Deep Generative Models

Vishnu Lokhande

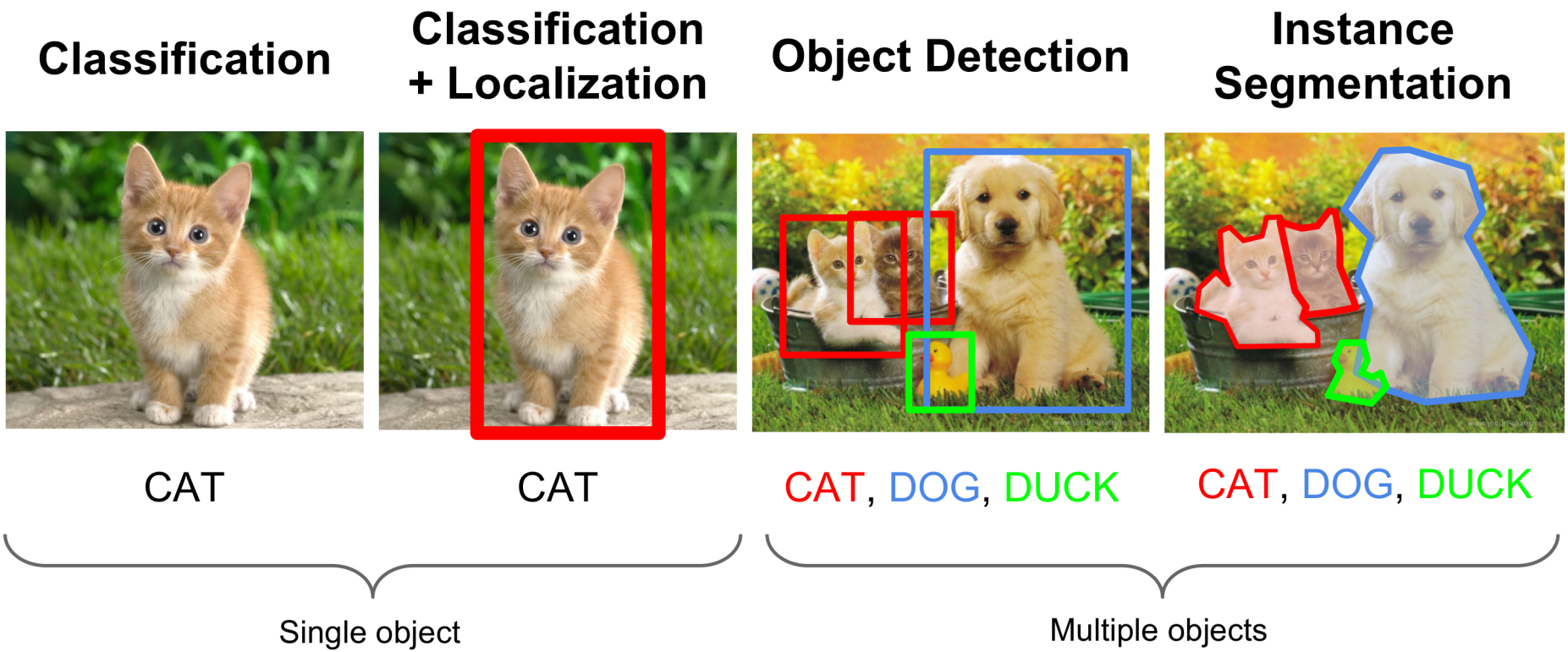
Department of Computer Science and Engineering
University at Buffalo, SUNY
vishnulo@buffalo.edu

April 2, 2025

Supervised vs. Unsupervised Learning[1]

Supervised Learning

- **Data** $(\mathbf{x}, y)$: $\mathbf{x}$ is data, y is label/output.
- **Goal**: Learn a function to map $\mathbf{x} \rightarrow y$.



[1] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Supervised vs. Unsupervised Learning[1]

Supervised Learning

- **Data** $(\mathbf{x}, y)$: $\mathbf{x}$ is data, y is label/output.
- **Goal**: Learn a function to map $\mathbf{x} \rightarrow y$.

1. A park with a clock tower in the background
2. A place with a tall building in the background
3. A park with a tall tree in the middle

[1] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.

Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.

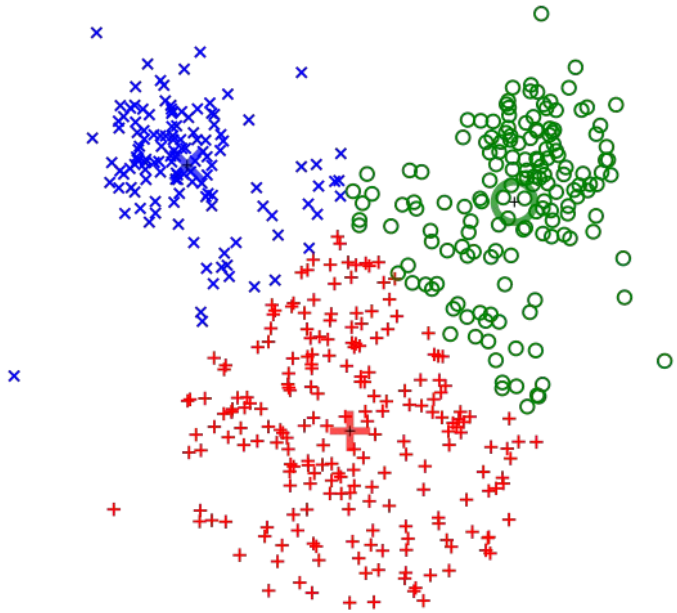


Figure: Kmeans

Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.

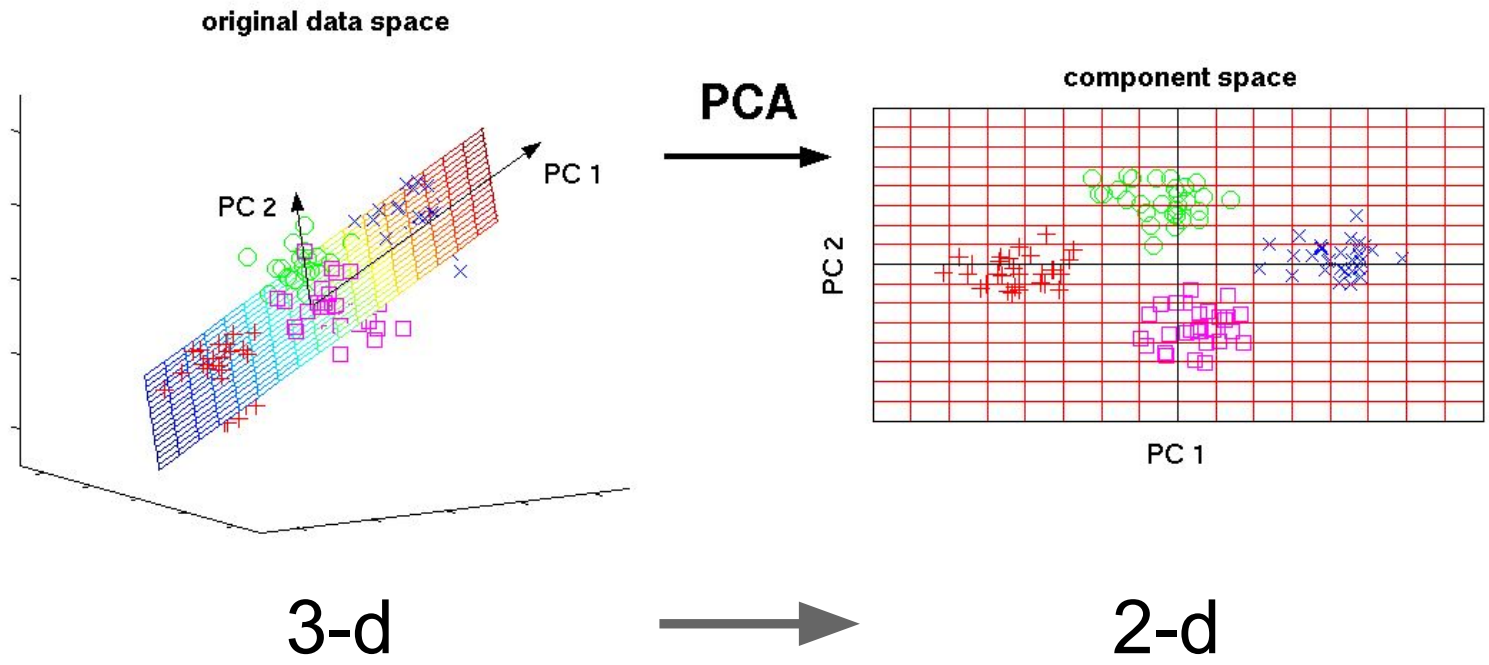


Figure: Principal component analysis (dimension reduction)

Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.

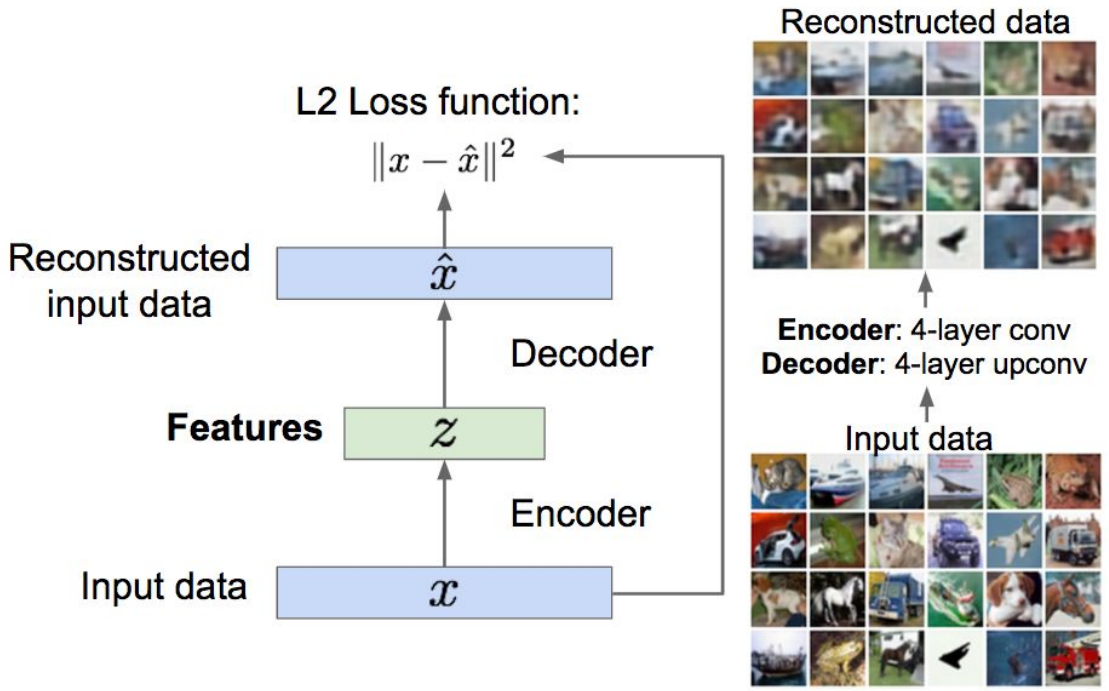
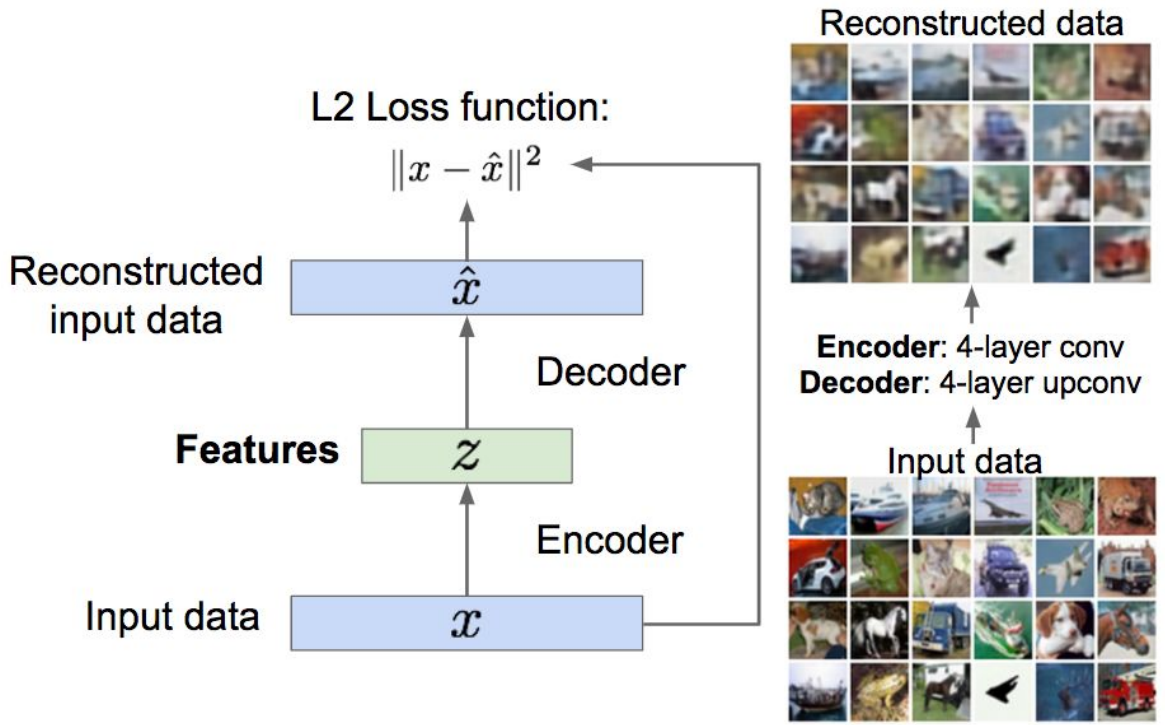


Figure: Autoencoders (feature learning)

Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.



$\Rightarrow$ Variational autoencoder (VAE)

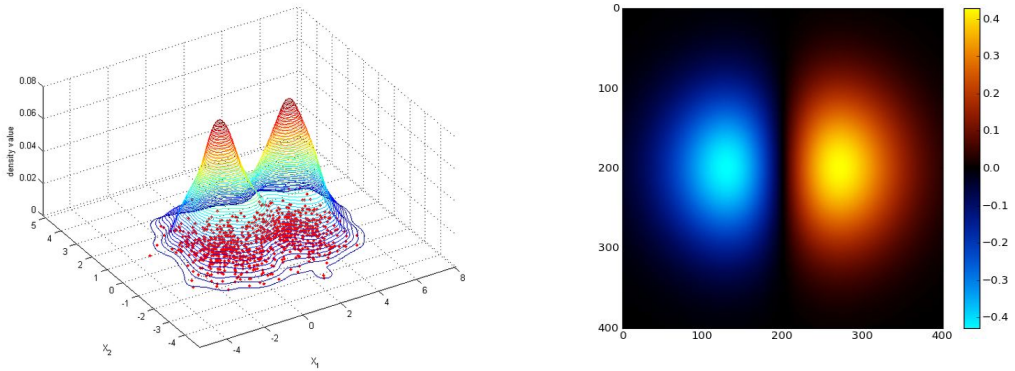
Figure: Autoencoders (feature learning)

Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.

1-d density estimation



2-d density estimation

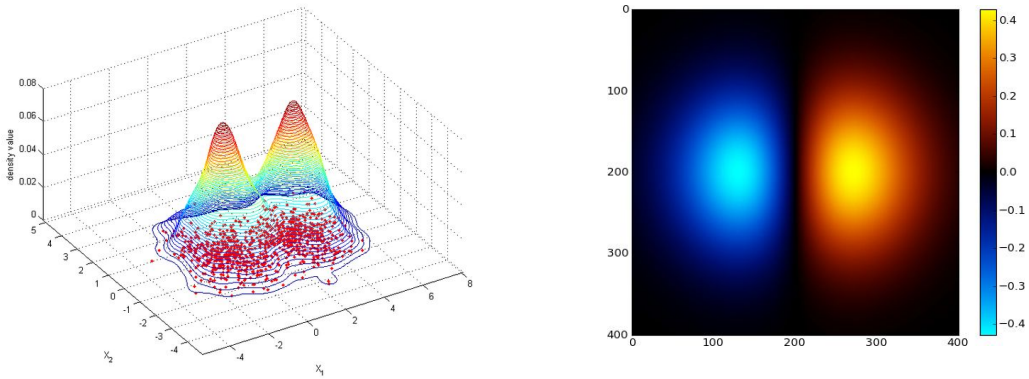
Supervised vs. Unsupervised Learning

Unsupervised Learning

- **Data x**: no labels.
- **Goal**: Learn some underlying hidden structure of the data:
  - Define different objective functions for different models.

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

$\Rightarrow$

Generative adversarial networks (GAN)

Generative Models

Generate new samples from the same data distribution.

training data $\sim p_{\text{data}}(\mathbf{x})$

generated samples $\sim p_{\text{model}}(\mathbf{x})$

- Goal is to learn $p_{\text{model}}(\mathbf{x})$ such that it is close to $p_{\text{data}}(\mathbf{x})$.

Generative Models

Generate new samples from the same data distribution.

training data $\sim p_{\text{data}}(\mathbf{x})$

generated samples $\sim p_{\text{model}}(\mathbf{x})$

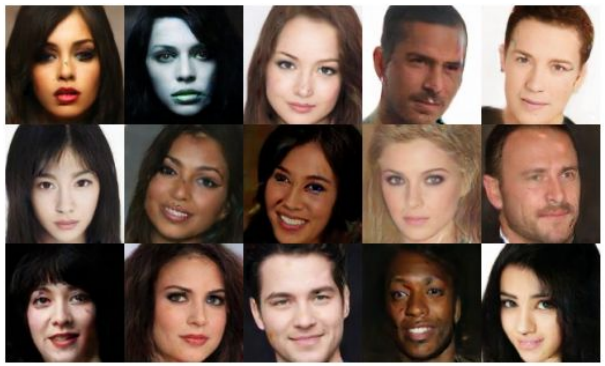
- Goal is to learn $p_{\text{model}}(\mathbf{x})$ such that it is close to $p_{\text{data}}(\mathbf{x})$.

Several flavors

- Explicit density estimation: explicitly define and solve for $p_{\text{model}}(\mathbf{x})$:
  - *e.g.*, variational autoencoder (VAE) (in this case, $\mathbf{x}$ refers to the latent representation of the data)
- Implicit density estimation: learn a model that can sample from $p_{\text{model}}(\mathbf{x})$ w/o explicitly defining it:
  - *e.g.*, generative adversarial nets (GAN)

Why Generative Models?

1. Realistic samples for artwork, super-resolution, colorization *etc*.


2. Generative models of time-series data can be used for simulation and planning $\Rightarrow$ reinforcement learning applications.
3. Training generative models can also enable inference of latent representations that can be useful as general features.
4. Effective use of unlabeled data.

