training GANs: A Two-player Game

- $D_{\theta_d}(\mathbf{x})$: Discriminator (parameterized by θ_d) takes input as an image $\mathbf{x}$, and outputs likelihood in $[0, 1]$ to tell if it is a real image or not.
- $G_{\theta_g}(\mathbf{z})$: Generator (parameterized by θ_g) takes input as a random noise $\mathbf{z}$, and outputs an image.



Training GANs: A Two-player Game

- $D_{\theta_d}(\mathbf{x})$: Discriminator (parameterized by θ_d) takes input as an image $\mathbf{x}$, and outputs likelihood in $[0, 1]$ to tell if it is a real image or not.
- $G_{\theta_g}(\mathbf{z})$: Generator (parameterized by θ_g) takes input as a random noise $\mathbf{z}$, and outputs an image.

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

Training GANs: A Two-player Game

- $D_{\theta_d}(\mathbf{x})$: Discriminator (parameterized by θ_d) takes input as an image $\mathbf{x}$, and outputs likelihood in $[0, 1]$ to tell if it is a real image or not.
- $G_{\theta_g}(\mathbf{z})$: Generator (parameterized by θ_g) takes input as a random noise $\mathbf{z}$, and outputs an image.

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

- Discriminator wants to maximize objective such that $D_{\theta_d}(\mathbf{x})$ is close to 1 (real) and $D_{\theta_d}(G_{\theta_g}(\mathbf{z}))$ is close to 0 (fake).
- Generator wants to minimize objective such that $D_{\theta_d}(G_{\theta_g}(\mathbf{z}))$ is close to 1 (discriminator is fooled into thinking generated $G_{\theta_g}(\mathbf{z})$ is real).

Training GANs: A Two-player Game

A minimax objective function:

$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\boldsymbol{\theta}_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\mathbf{z}))\right) \right]$$

Alternate between

1. Gradient ascent on discriminator:

$$\max_{\boldsymbol{\theta}_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\boldsymbol{\theta}_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\mathbf{z}))\right) \right]$$

2. Gradient descent on generator:

$$\min_{\boldsymbol{\theta}_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\mathbf{z}))\right)$$

- Balance is achieved when $D_{\theta_d}(\cdot)$ outputs 0.5 for images from both generator or training data $\Rightarrow$ discriminator can't distinguish between real and fake data $\Rightarrow$ formal derivation later.

Training GANs: A Two-player Game

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

Alternate between

1. Gradient ascent on discriminator:

$$\max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

2. Gradient descent on generator:

$$\min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

- Balance is achieved when $D_{\theta_d}(\cdot)$ outputs 0.5 for images from both generator or training data $\Rightarrow$ discriminator can't distinguish between real and fake data $\Rightarrow$ formal derivation later.

Quiz

- Design a VAE to model sequence data (continuous data). Draw the computational graph. If an RNN is used, unroll the nodes along time in the computational graph. You don't need to draw the loss node.

Training GANs: A Two-player Game

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

- Gradient descent on generator:

$$\min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - optimizing the generator is hard in practice.

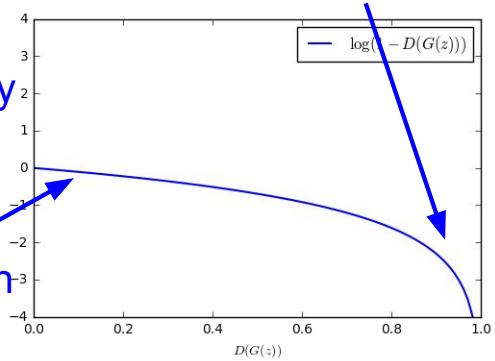
- How about changing it to:

$$\max_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - locally correct
  - works better, standard in practice

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Training GANs: A Two-player Game

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

- Gradient descent on generator:

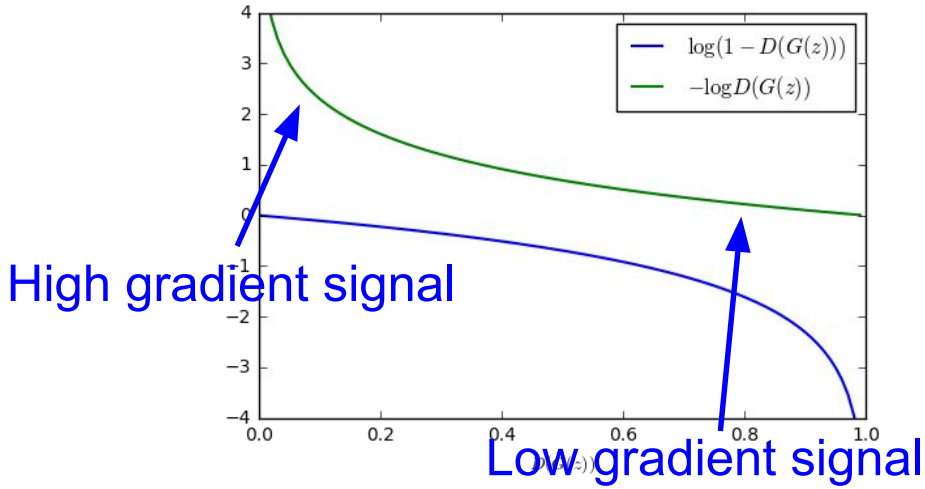
$$\min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - optimizing the generator is hard in practice.