Taxonomy of Generative Models

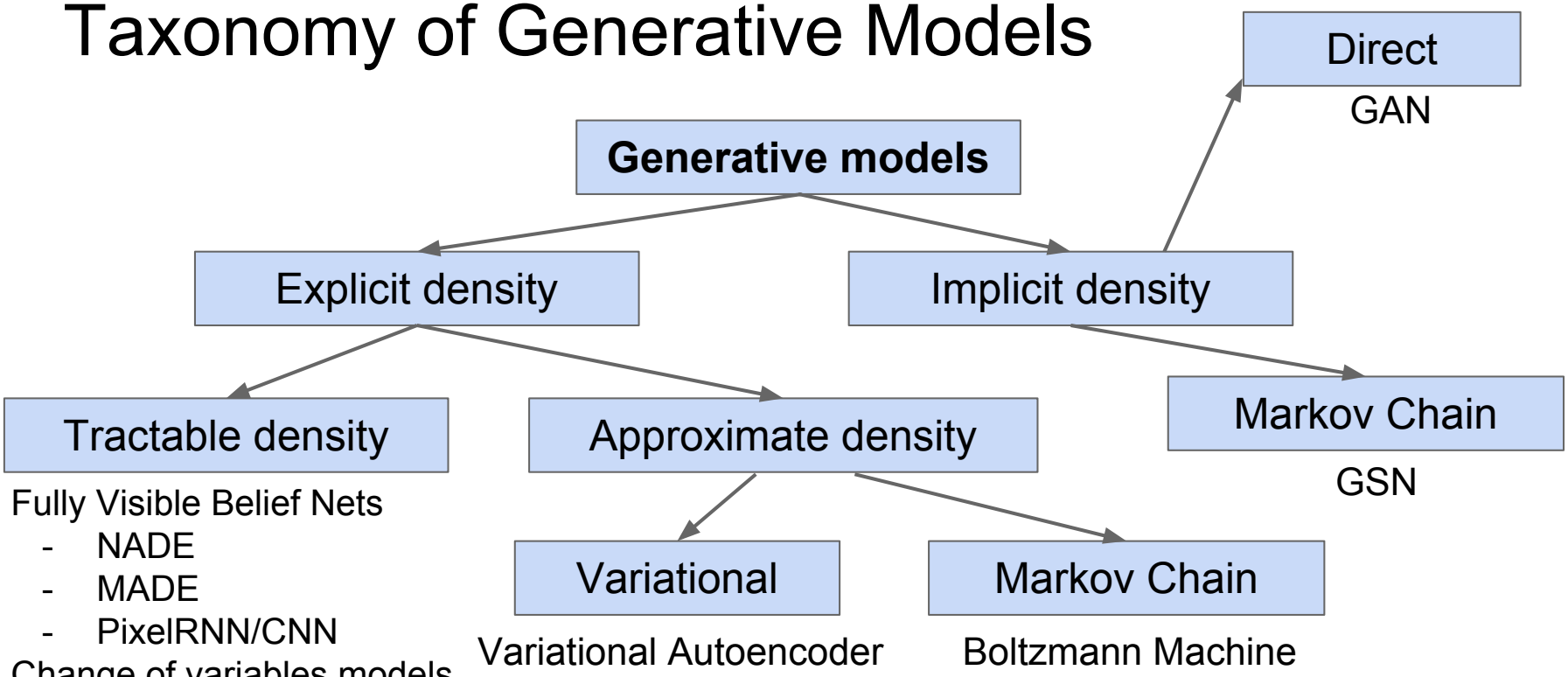


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

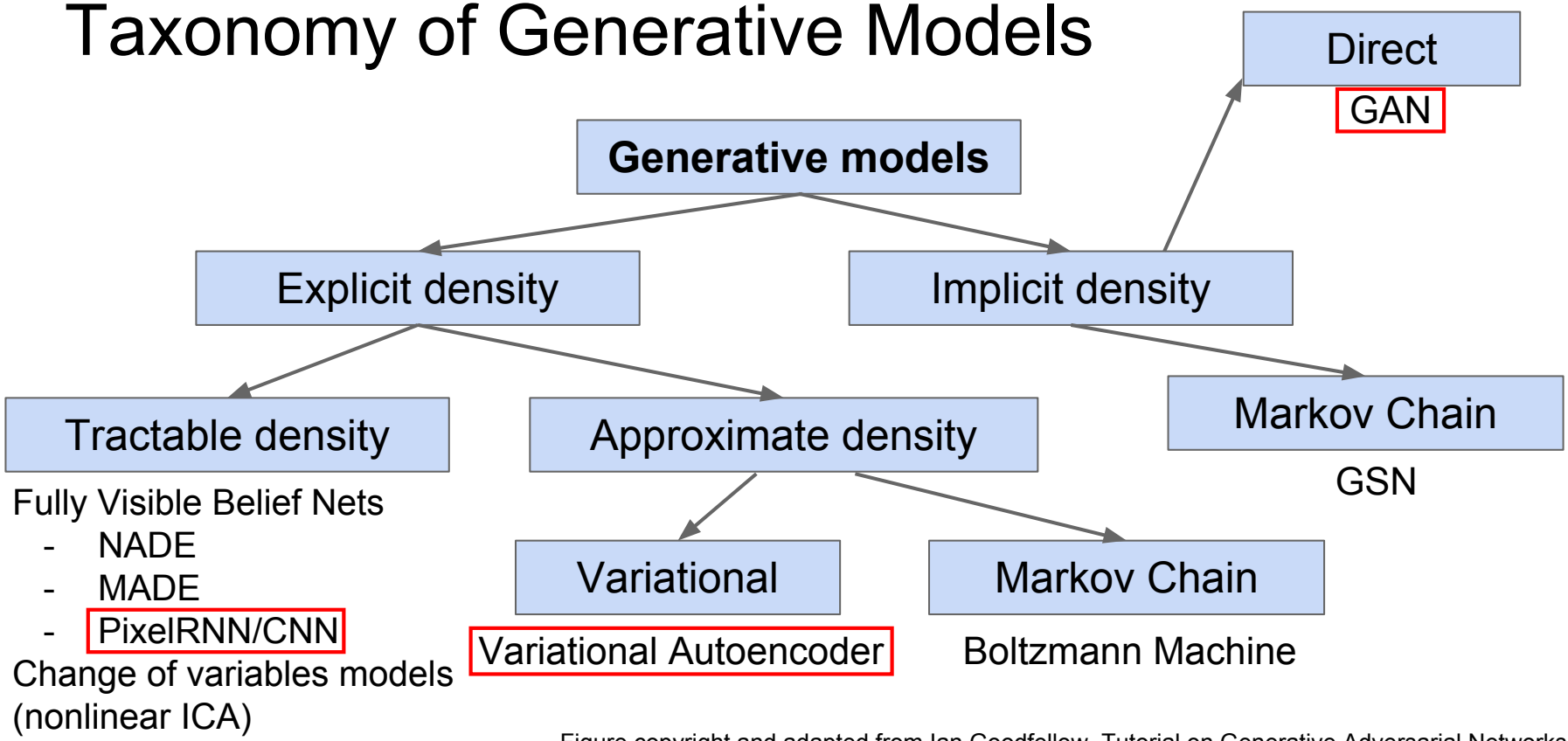


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Autoreressive Image Modeling: PixelRNN/PixelCNN

Fully Visible Belief Network

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - we will use RNN and CNN.
- How to define the order of $\{x_i\}$?
  - rely on specific problems.

Fully Visible Belief Network

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - we will use RNN and CNN.
- How to define the order of $\{x_i\}$?
  - rely on specific problems.

Fully Visible Belief Network

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - we will use RNN and CNN.
- How to define the order of $\{x_i\}$?
  - rely on specific problems.

Fully Visible Belief Network

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - we will use RNN and CNN.
- How to define the order of $\{x_i\}$?
  - rely on specific problems.

Fully Visible Belief Network

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - we will use RNN and CNN.
- How to define the order of $\{x_i\}$?
  - rely on specific problems.

Fully Visible Belief Network

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

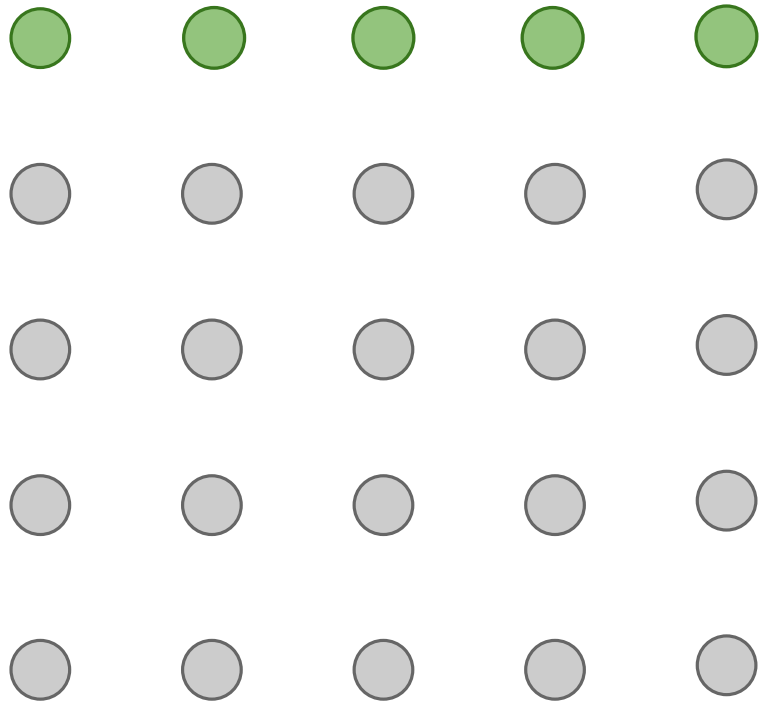
Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - we will use RNN and CNN.
- How to define the order of $\{x_i\}$?
  - rely on specific problems.

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

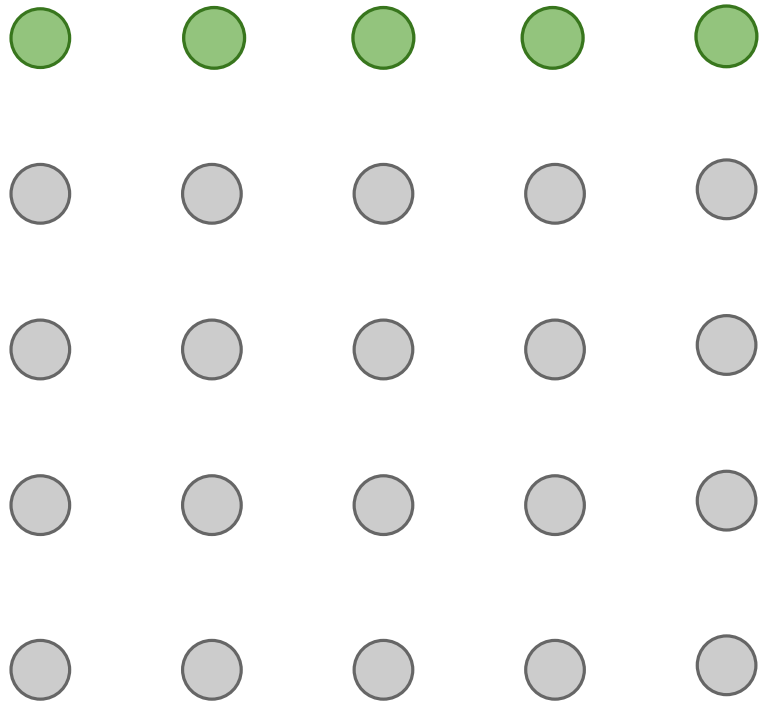


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

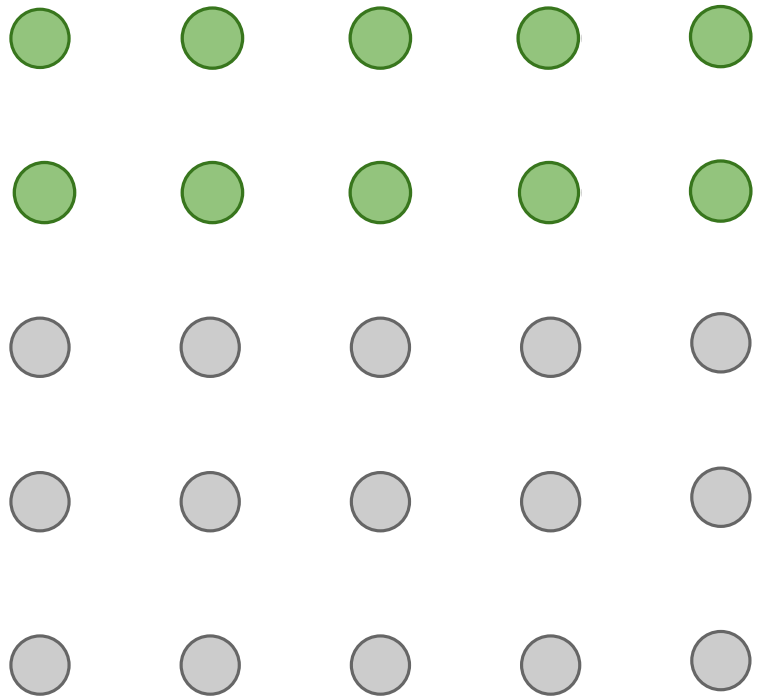


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

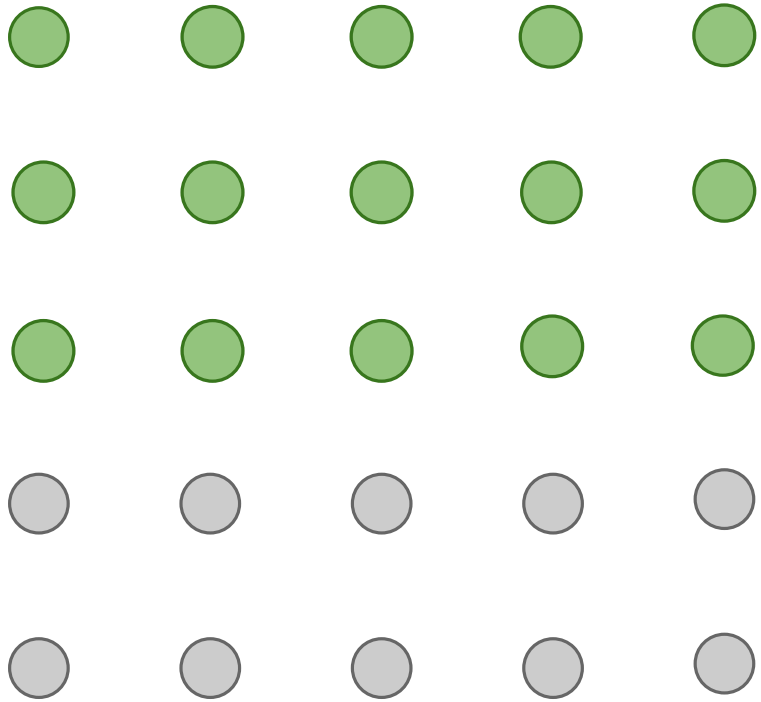


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

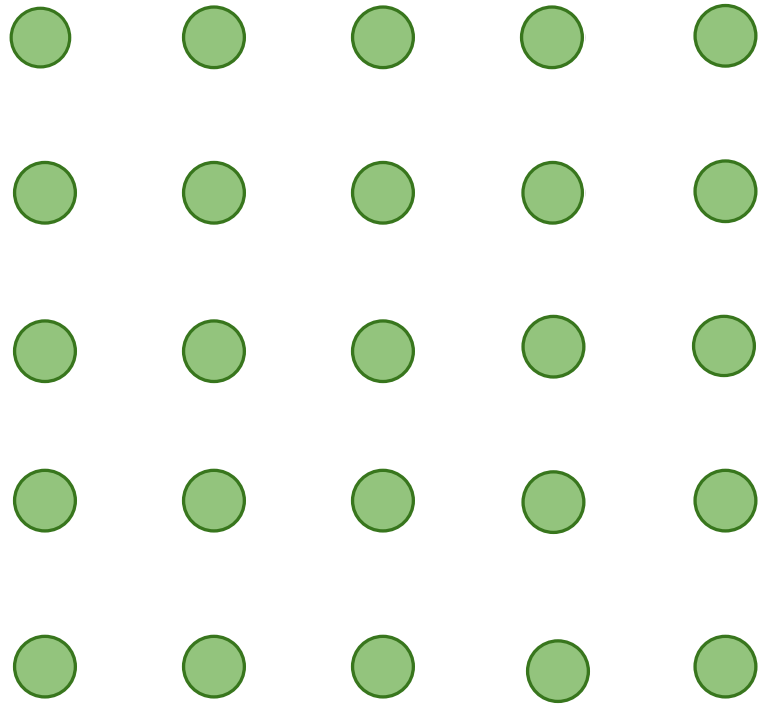


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

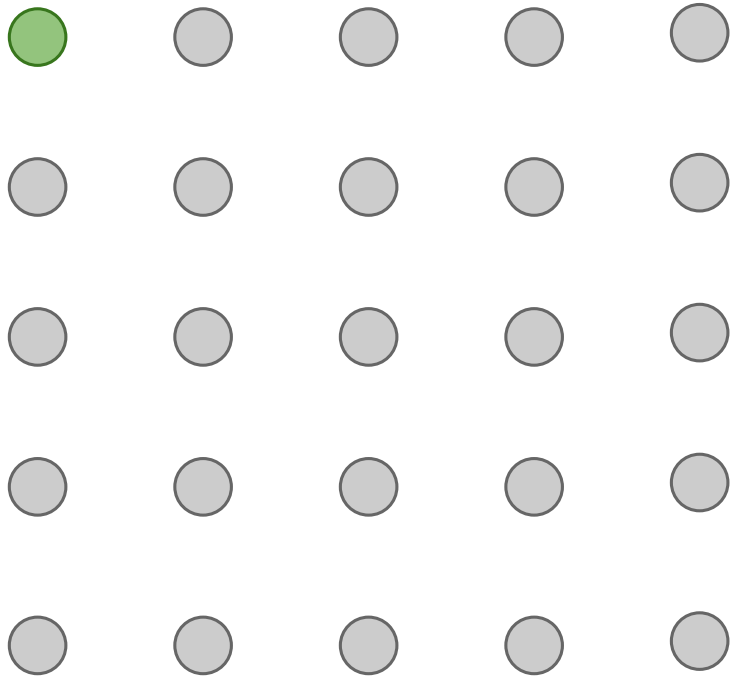


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

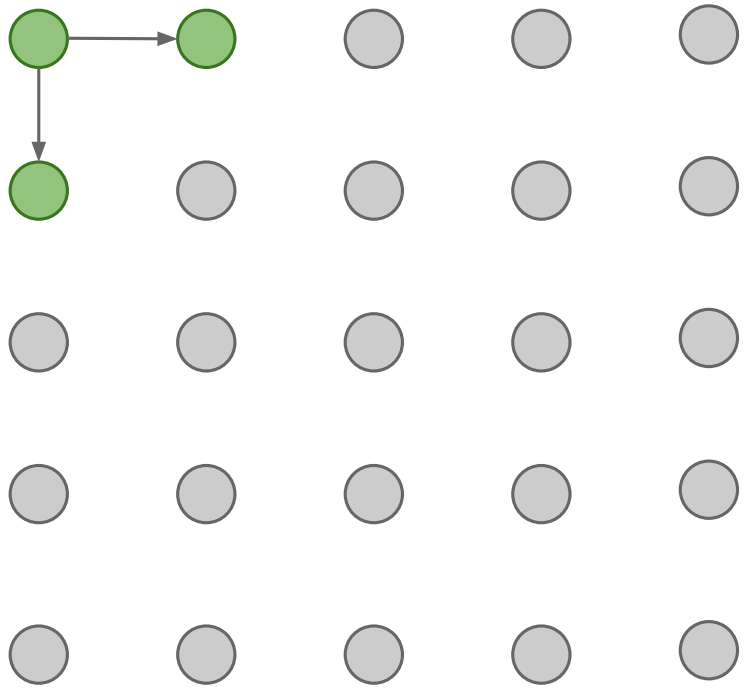


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

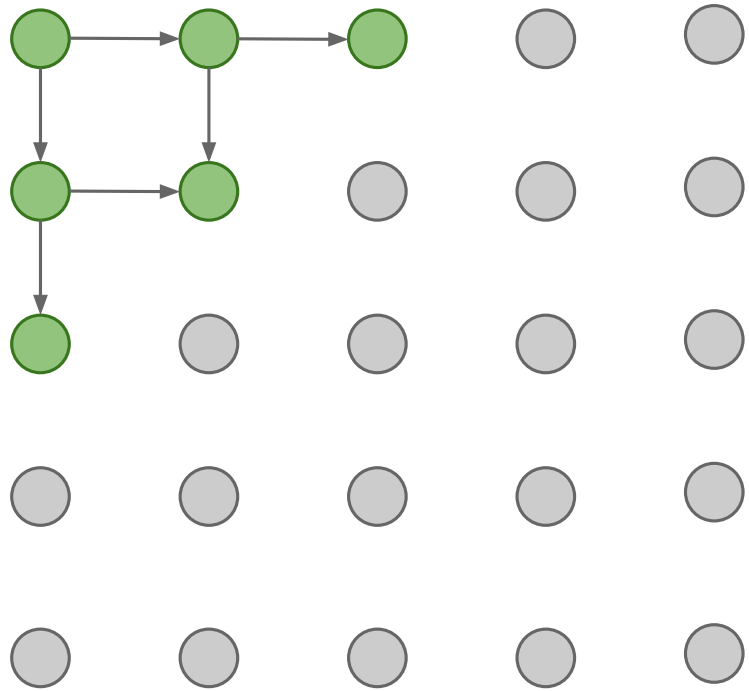


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.

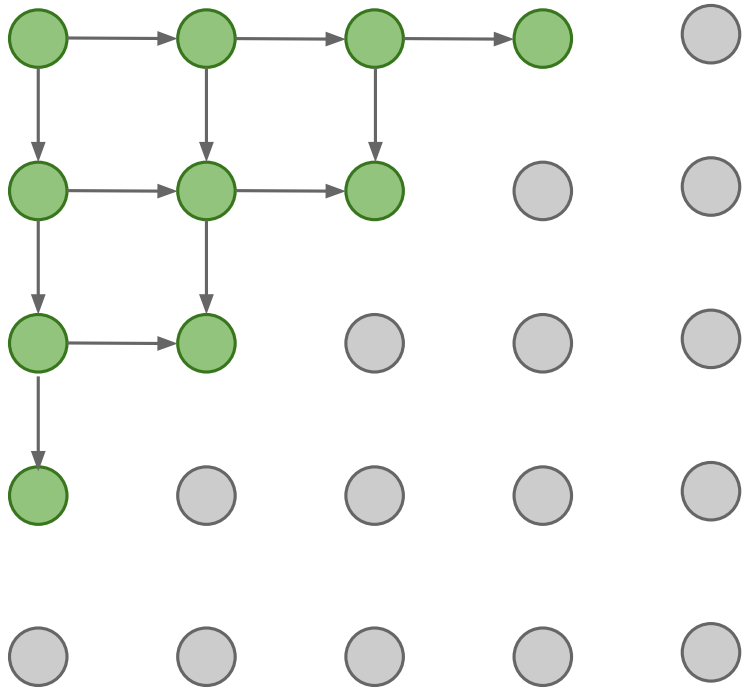


van der Oord *et al.* 2016

Pixel RNN

Let's now consider modeling an image.

1. Dependency on previous pixels modeled using an RNN (LSTM).
2. Implemented via Row LSTM and Diagonal BiLSTM.
3. For Row LSTM, generate image pixels row by row.
4. For Diagonal BiLSTM, generate image pixels starting from corner.



van der Oord *et al.* 2016

Convolutional LSTM

LSTM Equations

$$i = \sigma(x_i U^i + \mathbf{h}_{i-1} W^i)$$
$$f = \sigma(x_i U^f + \mathbf{h}_{i-1} W^f)$$
$$o = \sigma(x_i U^o + \mathbf{h}_{i-1} W^o)$$
$$g = tanh(x_i U^g + \mathbf{h}_{i-1} W^g)$$

Gates - Control how much information is allowed through

$$c_i = c_{i-1} \circ f + g \circ i$$
$$\mathbf{h}_i = \tanh(c_i) \circ o$$

States - Hold information about all time steps up till now {0, i, ..., i-1, i}

LSTM Equations

$$i = \sigma(x_i U^i + \mathbf{h}_{i-1} W^i)$$
$$f = \sigma(x_i U^f + \mathbf{h}_{i-1} W^f)$$
$$o = \sigma(x_i U^o + \mathbf{h}_{i-1} W^o)$$
$$g = tanh(x_i U^g + \mathbf{h}_{i-1} W^g)$$

Like Convolutional LSTM - replaced fully-connected layer with convolutional layer

$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma(\mathbf{K}^{ss} \circledast \mathbf{h}_{i-1} + \mathbf{K}^{is} \circledast \mathbf{x}_i)$$

$$c_i = c_{i-1} \circ f + g \circ i$$
$$\mathbf{h}_i = \tanh(c_i) \circ o$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

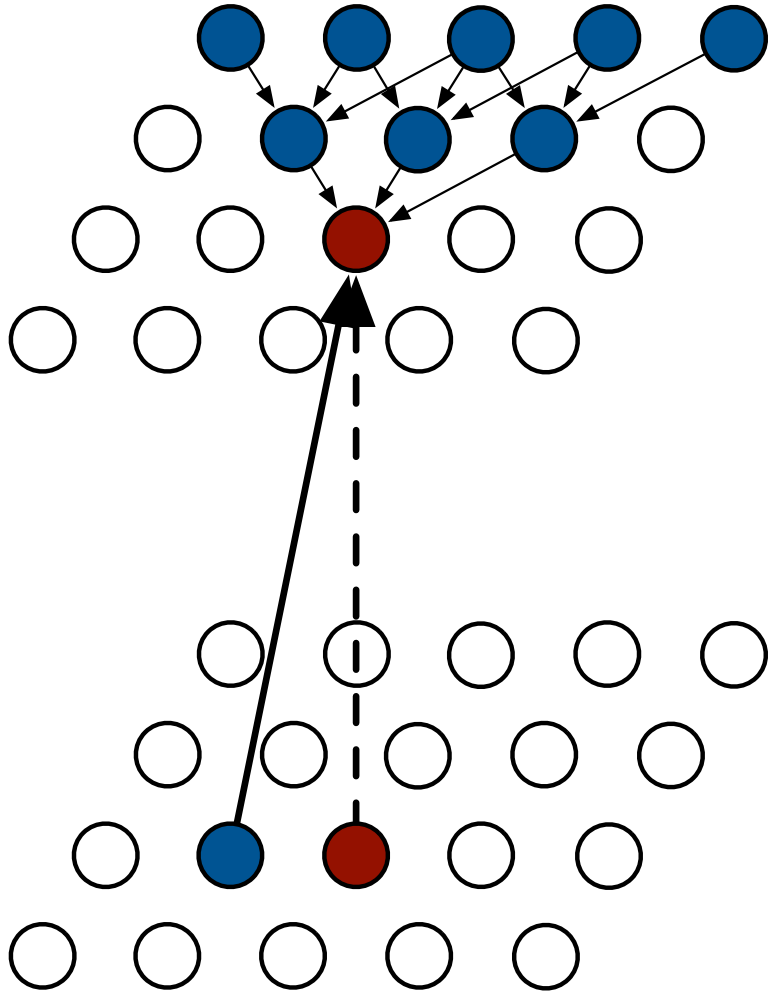
Row LSTM

- $\mathbf{h}_i$: hidden representation of the i-th row – $h \times n$
- $\mathbf{x}_i$: input map of the i-th row, obtained by doing 1-D convolution on the original image – $h \times n$
- The figure shows the case of $h = 1$

$$[\mathbf{o}_i^j, \mathbf{f}_i^j, \mathbf{i}_i^j, \mathbf{g}_i^j] = \sigma\left(\mathbf{K}^{ss} * \mathbf{h}_{i-1}^{j-1:j+1} + \mathbf{K}^{is} * \mathbf{x}_i^{j-1:j}\right)$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$

$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$



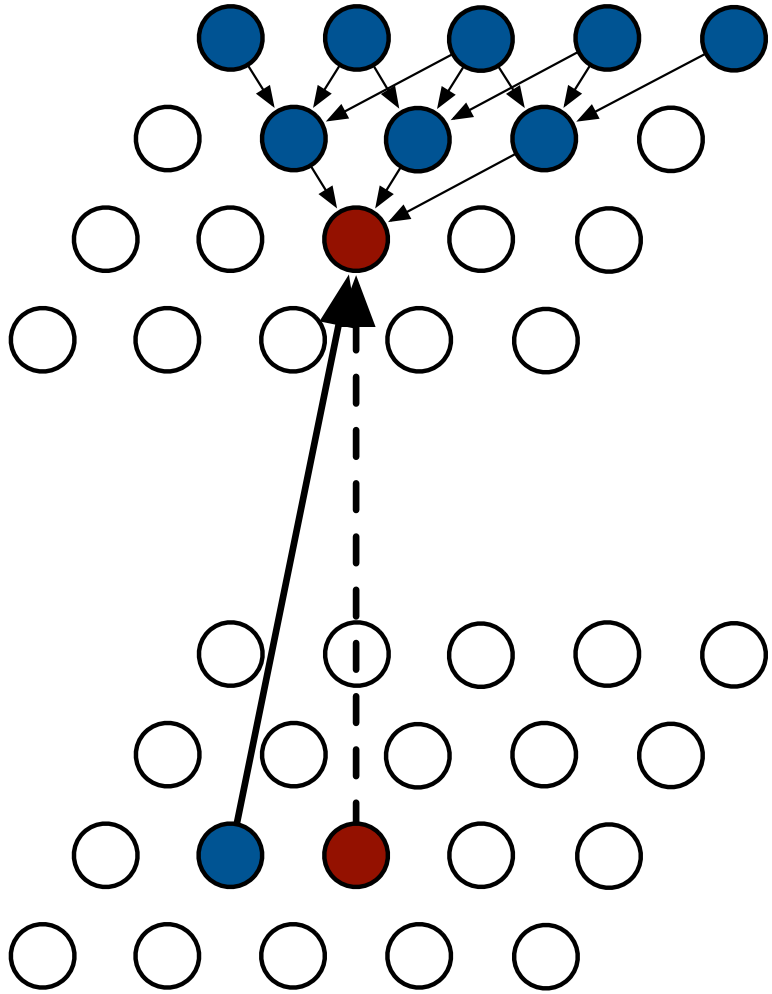
Row LSTM

$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma\left(\mathbf{K}^{ss} * \mathbf{h}_{i-1} + \mathbf{K}^{is} * \mathbf{x}_i\right)$$

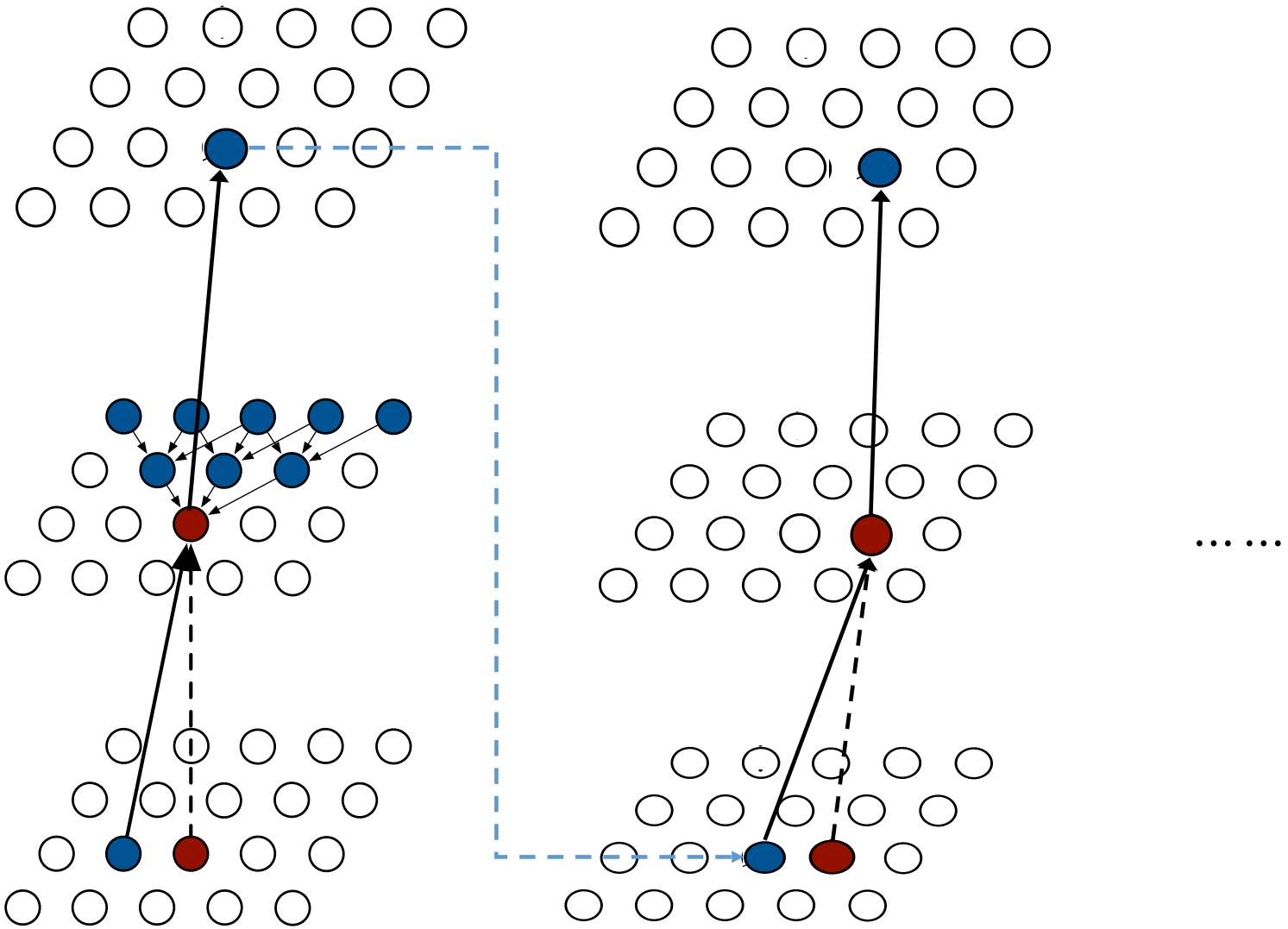
$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$

$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

- Has a triangular receipt filed, unable to capture the entire available context.

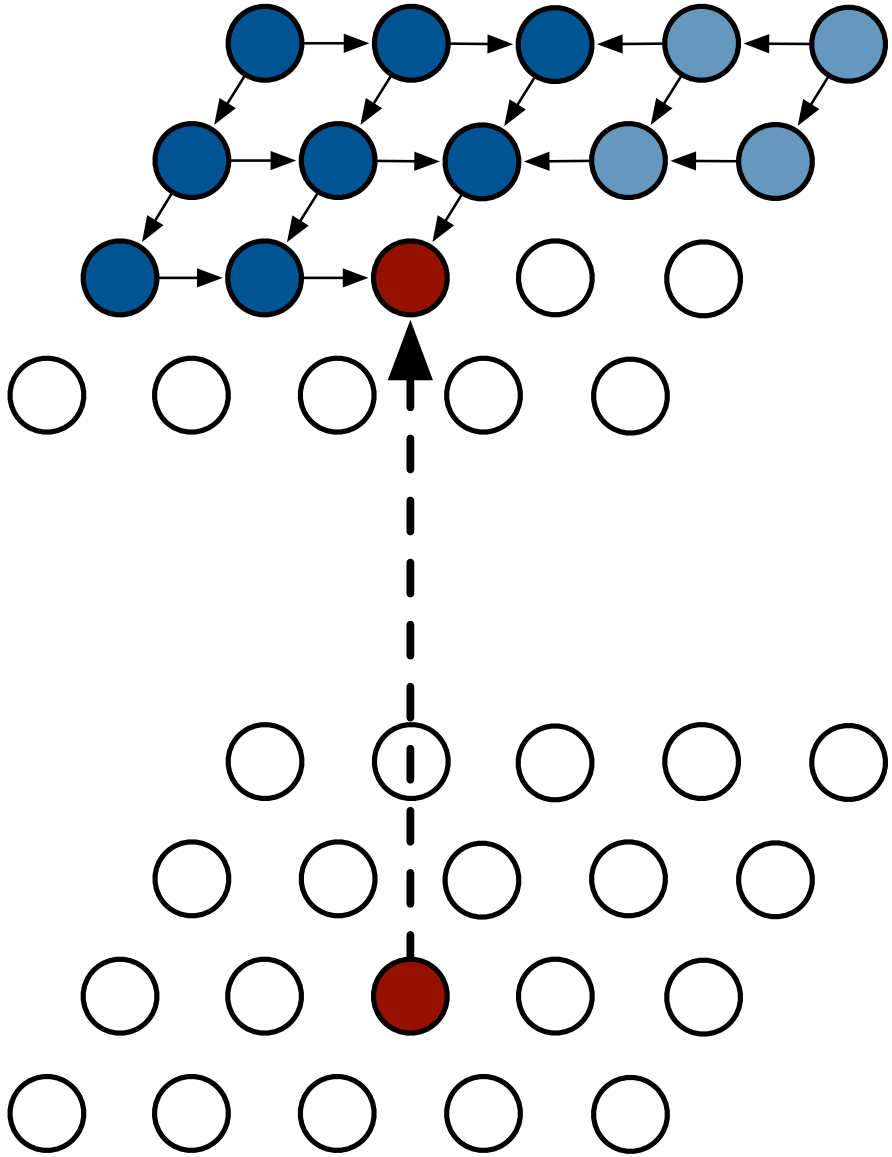


Row LSTM: Generation



Diagonal BiLSTM

- Be able to reach the whole dependent receipt field.



Pixel CNN

1. Still generate image pixels starting from corner.
2. Dependency on previous pixels now modeled using a CNN over context region.

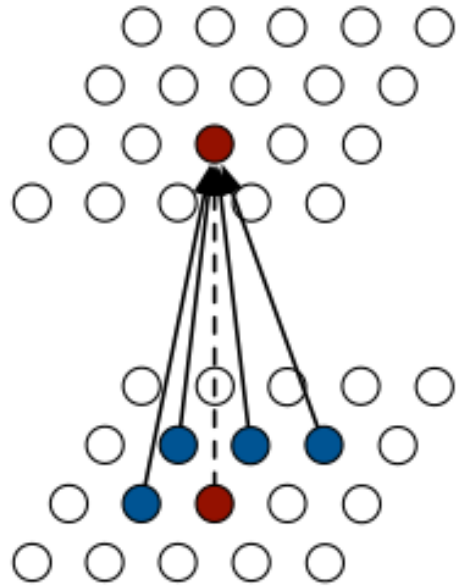
$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma\left(\mathbf{K}^{ss} * \mathbf{h}_{i-1} + \mathbf{K}^{is} * \mathbf{x}_i\right)$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$

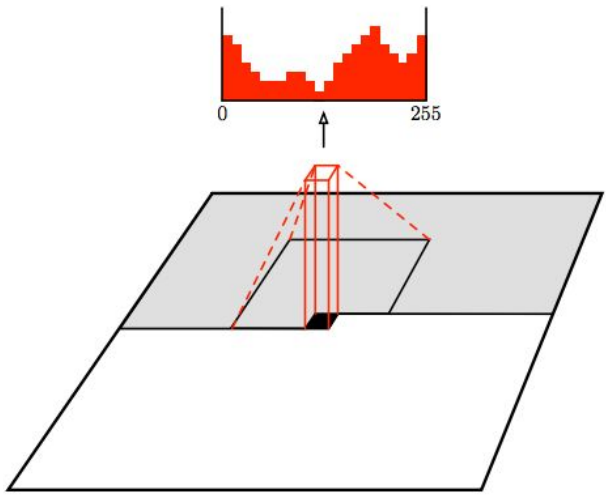
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

$\Rightarrow$ CNN for $\mathbf{h}_i$

3. Training is faster than PixelRNN
   - can parallelize convolutions since context region values known from training images.
4. Generation must still proceed sequentially, thus slow.



PixelCNN



van der Oord *et al.* 2016

Pixel CNN

1. Still generate image pixels starting from corner.
2. Dependency on previous pixels now modeled using a CNN over context region.

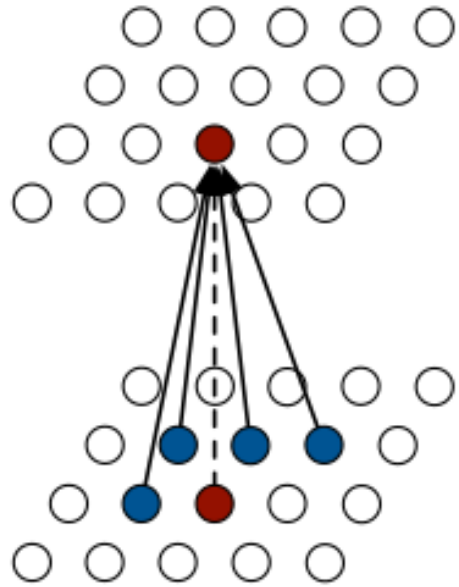
$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma\left(\mathbf{K}^{ss} * \mathbf{h}_{i-1} + \mathbf{K}^{is} * \mathbf{x}_i\right)$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$

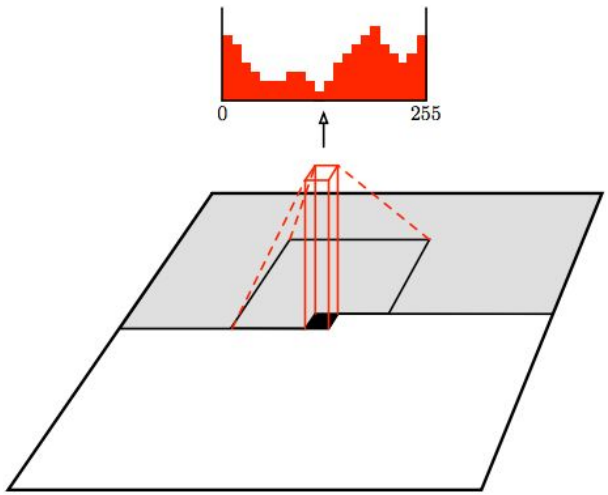
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

$\Rightarrow$ CNN for $\mathbf{h}_i$

3. Training is faster than PixelRNN
   - can parallelize convolutions since context region values known from training images.
4. Generation must still proceed sequentially, thus slow.



PixelCNN



van der Oord *et al.* 2016

Pixel CNN

1. Still generate image pixels starting from corner.
2. Dependency on previous pixels now modeled using a CNN over context region.