- How about changing it to:

$$\max_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - locally correct
  - works better, standard in practice



Training GANs: A Two-player Game

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

- Gradient descent on generator:

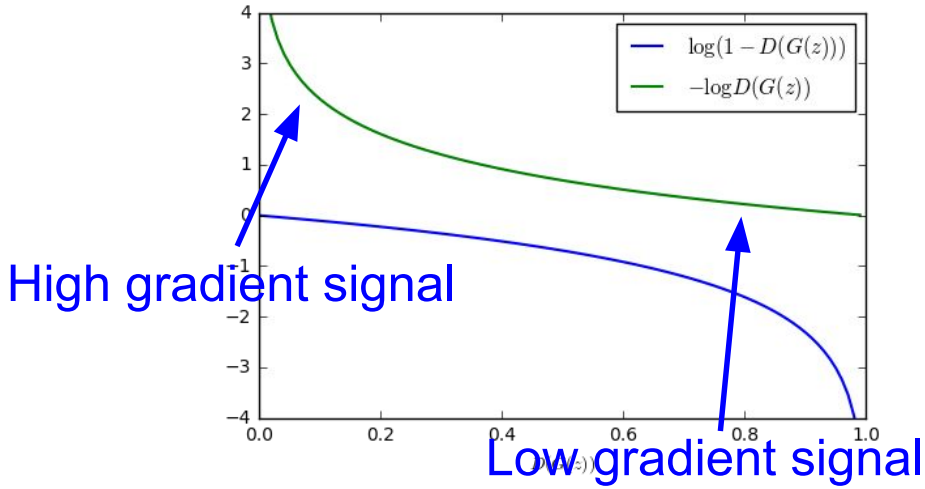
$$\min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - optimizing the generator is hard in practice.

- How about changing it to:

$$\max_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - locally correct
  - works better, standard in practice



Training GANs: A Two-Player Game

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

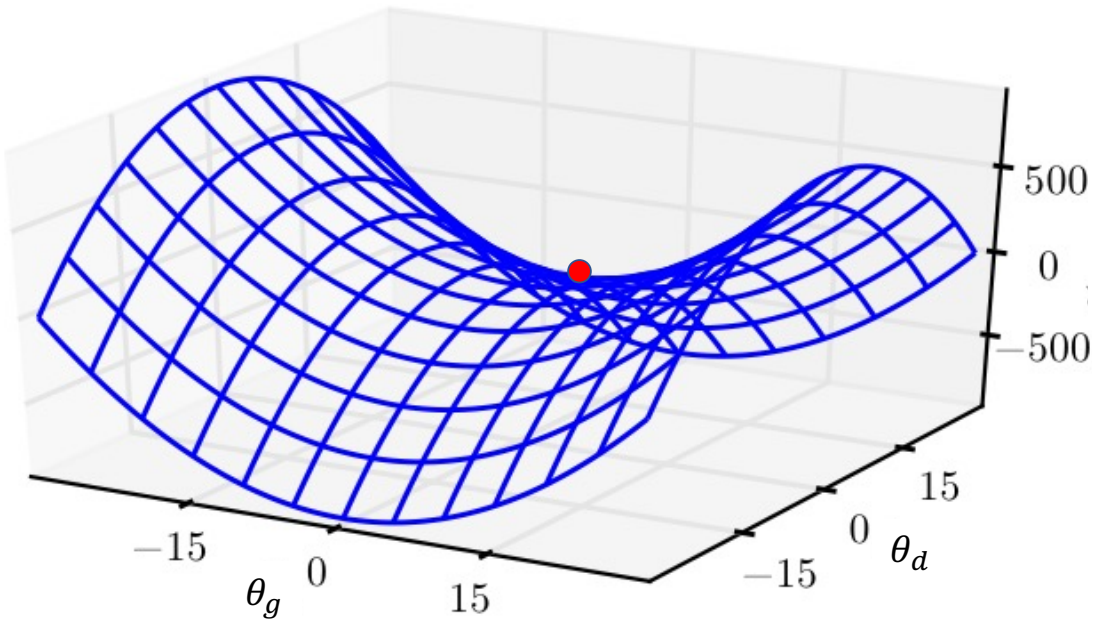
- What does an optimal solution of GAN look like?
  - A local minimum/maximum?
  - Or $\cdots$

Training GANs: A Two-Player Game

A minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

- The optimal solution for the min-max procedure is a saddle point of the GAN objective.



Training GANs: A Two-Player Game

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

Training

for number of training iterations **do**

for k steps **do**

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

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

end for

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

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

- No best rule for choosing k.

Training GANs: A Two-Player Game

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\theta_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right) \right]$$

Training

for number of training iterations **do**

for k steps **do**

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

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

end for

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

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

- No best rule for choosing k.