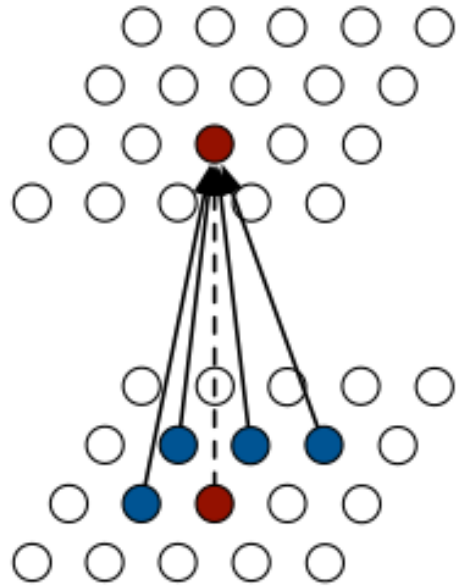
$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma\left(\mathbf{K}^{ss} * \mathbf{h}_{i-1} + \mathbf{K}^{is} * \mathbf{x}_i\right)$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$

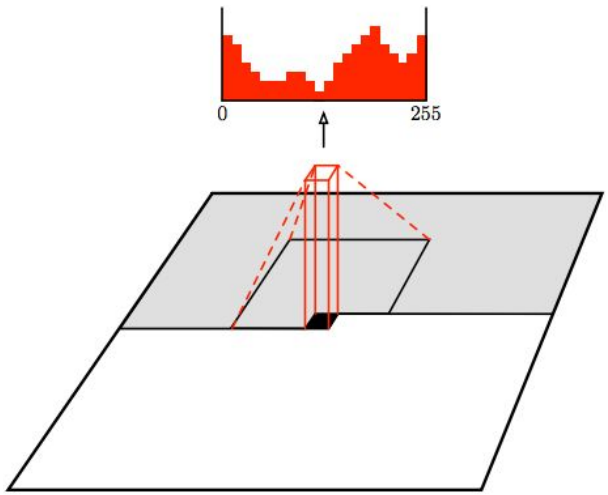
$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

$\Rightarrow$ CNN for $\mathbf{h}_i$

3. Training is faster than PixelRNN
   - can parallelize convolutions since context region values known from training images.
4. Generation must still proceed sequentially, thus slow.



PixelCNN



van der Oord *et al.* 2016

Pixel CNN

1. Still generate image pixels starting from corner.
2. Dependency on previous pixels now modeled using a CNN over context region.

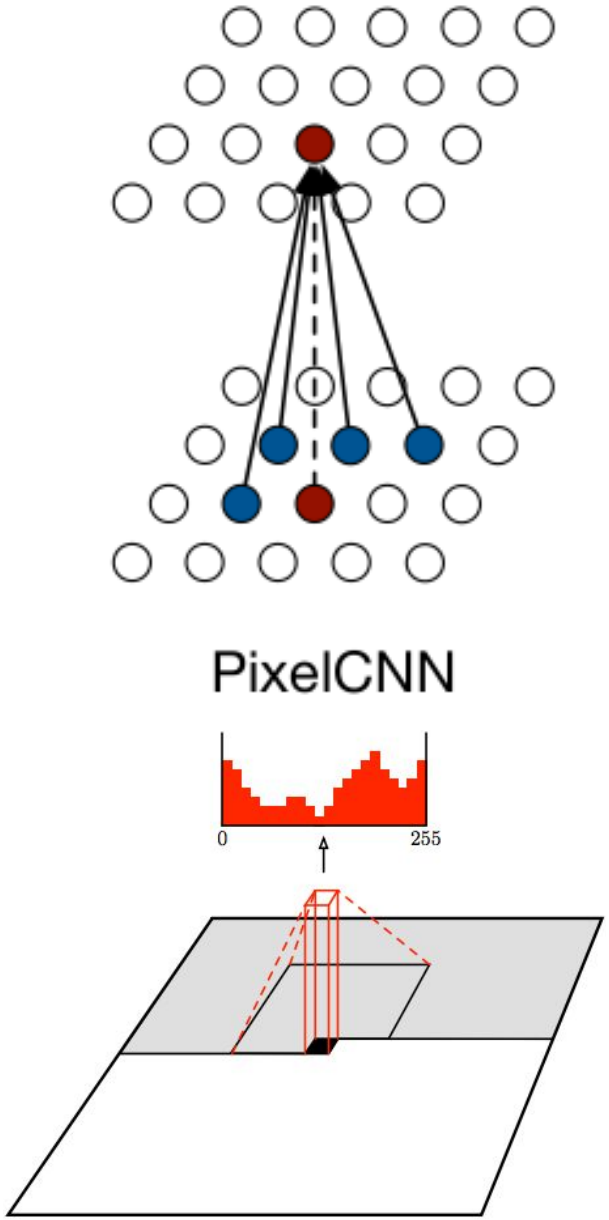
$$[\mathbf{o}_i, \mathbf{f}_i, \mathbf{i}_i, \mathbf{g}_i] = \sigma\left(\mathbf{K}^{ss} * \mathbf{h}_{i-1} + \mathbf{K}^{is} * \mathbf{x}_i\right)$$

$$\mathbf{c}_i = \mathbf{f}_i \odot \mathbf{c}_{i-1} + \mathbf{i}_i \odot \mathbf{g}_i$$

$$\mathbf{h}_i = \mathbf{o}_i \odot \tanh(\mathbf{c}_i)$$

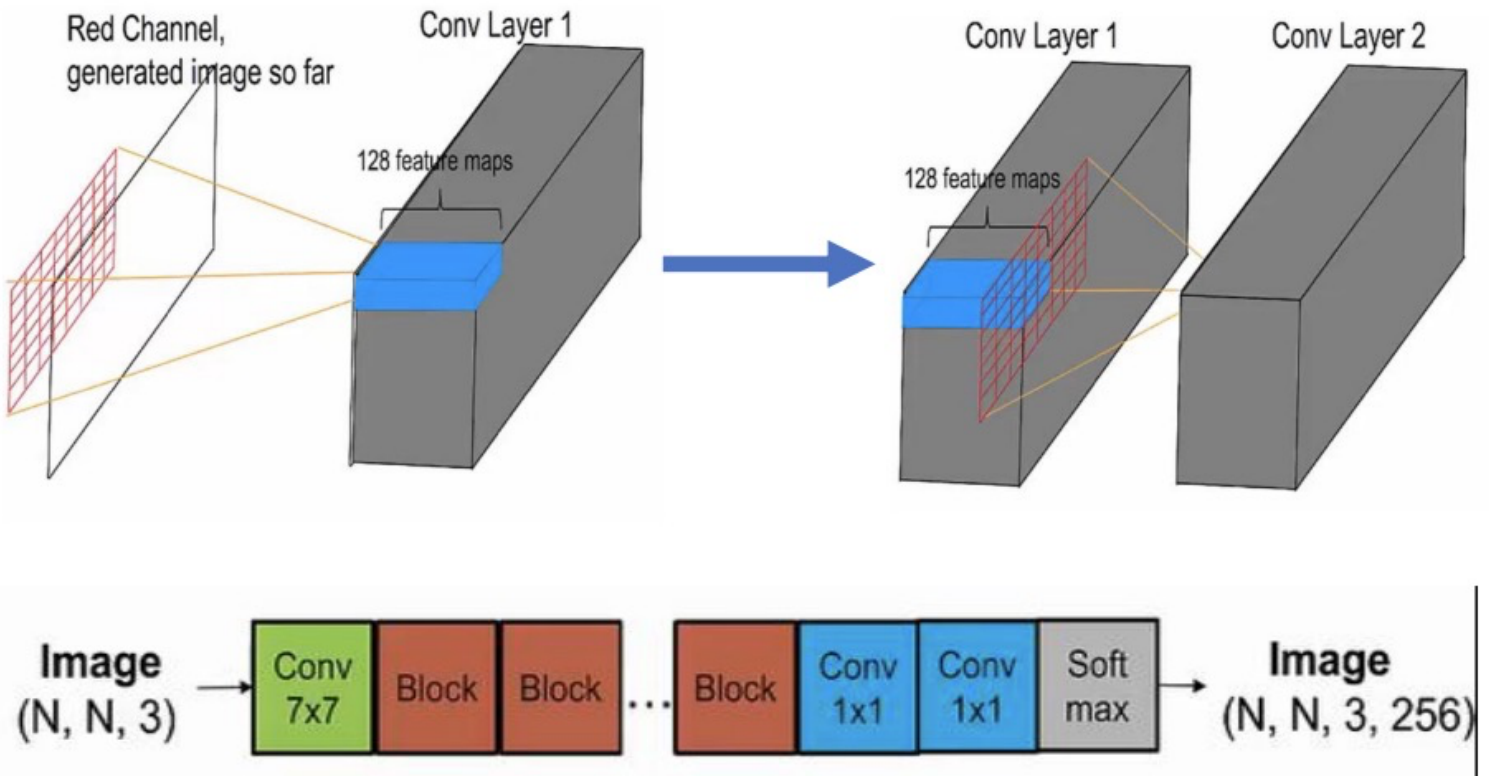
$\Rightarrow$ CNN for $\mathbf{h}_i$

3. Training is faster than PixelRNN
   - can parallelize convolutions since context region values known from training images.
4. Generation must still proceed sequentially, thus slow.



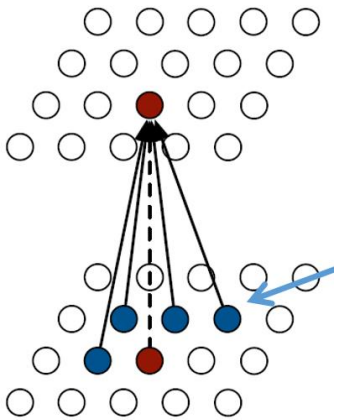
van der Oord *et al.* 2016

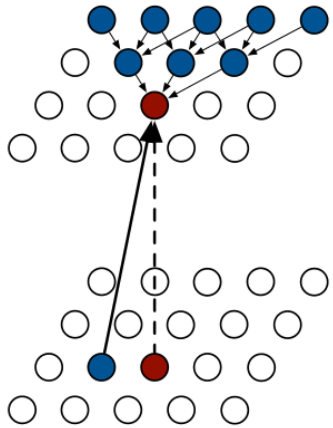
PixelCNN Architecture

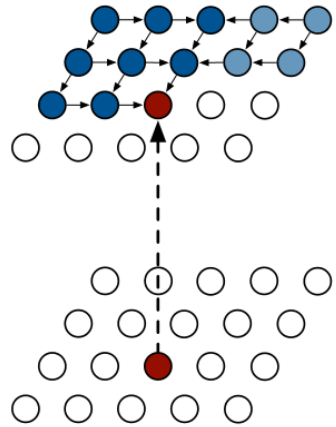


Comparison

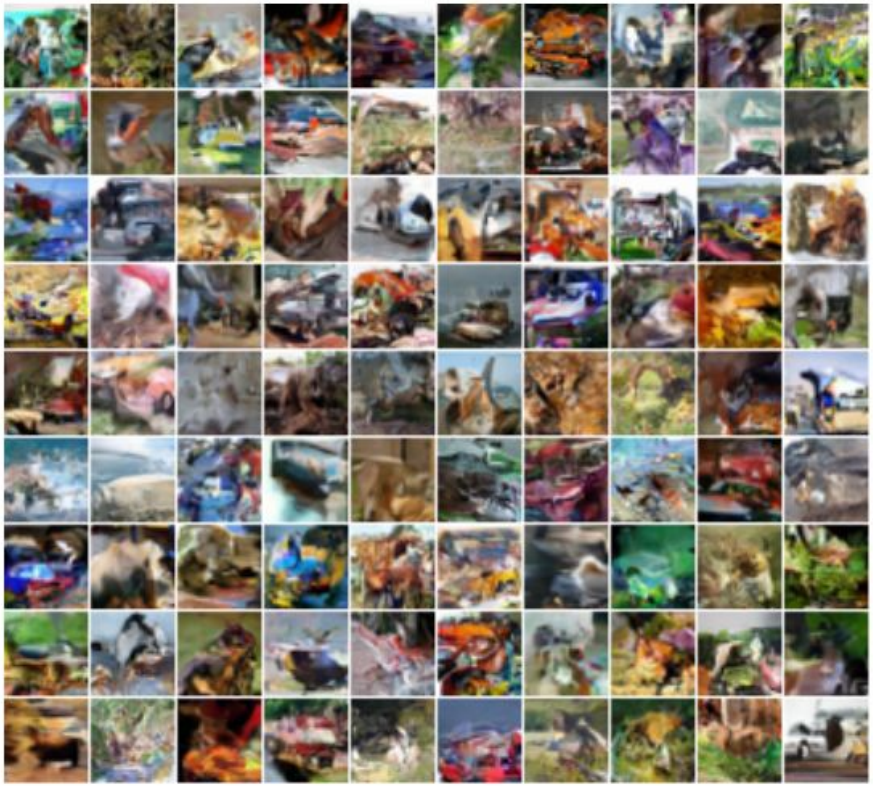
PixelCNN	PixelRNN – Row LSTM	PixelRNN – Diagonal BiLSTM
Full dependency field	Triangular receptive field	Full dependency field
Fastest	Slow	Slowest
Worst log-likelihood	-	Best log-likelihood



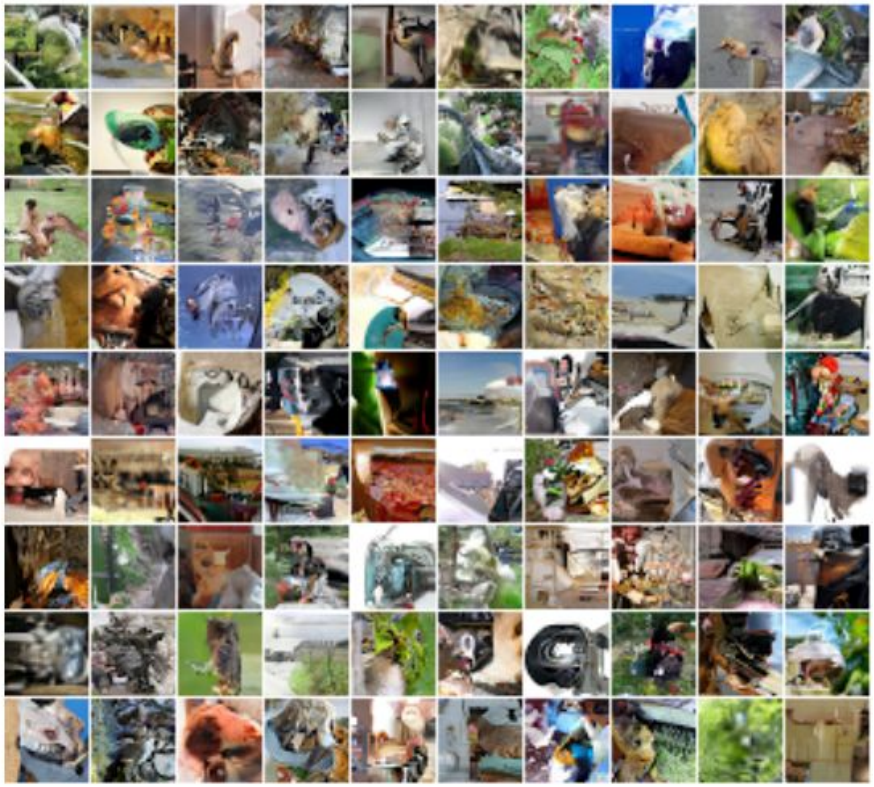




Generation Samples with Diagonal BiLSTM



32x32 CIFAR-10

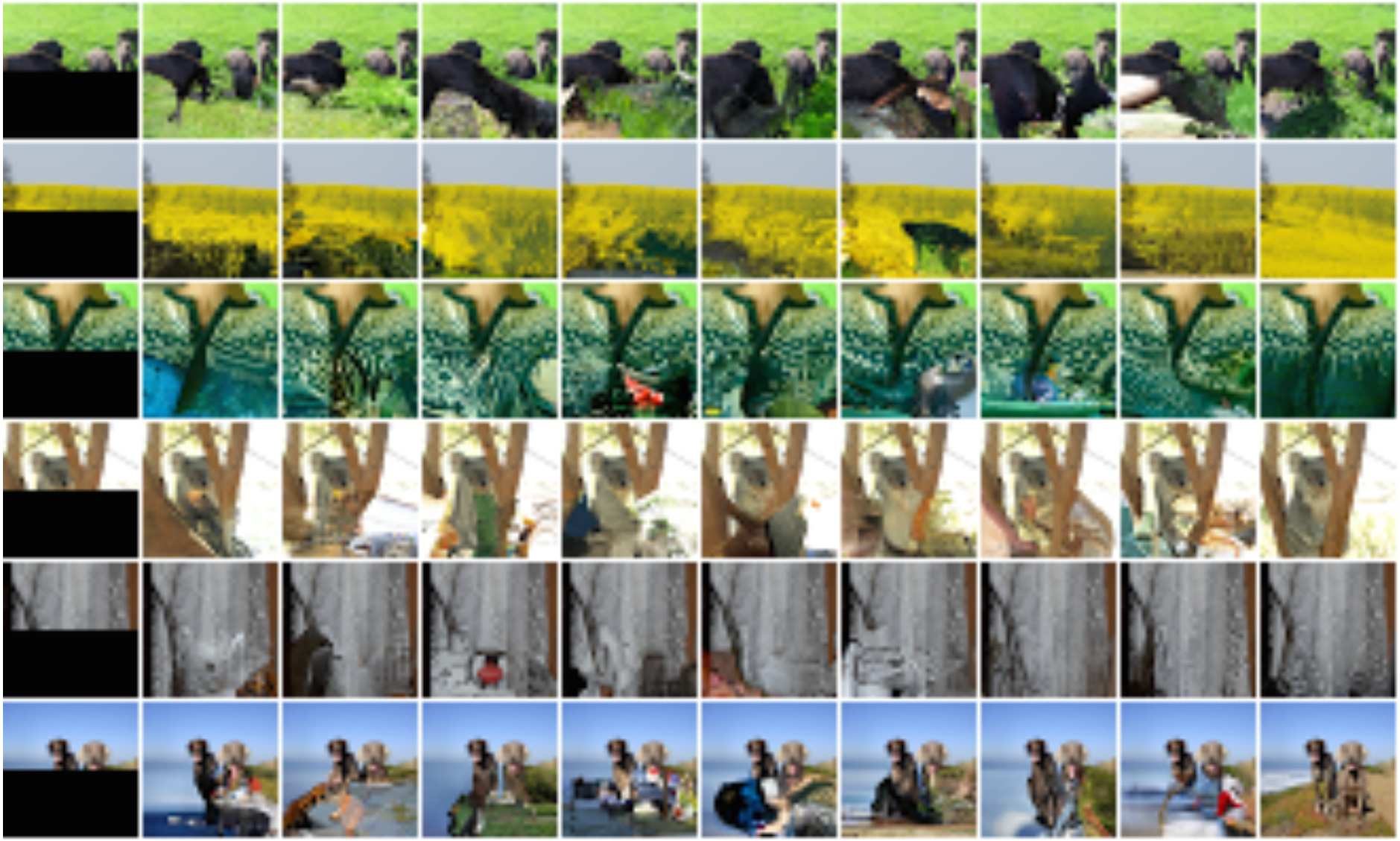


32x32 ImageNet

van der Oord *et al.* 2016

Image Completion

occluded completions original



Recap:Deep Generative Models

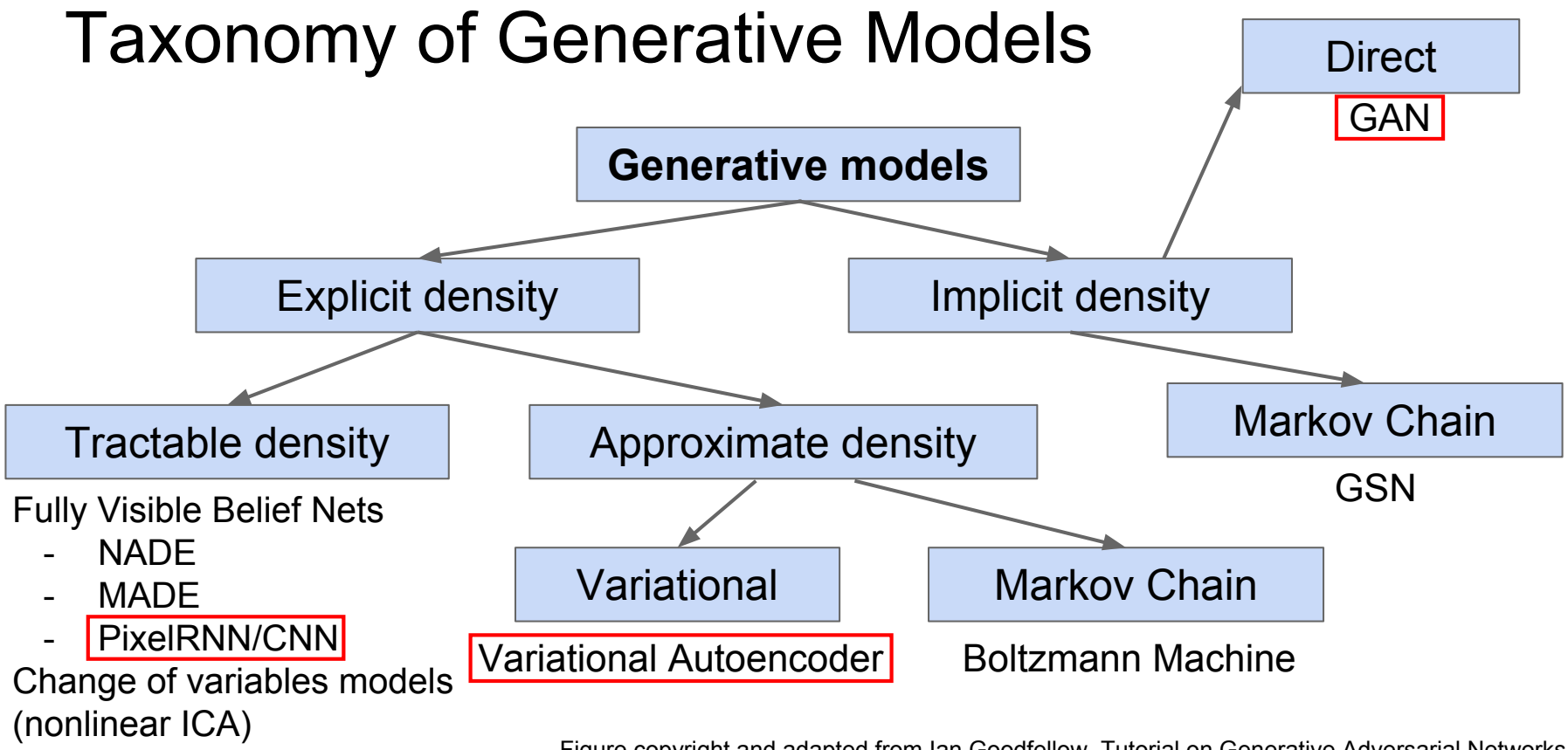


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Recap: Pixel RNN/CNN

Explicit density model

- Decompose likelihood of an image **x** into product of conditional distributions for all pixels:

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

Likelihood of image $\boldsymbol{x}$

Probability of i-th pixel given all previous pixels

- Solve by maximum likelihood estimator.
- Parameterize $p(x_i|x_1, \cdots, x_{i-1})$ with a neural network to represent a complex distribution:
  - Use RNN and CNN.

Variational Autoencoders[2] (VAE)

[2] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

VAE Overlook

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:

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

3. Learning/inference has to resorted to some approximations, *e.g.*, by optimizing on a lower bound of $p_\theta(\mathbf{x})$:
   - variational inference

VAE Overlook

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 $p_\theta(\mathbf{x})$:
   - variational inference

VAE Overlook

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 $p_\theta(\mathbf{x})$:
   - variational inference

Background: Variational Inference[3] (VAE)

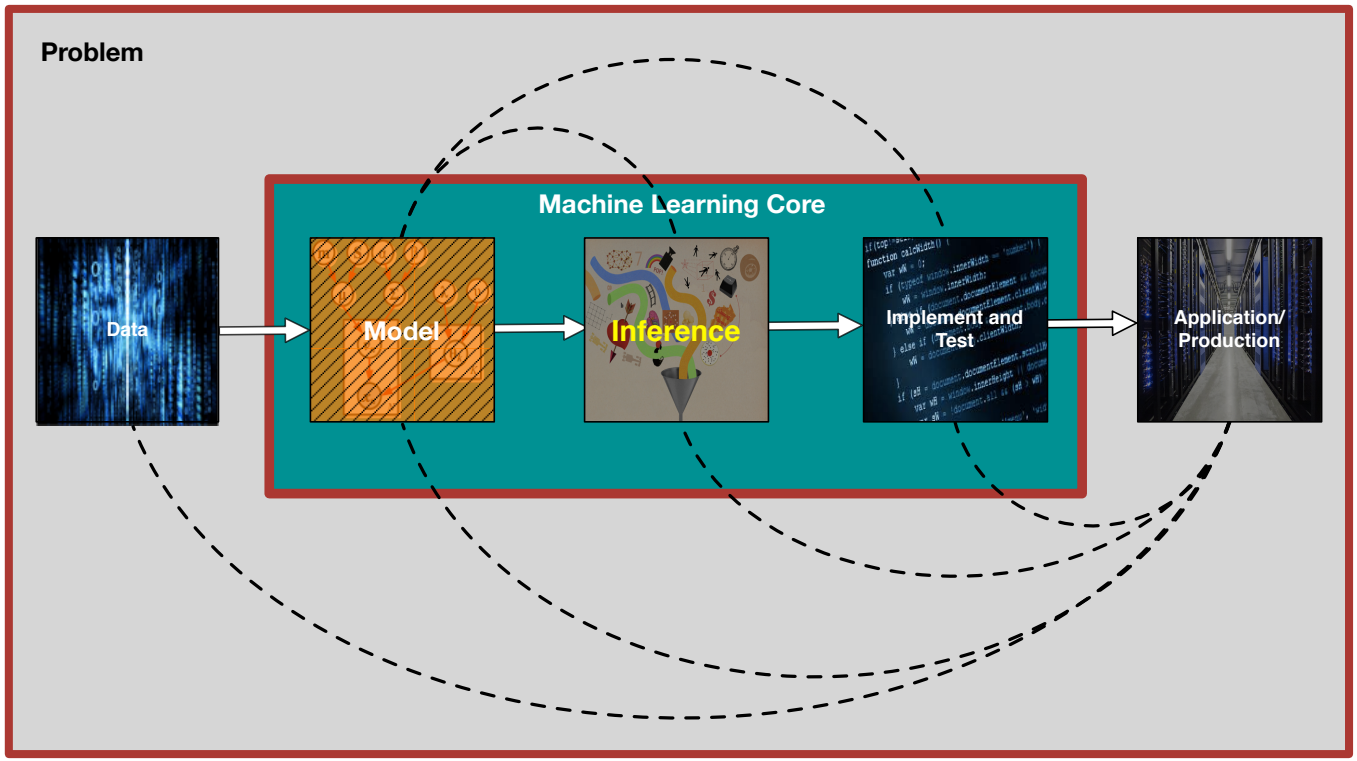
[3] Partially adapted from http://shakirm.com/papers/VITutorial.pdf

Probability Reasoning

Modern applications and data strongly favour probabilistic modelling:

- Noise in the data and account for our lack of knowledge.
- Non-i.i.d., non-stationary data.
- Explore and extract the underlying structure in the data.
- Consistency in our beliefs about the data and systems we study.

Probabilistic Inference



In probabilistic models, we must reason over the probability of events

Statistical Inference

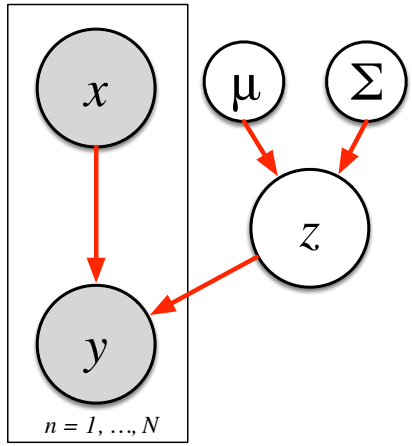
Any mechanism by which we deduce the probabilities in our model based on data.

Inference links the observed data with our statistical assumptions and allows us to ask questions of our data: predictions, visualisation, model selection.

Modeling and Inference

Probabilistic modelling will involve:

- Decide on a priori beliefs.
- Posit an explanation of how the observed data is generated, i.e. provide a probabilistic description.



Regression: Linear combination of inputs to give response.

Bayes' rule highlights many of the inferential problems we will face.

$$\underbrace{p(z|y)}_{\text{Posterior}} = \frac{\overbrace{p(y|z)}^{\text{Likelihood}}\ \overbrace{p(z)}^{\text{Prior}}}{\underbrace{\int p(y,z)dz}_{\text{Marginal likelihood/ Model evidence}}}$$

Inference Problems

Most inference problems will be one of:

- Marginalization:

$$p(y) = \int p(y, \theta) \mathrm{d}\theta$$

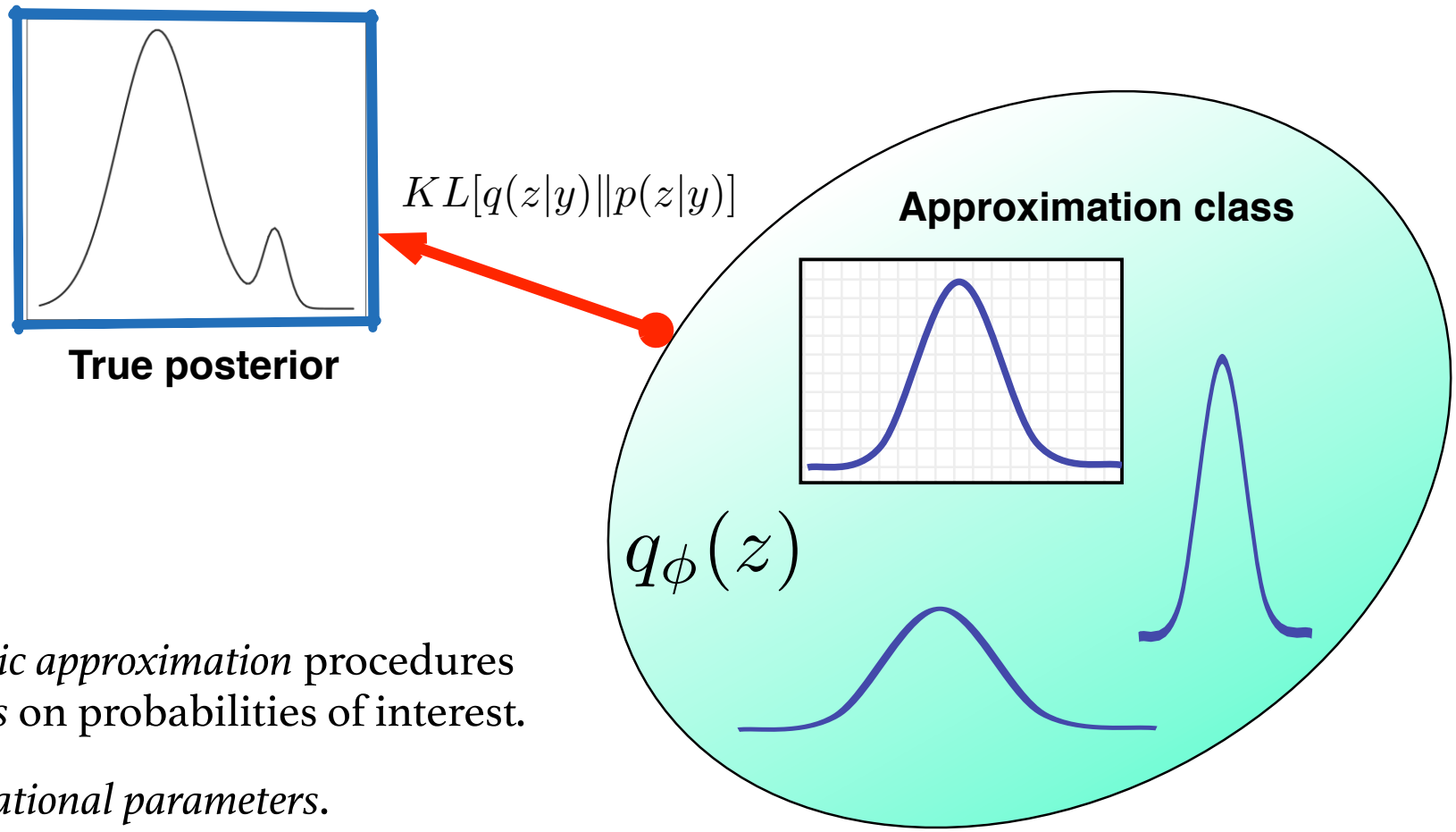
- Expectation:

$$\mathbb{E}\left[f(y)|x\right] = \int f(y) p(y|x) \mathrm{d}y$$

- Prediction:

$$p(y_{t+1}) = \int p(y_{t+1}|y_t) p(y_t) \mathrm{d}y_t$$

What is a Variational Method?

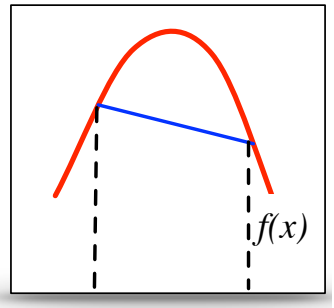


Deterministic *approximation* procedures with *bounds* on probabilities of interest.

Fit the *variational parameters*.

Jensen's Inequality

An important result from convex analysis:



For concave functions *f(.)*

$$f(\mathbb{E}[x]) \geq \mathbb{E}[f(x)]$$

Logarithms are strictly *concave* allowing us to use Jensen's inequality.

$$\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$$

Variational Inference

Integral problem

$$\log p(y) = \log \int p(y|z)p(z)dz$$

Proposal

$$\log p(y) = \log \int p(y|z)p(z)\frac{q(z)}{q(z)}dz$$

Importance Weight

$$\log p(y) = \log \int p(y|z)\frac{p(z)}{q(z)}q(z)dz$$

Jensen's inequality

$$\log \int p(x)g(x)dx \geq \int p(x) \log g(x)dx$$

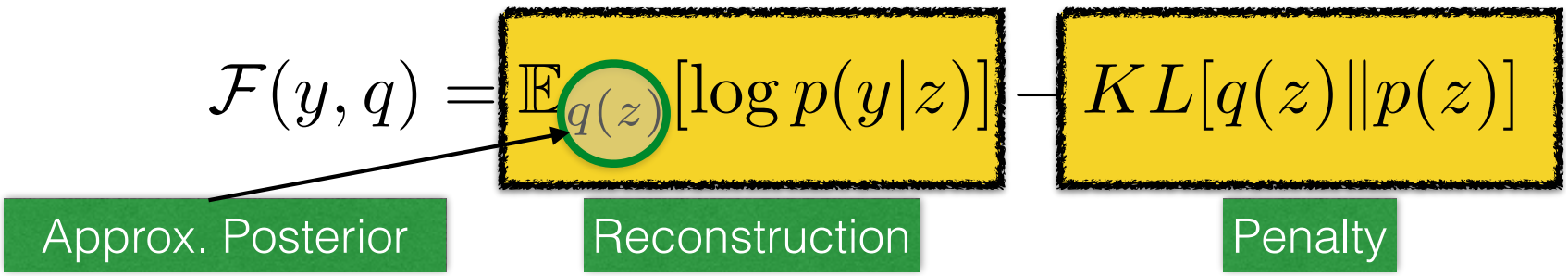
$$\log p(y) \geq \int q(z) \log \left(p(y|z)\frac{p(z)}{q(z)} \right) dz$$

$$= \int q(z) \log p(y|z) - \int q(z) \log \frac{q(z)}{p(z)}$$

Variational lower bound

$$= \mathbb{E}_{q(z)}[\log p(y|z)] - KL[q(z)\|p(z)]$$

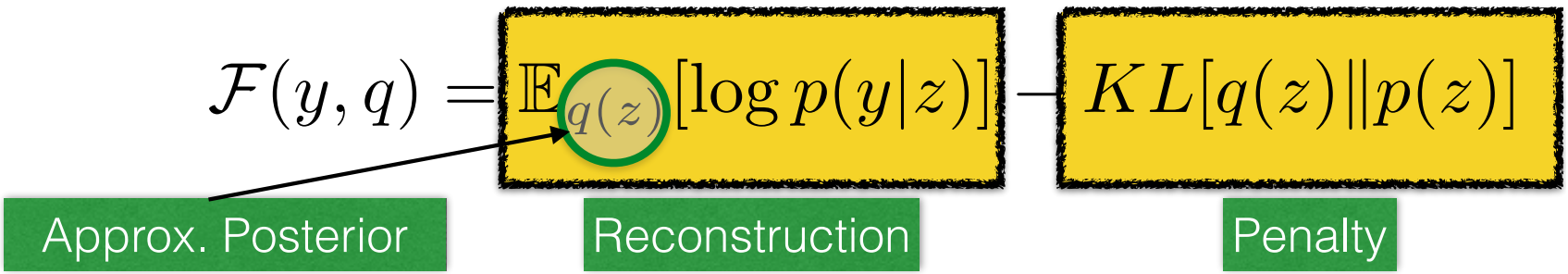
Variational Inference



Interpretation

- Approximate posterior distribution $q(z)$: Best match to true posterior $p(z|y)$, one of the unknown inferential quantities of interest to us.
- Reconstruction cost: The expected log-likelihood measure how well samples from $q(z)$ are able to explain the data y.
- Penalty: Ensures the the explanation of the data $q(z)$ does not deviate too far from your beliefs $p(z)$. A mechanism for realizing Okham's razor.

Variational Inference



Some comments on q

- Integration is now optimization: optimize for $q(z)$ directly:

  z typically depends on data y, but write $q(z)$ for notation simplicity.
  Easy convergence assessment.

- Variational parameters: parameters of $q(z)$:

  e.g., if a Gaussian, the parameters are mean and variance.
  Optimization allows us to tighten the bound and get as close as possible to the true marginal likelihood.

An Identity

$$\mathcal{F}(y, q) = \log p(y) - KL(q(z) \| p(z|y))$$

$$\rightarrow \max_q \mathcal{F}(y, q) \equiv \min_q KL(q(z) \| p(z|y))$$

Free-form and Fixed-form Solutions (Optional)

Free-form

- Solves for the exact distribution $q(z)$ by setting the functional derivative to zero via calculus of variations:

$$\frac{\delta \mathcal{F}(y, q)}{\delta q(z)} = 0, \quad s.t. \quad \int q(z) \mathrm{d}z = 1$$
$$\Rightarrow q(z) \propto p(z) p(y|z, \theta)$$

- The optimal solution is the true posterior distribution, but solving for the normalization is our original problem.

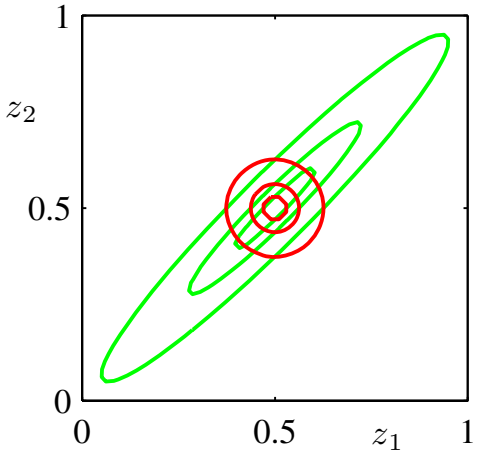
Fixed-form

- Specify an explicit form of $q(z)$, *e.g.*, a normal distribution.
- Parameter in q is called the variational parameter.

Mean-field Variational Inference

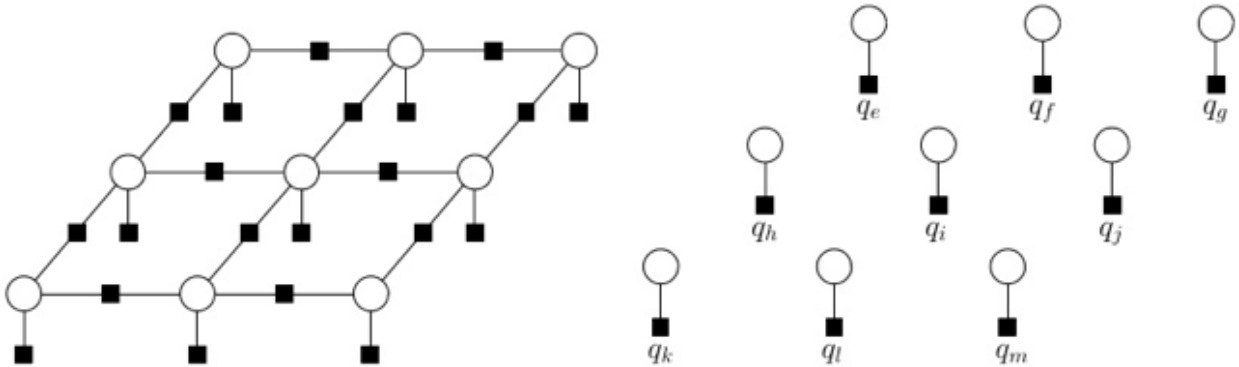
Mean-field methods assume that the distribution is factorised.

$$q(z) = \prod_i q_i(z_i)$$



$$q(z) = \prod_i \mathcal{N}(z_i|\mu_i, \sigma_i^2)$$

Restricted class of approximations: every dimension (or subset of dimensions) of the posterior is independent.



Example: Mean-field for Latent Gaussian Models

Generative model:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), \quad \mathbf{y} \sim p(\mathbf{y} \,|\, f_\theta(\mathbf{z}))$$

Variational distribution:

$$q(\mathbf{z}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2)$$

Example: Mean-field for Latent Gaussian Models

$$\begin{aligned}
\mathcal{F}(\mathbf{y}, q) &= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - KL\left[q(\mathbf{z}) \| p(\mathbf{z})\right] \\
&= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - \sum_i KL\left[q(z_i) \| p(z_i)\right] \\
&= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - \sum_i KL\left[\mathcal{N}(z_i; \mu_i, \sigma_i^2) \| \mathcal{N}(z_i; 0, 1)\right] \\
&= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - \frac{1}{2}\sum_i \left(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2\right)
\end{aligned}$$

KL between two Gaussians

Let $p(x) = \mathcal{N}(x; \mu_1, \sigma_1^2)$, $q(x) = \mathcal{N}(x; \mu_2, \sigma_2^2)$, then

$$KL(p(x) \| q(x)) = \log \frac{\sigma_2^2}{\sigma_1^2} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

Example: Mean-field for Latent Gaussian Models

$$
\begin{aligned}
\mathcal{F}(\mathbf{y}, q) &= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - KL\left[q(\mathbf{z}) \| p(\mathbf{z})\right] \\
&= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - \sum_i KL\left[q(z_i) \| p(z_i)\right] \\
&= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - \sum_i KL\left[\mathcal{N}(z_i; \mu_i, \sigma_i^2) \| \mathcal{N}(z_i; 0, 1)\right] \\
&= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{y} \mid \mathbf{z})\right] - \frac{1}{2}\sum_i \left(\sigma_i^2 + \mu_i^2 - 1 - \log \sigma_i^2\right)
\end{aligned}
$$

Optimize parameters of $q(z)$ by gradient descent.

Optimization for the Variational Bound

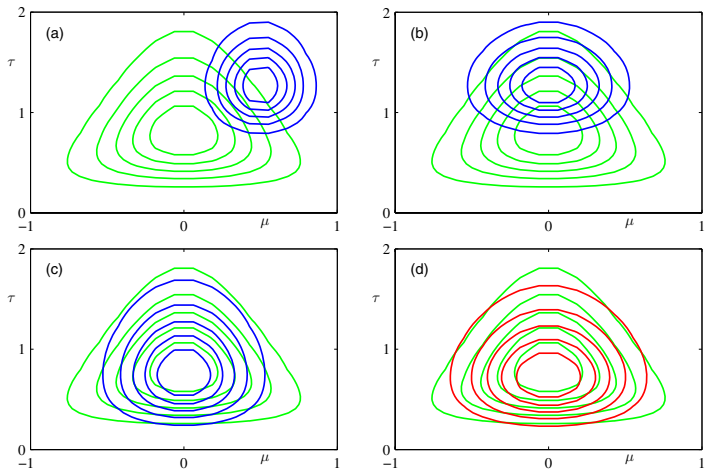
$$\max_{q,\theta} \mathcal{F}(y, q) = \mathbb{E}_{q(z)}[\log p(y|z, \theta)] - KL[q(z)||p(z)]$$

Approx. Posterior

Reconstruction

Penalty

- *Variational EM*
- *Stochastic Variational Inference*
- *Doubly Stochastic Variational Inference*
- *Amortised Inference*



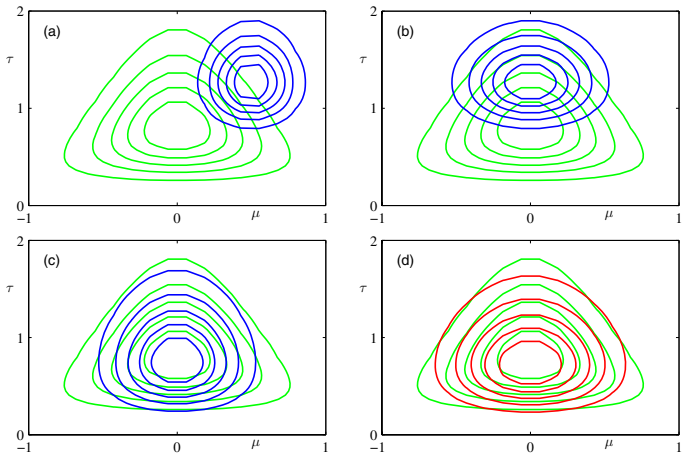
Optimization for the Variational Bound

$$\max_{q,\theta} \mathcal{F}(y, q) = \mathbb{E}_{q(z)}[\log p(y|z, \theta)] - KL[q(z)\|p(z)]$$

Approx. Posterior | Reconstruction | Penalty

- *Variational EM*

- *Stochastic Variational Inference*
- *Doubly Stochastic Variational Inference*
- *Amortised Inference*



Dealt with big data and complicated $q(z)$

Variational Inference

- Variational EM:
  - $q(z)$
- Stochastic variational inference:
  - $q(z|\alpha_z)$
- Amortised inference:
  - $q(z|\alpha)$

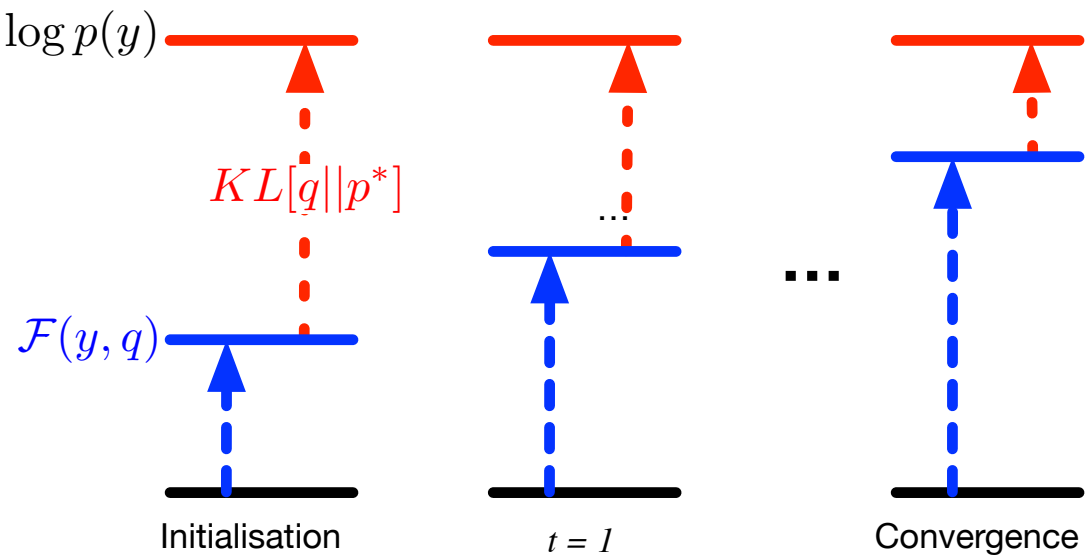
Variational Expectation Maximization

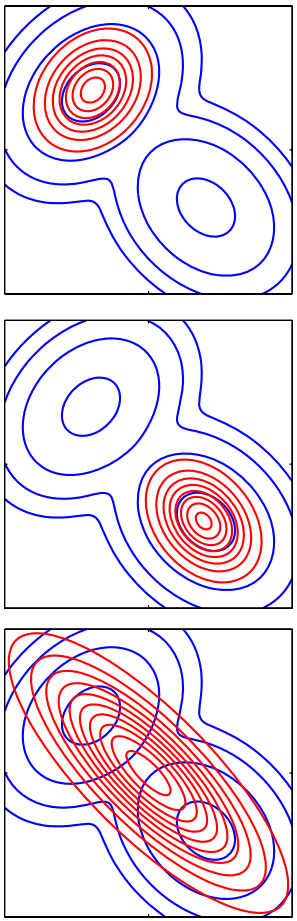
Alternating optimization for the variational parameters and then model parameters.

Identity: $\log p(y) = KL(q(z)||p*(z|y)) + \mathcal{F}(y, q(z))$

Repeat:

E-step	$\phi \propto \nabla_\phi \mathcal{F}(y, q)$	*Var. params*
M-step	$\theta \propto \nabla_\theta \mathcal{F}(y, q)$	*Model params*





Variational Expectation Maximization

Alternating optimization for the variational parameters and then model parameters.

Identity: $\log p(y) = KL(q(z)||p*(z|y)) + \mathcal{F}(y, q(z))$

Repeat:

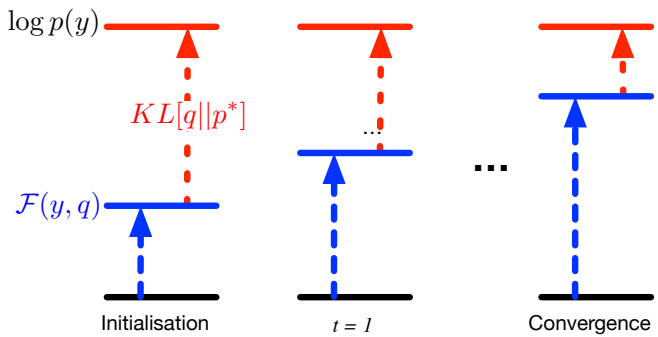
E-step *(Inference)*

For i = 1, ... N

$$\phi_n \propto \nabla_\phi \mathbb{E}_{q_\phi(z)}[\log p_\theta(y_n|z_n)] - \nabla_\phi KL[q(z_n)||p(z_n)]$$

M-step *(Parameter Learning)*

$$\theta \propto \frac{1}{N}\sum_n \mathbb{E}_{q_\phi(z)}[\nabla_\theta \log p_\theta(y_n|z_n)]$$

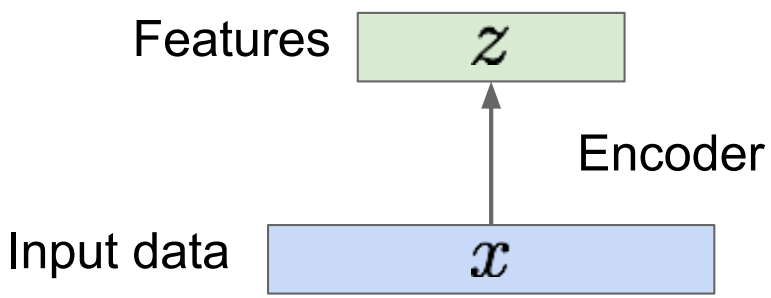


Autoencoder[4]

[4] Partially adapted from http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture13.pdf

Background: Autoencoder

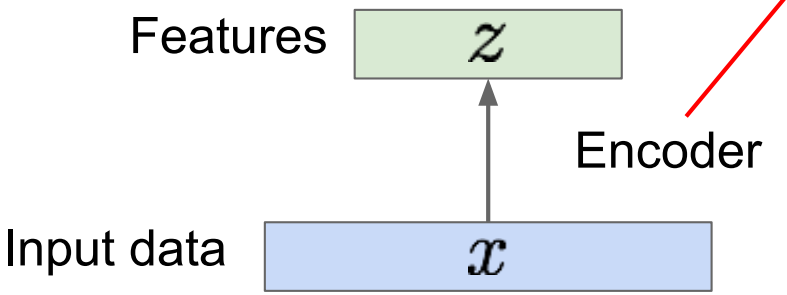
Learning a lower-dimensional feature representation from unlabeled training data



Background: Autoencoder

Learning a lower-dimensional feature representation from unlabeled training data

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN



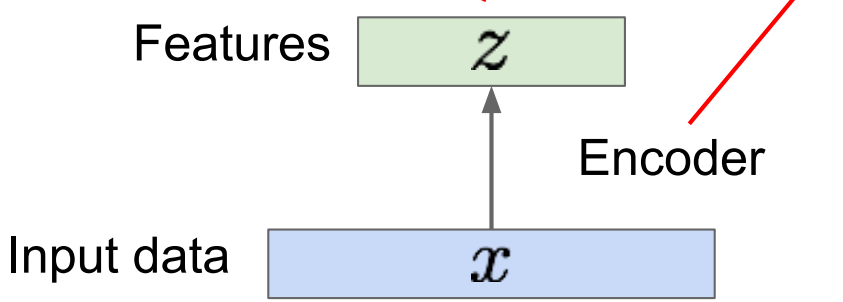
Background: Autoencoder

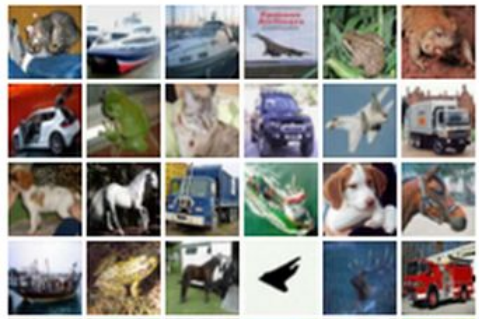
Learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





Background: Autoencoder

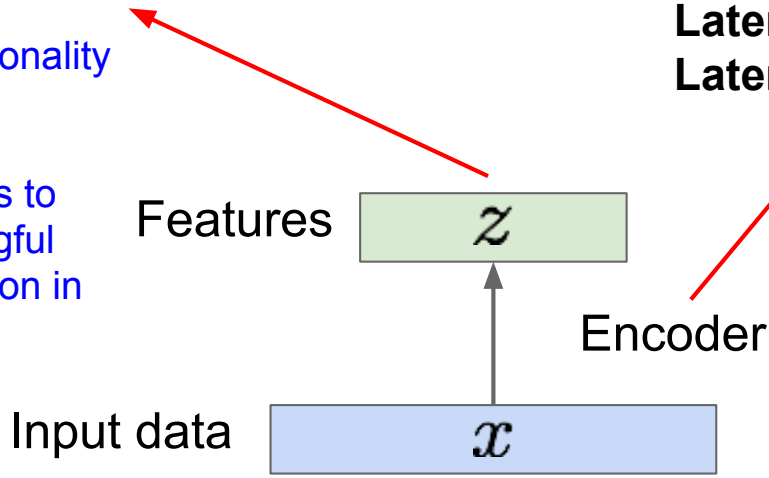
Learning a lower-dimensional feature representation from unlabeled training data

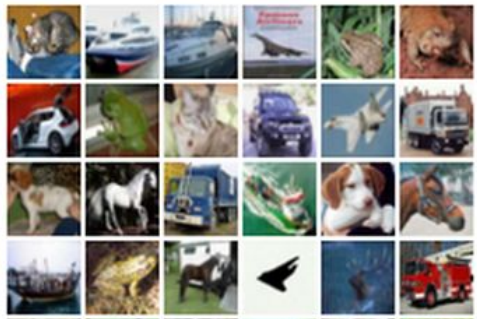
z usually smaller than **x** (dimensionality reduction)

Q: Why dimensionality reduction?

A: Want features to capture meaningful factors of variation in data

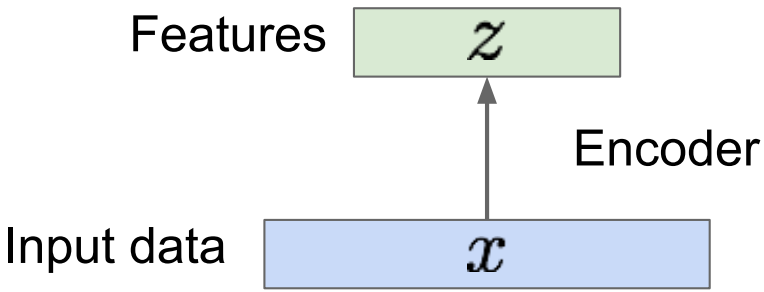
Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





Background: Autoencoder

How to learning this feature representation?

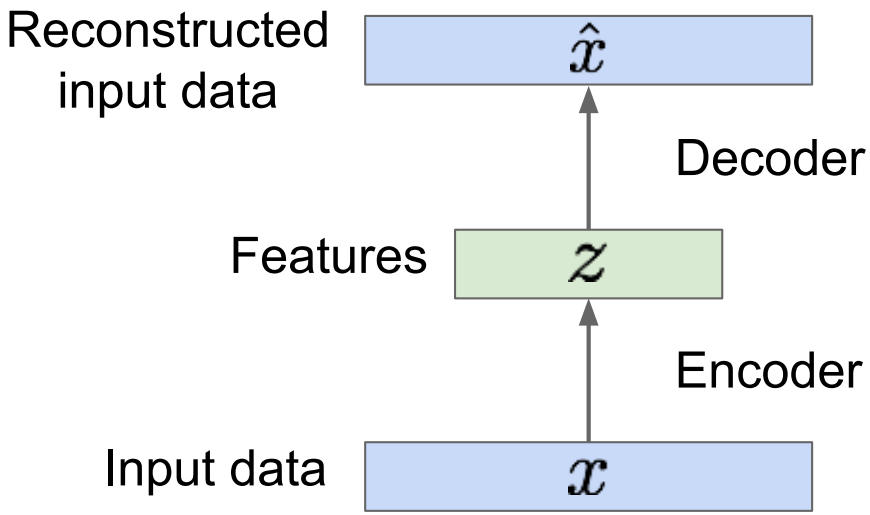


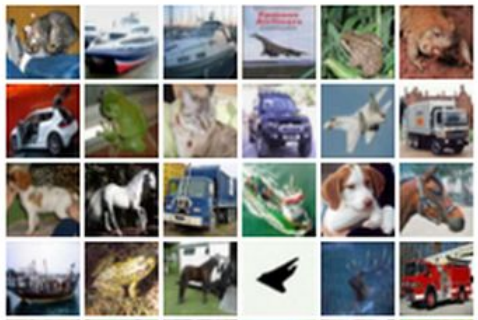
Background: Autoencoder

How to learning this feature representation?

We want that feature can be used to reconstruct itself:

- "Autoencoding" – encoding itself.



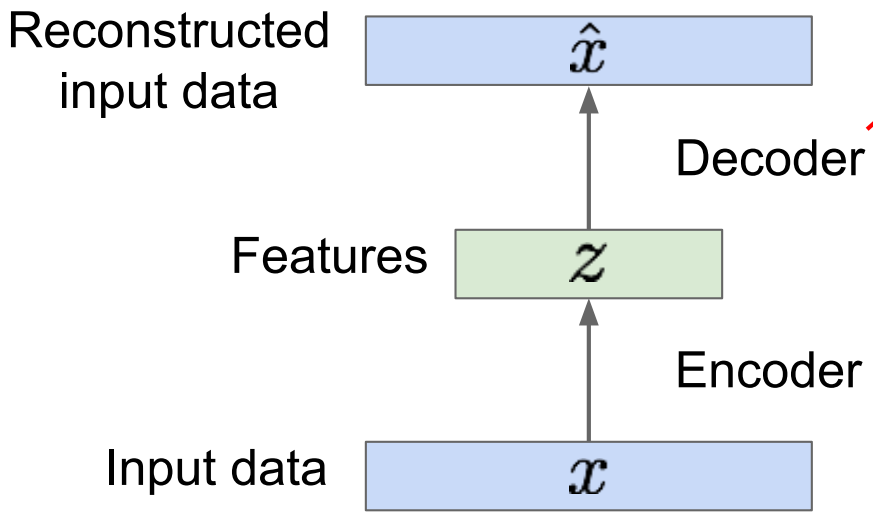


Background: Autoencoder

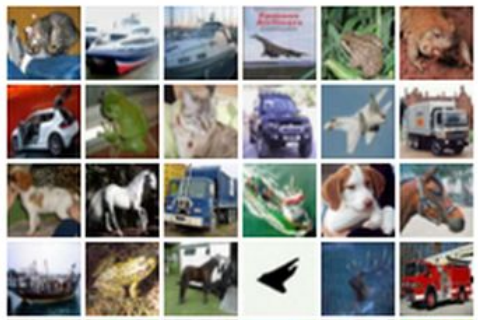
How to learning this feature representation?

We want that feature can be used to reconstruct itself:

- "Autoencoding" – encoding itself.

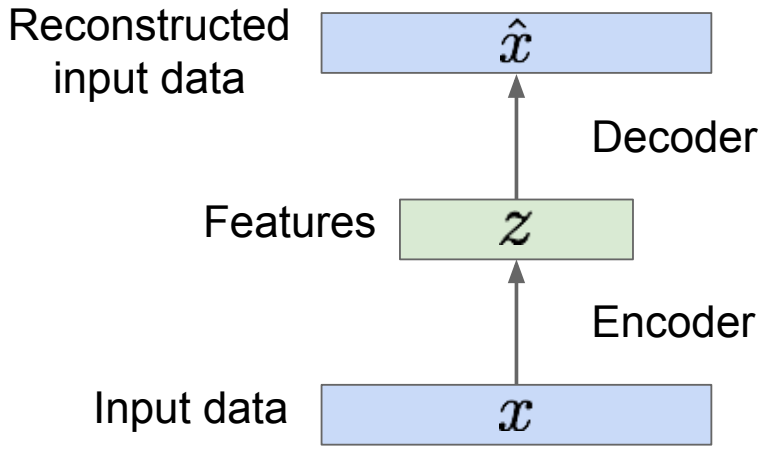


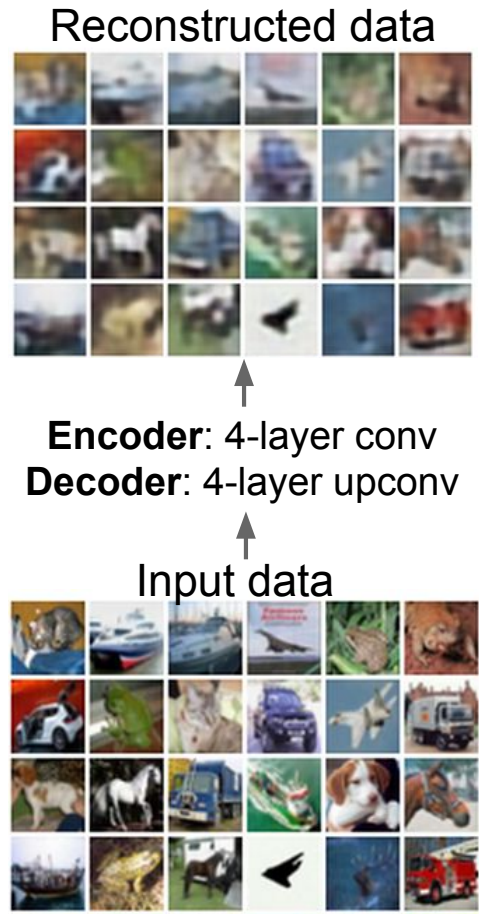
Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)



Background: Autoencoder

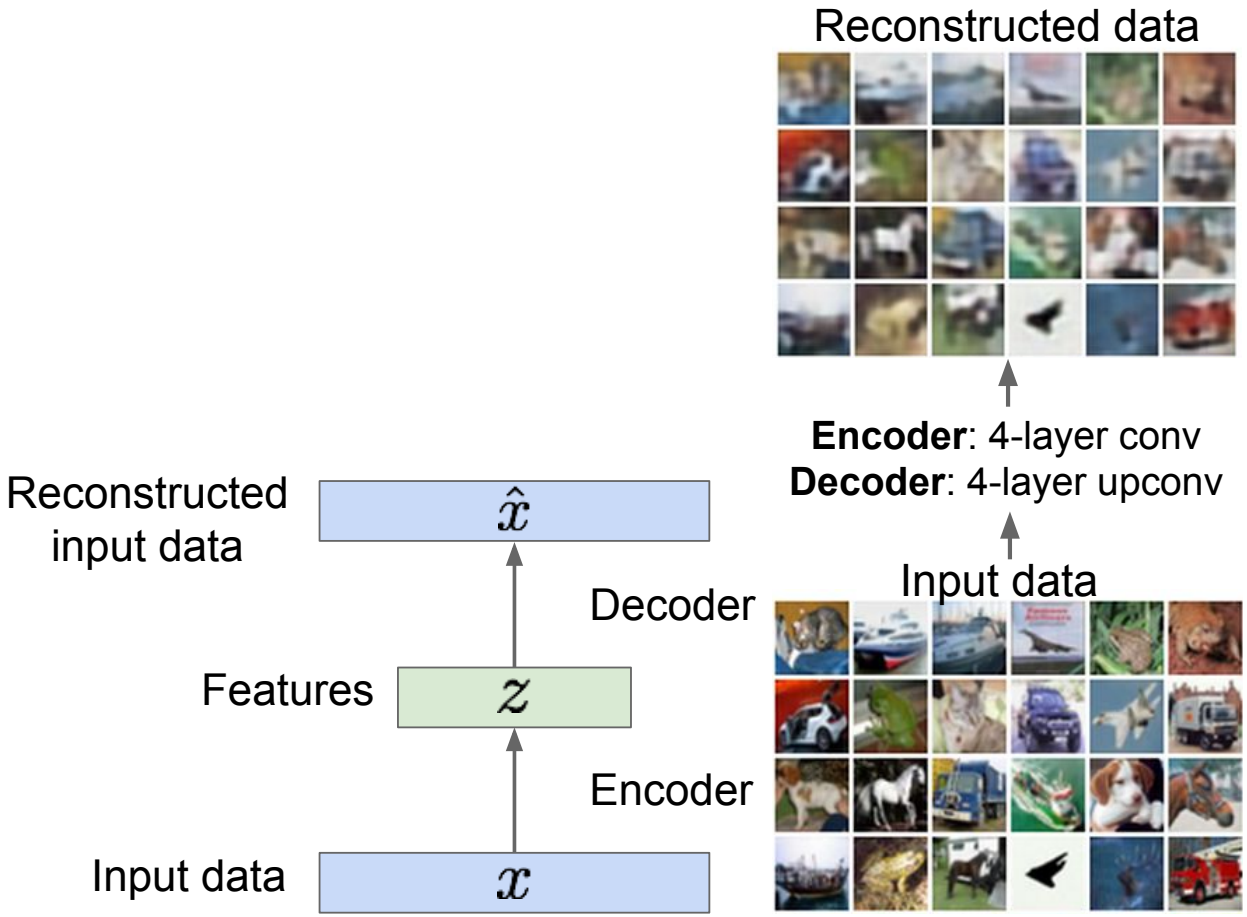
- We want that feature can be used to reconstruct itself
- Use L^2 loss: $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$
- Doesn't use labels





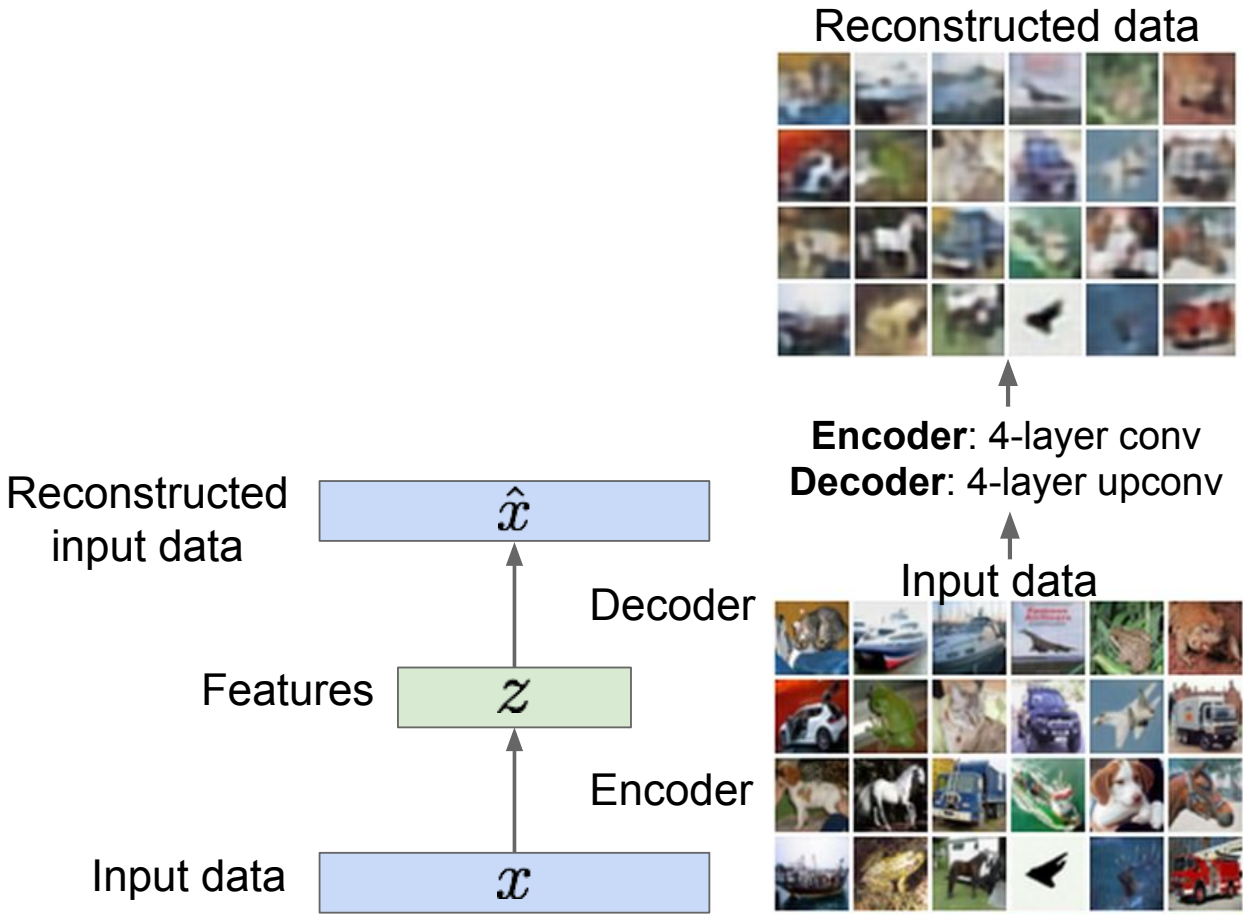
Background: Autoencoder

- We want that feature can be used to reconstruct itself
- Use L^2 loss: $\| \mathbf{x} - \hat{\mathbf{x}} \|^2$
- Doesn't use labels



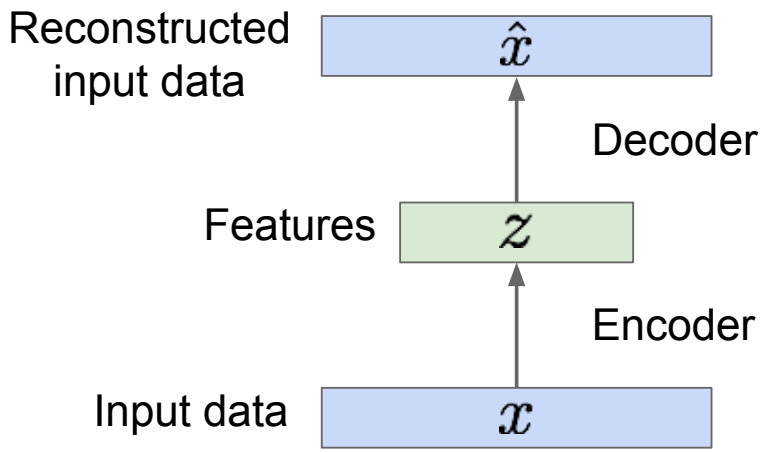
Background: Autoencoder

- We want that feature can be used to reconstruct itself
- Use L^2 loss: $\| \mathbf{x} - \hat{\mathbf{x}} \|^2$
- Doesn't use labels



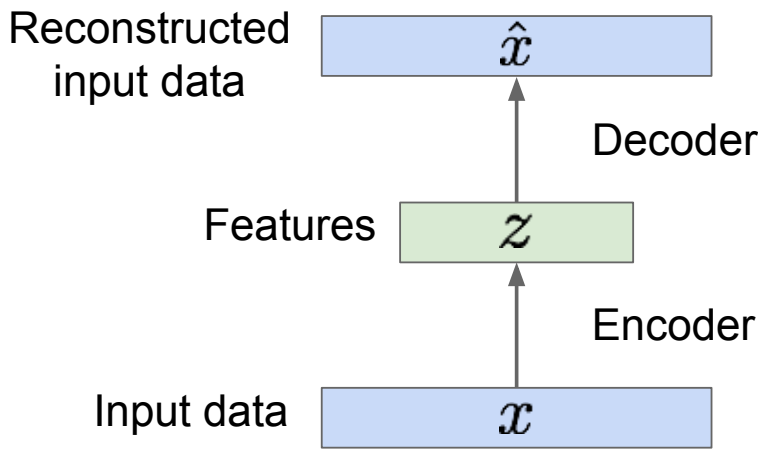
Background: Autoencoder

- What can an autoencoder do?
  - Reconstruct data.
  - Pretrain for supervised learning.



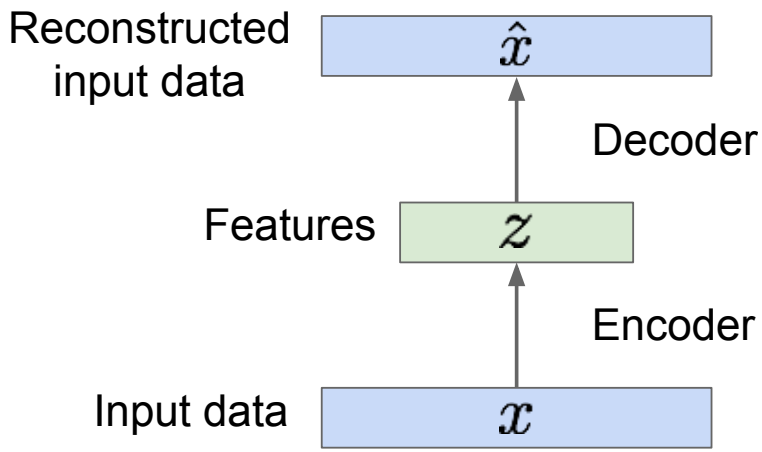
Background: Autoencoder

- What can an autoencoder do?
  - Reconstruct data.
  - Pretrain for supervised learning.



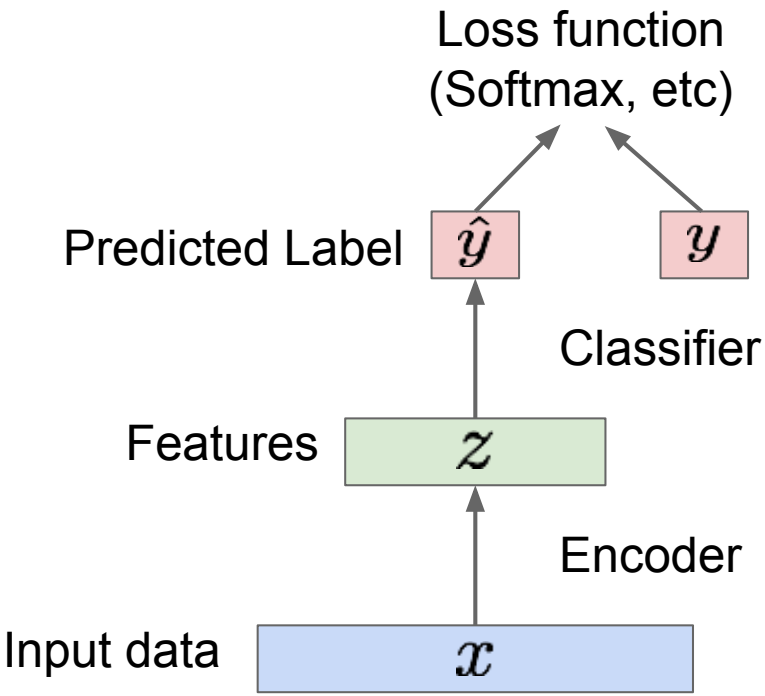
Background: Autoencoder

- What can an autoencoder do?
  - Reconstruct data.
  - Pretrain for supervised learning.



Background: Autoencoder

After training, throw away decoder.



Encoder can be used to initialize a **supervised** model

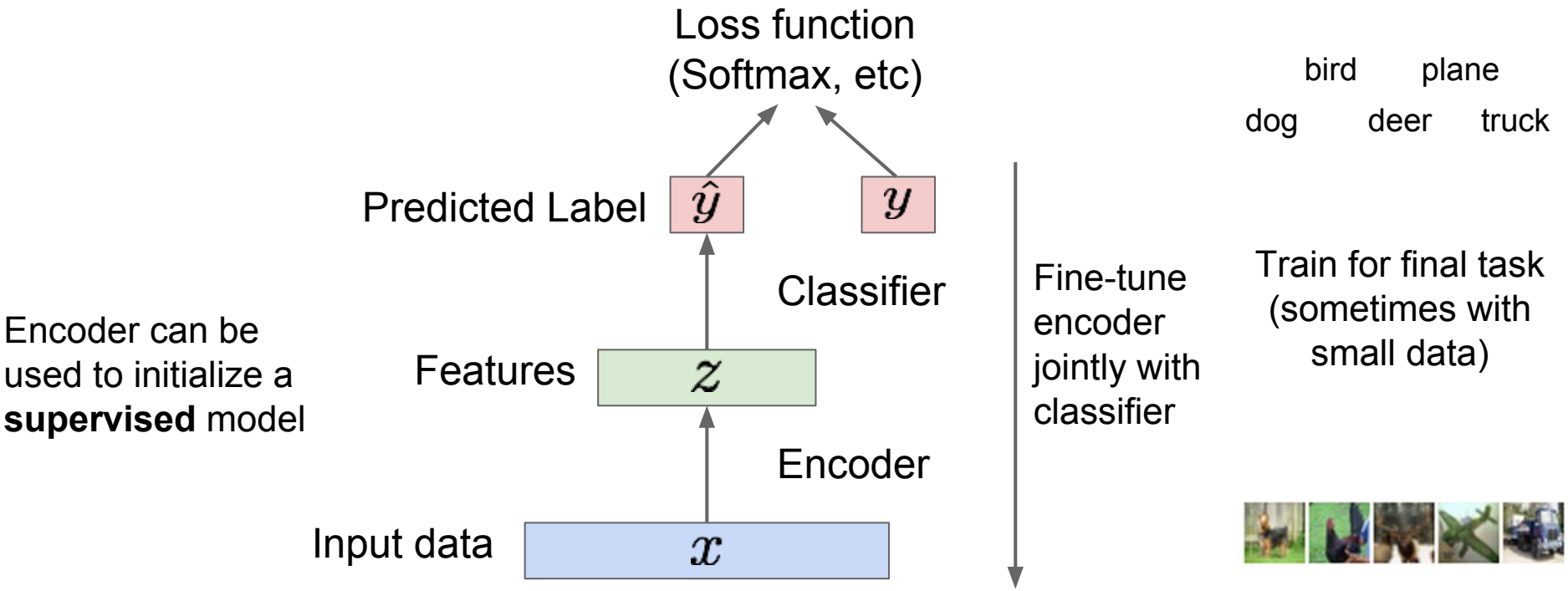
Fine-tune encoder jointly with classifier

bird plane
dog deer truck

Train for final task (sometimes with small data)

Background: Autoencoder

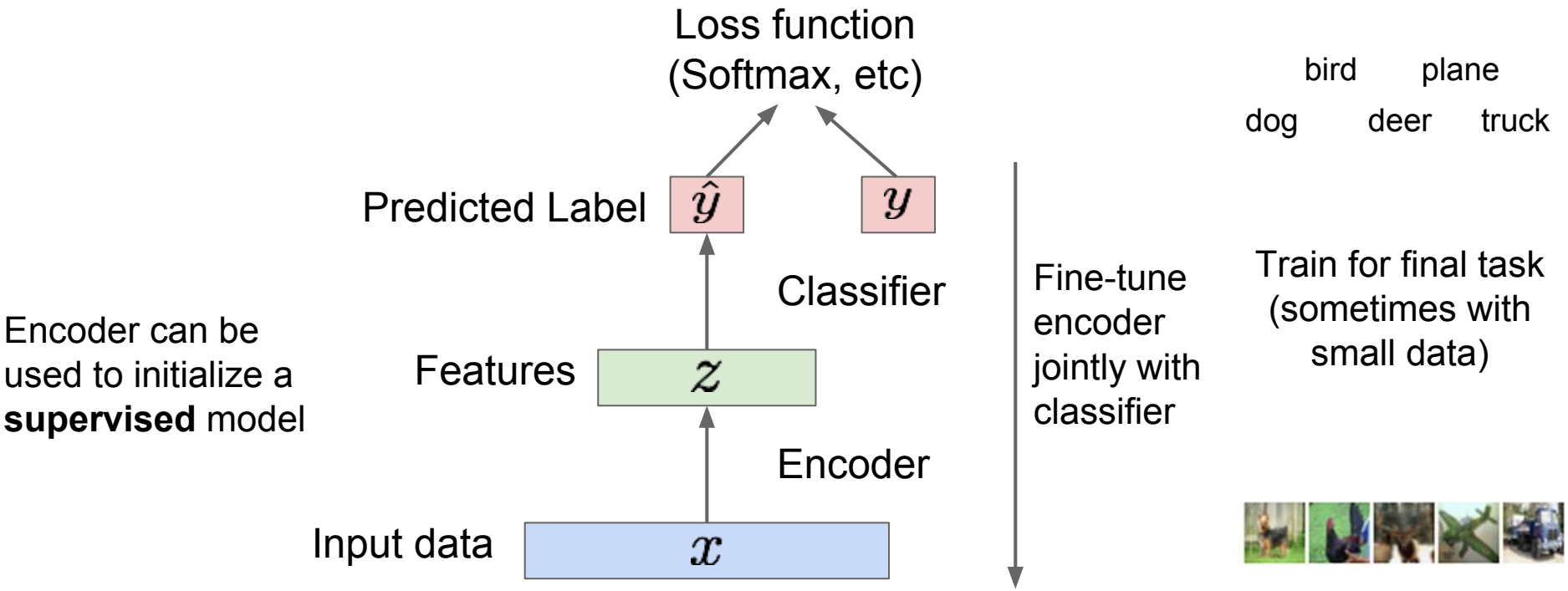
After training, throw away decoder.



- Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
- However, can we generate new images from it?
  - NO

Background: Autoencoder

After training, throw away decoder.



- Autoencoders can reconstruct data, and can learn features to initialize a supervised model.
- However, can we generate new images from it?
  - NO

Variational Autoencoder

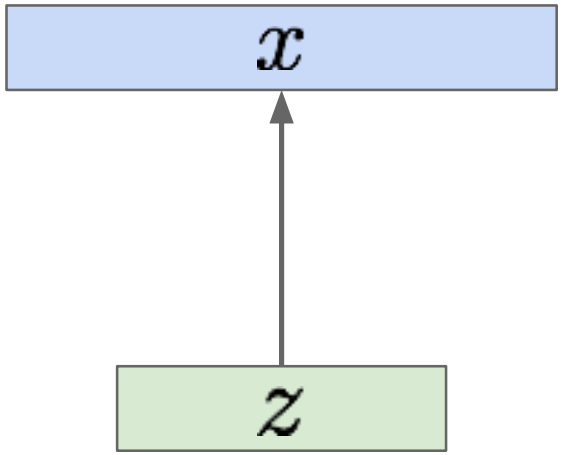
Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

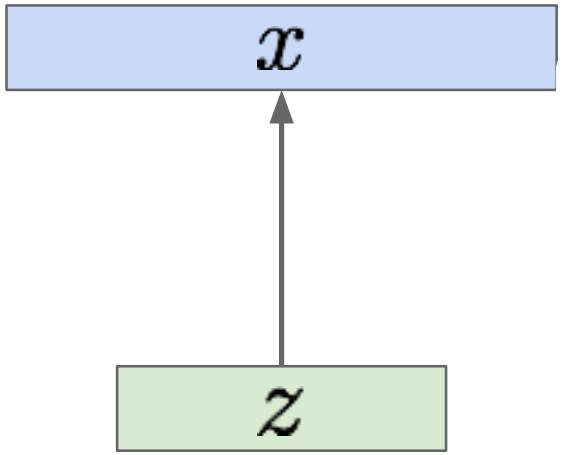
1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.
   - Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
   - Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$



Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.
   - Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
   - Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$



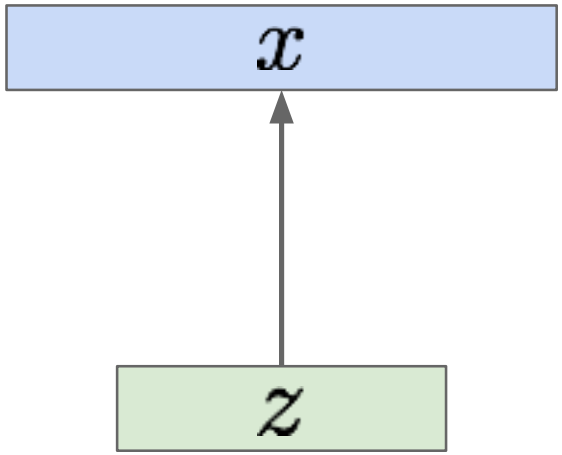
Intuition

$\mathbf{x}$ is an image, $\mathbf{z}$ is latent factors used to generate $\mathbf{x}$, *e.g.*, attributes, orientation, *etc*

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.
   - Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
   - Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$

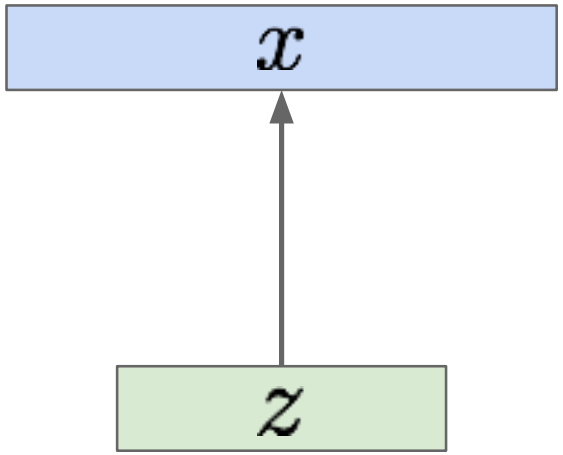


We want to estimate the true parameters θ^* of this generative model.

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.
   - Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
   - Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$

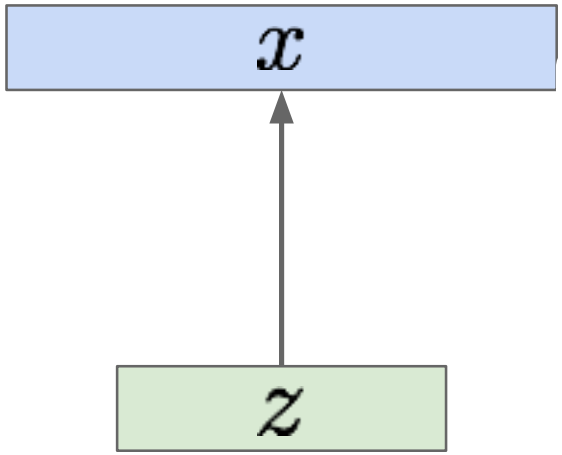


How should we represent this model?

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.
   - Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
   - Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$



How should we represent this model?

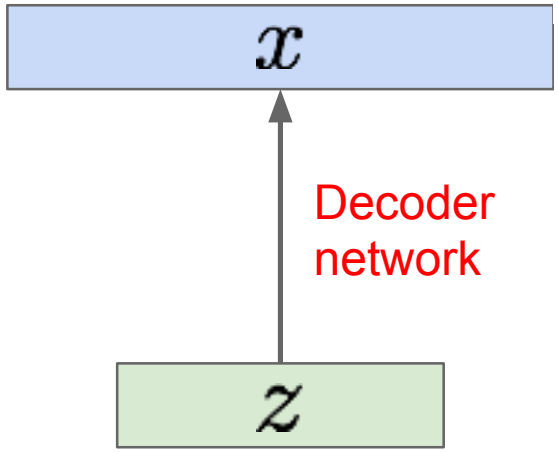
- Choose prior p(z) to be simple, *e.g.* Gaussian.

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.

- Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
- Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$



How should we represent this model?

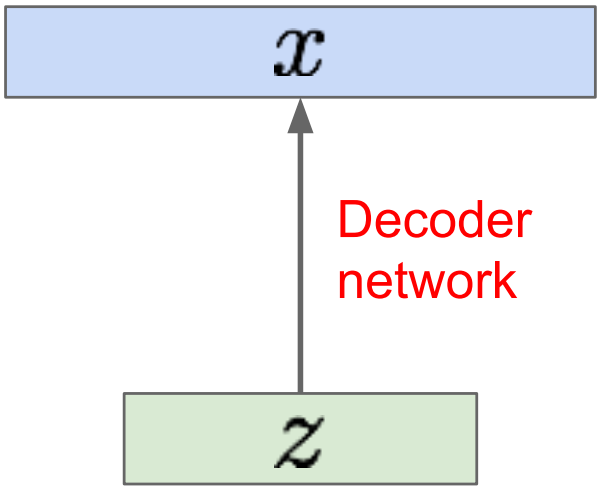
- Choose prior p(z) to be simple, *e.g.* Gaussian.
- Chose $p(\mathbf{x} \mid \mathbf{z})$ to be complex (generates image) $\rightarrow$ represented with a neural network.

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.

- Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
- Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$



How to train the model?

- Maximum likelihood?

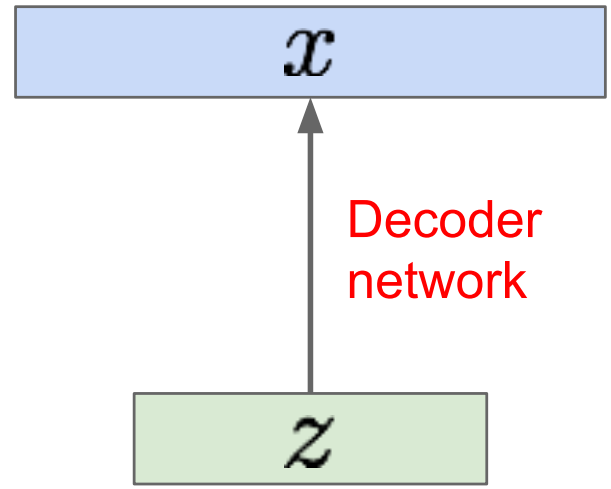
$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\,\mathbf{z}$$

Kingma & Welling, ICLR 2014

Variational Autoencoder $\Rightarrow$ Probabilistic Autoencoder

1. Define a generative probability model for the decoder.
2. Assume training data $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$ is generated from underlying unobserved (latent) representation $\mathbf{z}$.

- Sample latent $\mathbf{z}$ from true prior $p_{\theta^*}(\mathbf{z})$
- Sample data $\mathbf{x}$ from true conditional $p_{\theta^*}(\mathbf{x} \mid \mathbf{z})$



How to train the model?

- Maximum likelihood?

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

Intractable $\Rightarrow$ Variational Inference!

Kingma & Welling, ICLR 2014

Variational Autoencoder: Intractability

- Data likelihood: $p_{\theta}(\mathbf{x}) = \int p_{\theta}(\mathbf{x} \mid \mathbf{z}) p_{\theta}(\mathbf{z}) \mathrm{d}\,\mathbf{z}$

Variational Autoencoder: Intractability

- Data likelihood: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z}) \underbrace{p_\theta(\mathbf{z})}_{\text{simple Gaussian prior } \checkmark} \mathrm{d}\mathbf{z}$

Variational Autoencoder: Intractability

- Data likelihood: $p_{\theta}(\mathbf{x}) = \int \underbrace{p_{\theta}(\mathbf{x} \mid \mathbf{z})}_{\text{decoder neural network } \checkmark} p_{\theta}(\mathbf{z}) \mathrm{d}\,\mathbf{z}$
  - typically model as:

$$p_{\theta}(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_{\theta}(\mathbf{z}), \Sigma_{\theta}(\mathbf{z})\right) \ ,$$

where μ_{θ} and Σ_{θ} are two neural networks parameterized by θ.

Variational Autoencoder: Intractability

- Data likelihood: $p_\theta(\mathbf{x}) = \underbrace{\int}_{\text{intractible } \times} p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\,\mathbf{z}$

Variational Autoencoder: Intractability

- Data likelihood: $p_\theta(\mathbf{x}) = \underbrace{\int}_{\text{intractible } \times} p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\,\mathbf{z}$
- Posterior also intractable: $p_\theta(\mathbf{z} \mid \mathbf{x}) = \underbrace{p_\theta(\mathbf{x} \mid \mathbf{z})}_{\checkmark} \underbrace{p_\theta(\mathbf{z})}_{\checkmark} / \underbrace{p_\theta(\mathbf{x})}_{\times}$

Variational Autoencoder: Intractability

- Data likelihood: $p_\theta(\mathbf{x}) = \underbrace{\int}_{\text{intractible } \times} p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}$
- Posterior also intractable: $p_\theta(\mathbf{z} \mid \mathbf{x}) = \underbrace{p_\theta(\mathbf{x} \mid \mathbf{z})}_{\checkmark} \underbrace{p_\theta(\mathbf{z})}_{\checkmark} / \underbrace{p_\theta(\mathbf{x})}_{\times}$
- Solution:
  - In additional to define the decoder network $p_\theta(\mathbf{x} \mid \mathbf{z})$, define an additional *encoder network* $q_\phi(\mathbf{z} \mid \mathbf{x})$ that approximates the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ $\Rightarrow$ variational inference with variational distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$!.
  - We will see this allows us to derive a lower bound on the data likelihood, which can be optimized tractably.

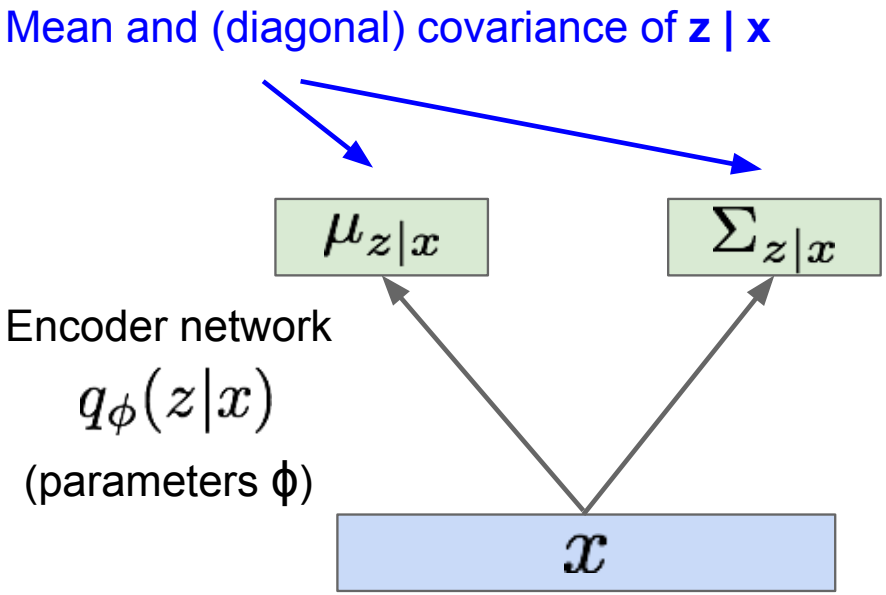
Variational Autoencoder: Intractability

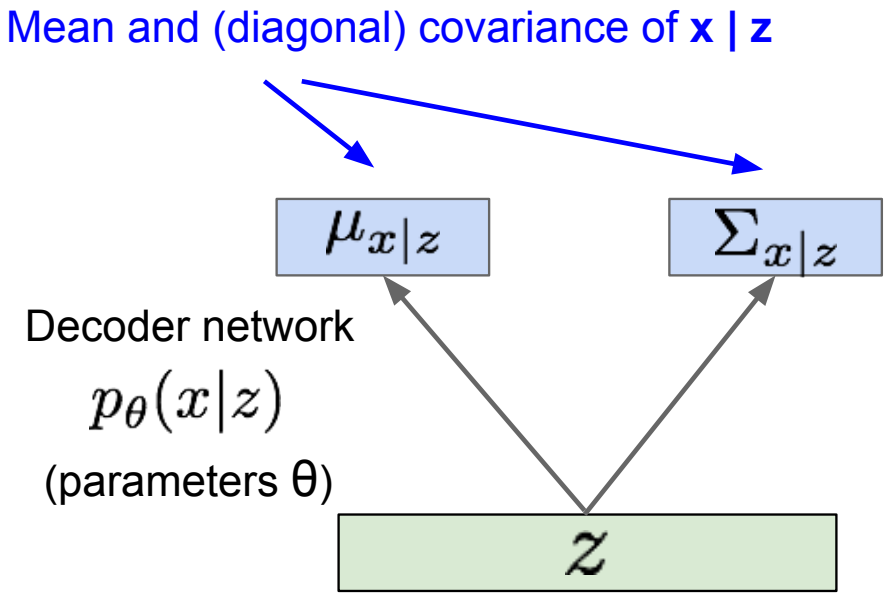
- Data likelihood: $p_\theta(\mathbf{x}) = \underbrace{\int}_{\text{intractible } \times} p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z}) \mathrm{d}\mathbf{z}$
- Posterior also intractable: $p_\theta(\mathbf{z} \mid \mathbf{x}) = \underbrace{p_\theta(\mathbf{x} \mid \mathbf{z})}_{\checkmark} \underbrace{p_\theta(\mathbf{z})}_{\checkmark} / \underbrace{p_\theta(\mathbf{x})}_{\times}$
- Solution:
  - In additional to define the decoder network $p_\theta(\mathbf{x} \mid \mathbf{z})$, define an additional *encoder network* $q_\phi(\mathbf{z} \mid \mathbf{x})$ that approximates the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ $\Rightarrow$ variational inference with variational distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$!.
  - We will see this allows us to derive a lower bound on the data likelihood, which can be optimized tractably.

Variational Autoencoder

Encoder and decoder networks are probabilistic

- Encoder also called inference/recognition network.
- Decoder also called generation network.
- Amortized structure.

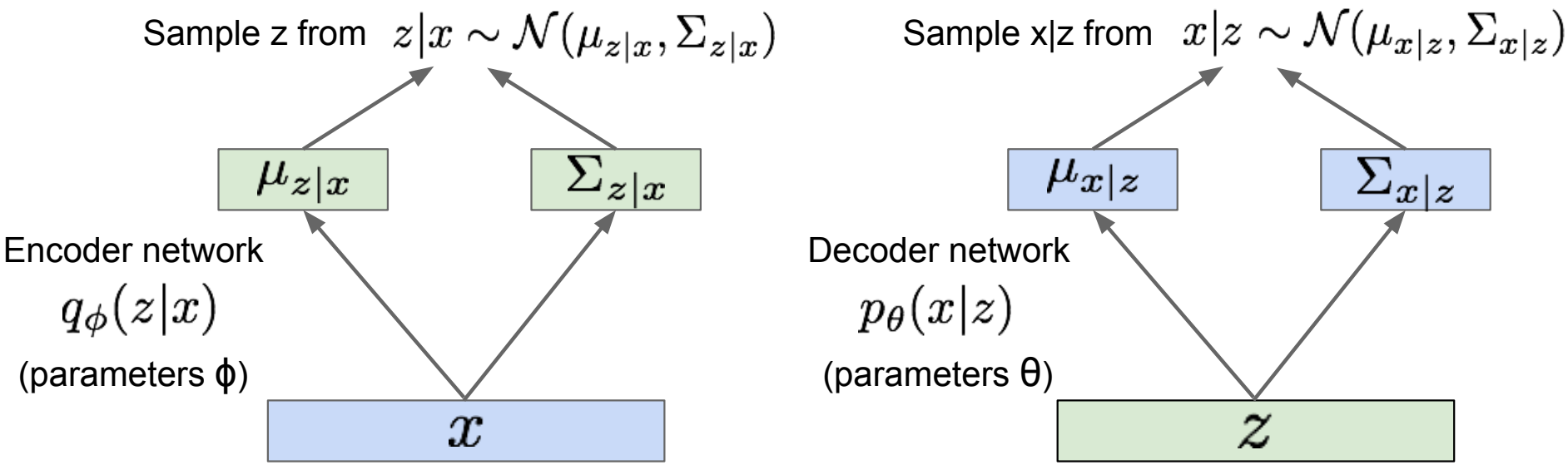




Variational Autoencoder

Encoder and decoder networks are probabilistic

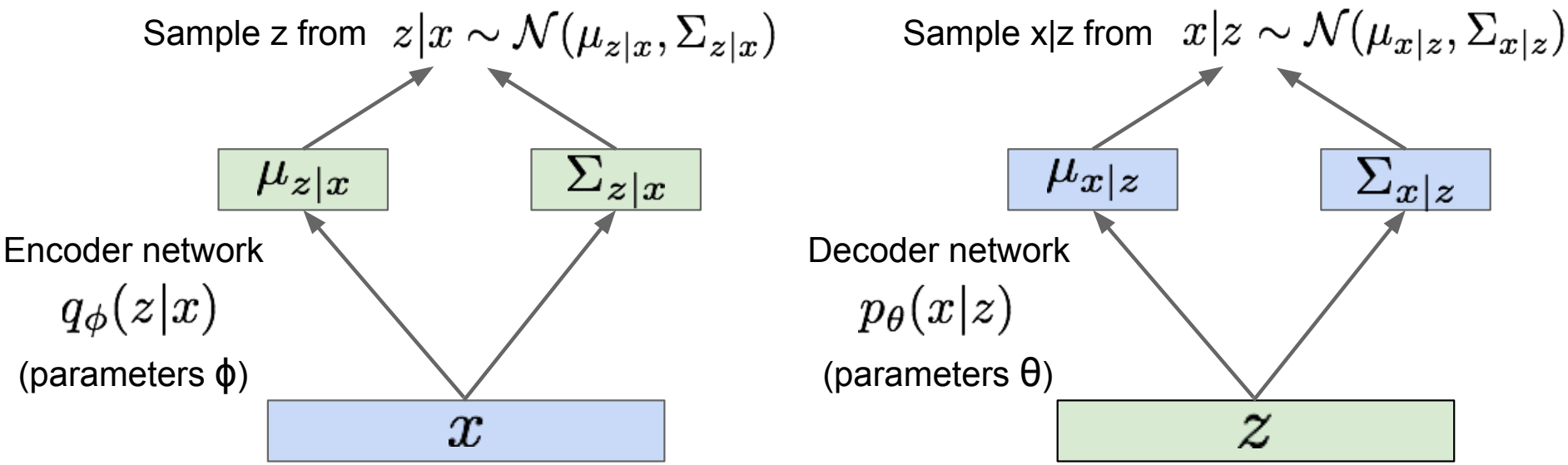
- Encoder also called inference/recognition network.
- Decoder also called generation network.
- Amortized structure.



Variational Autoencoder

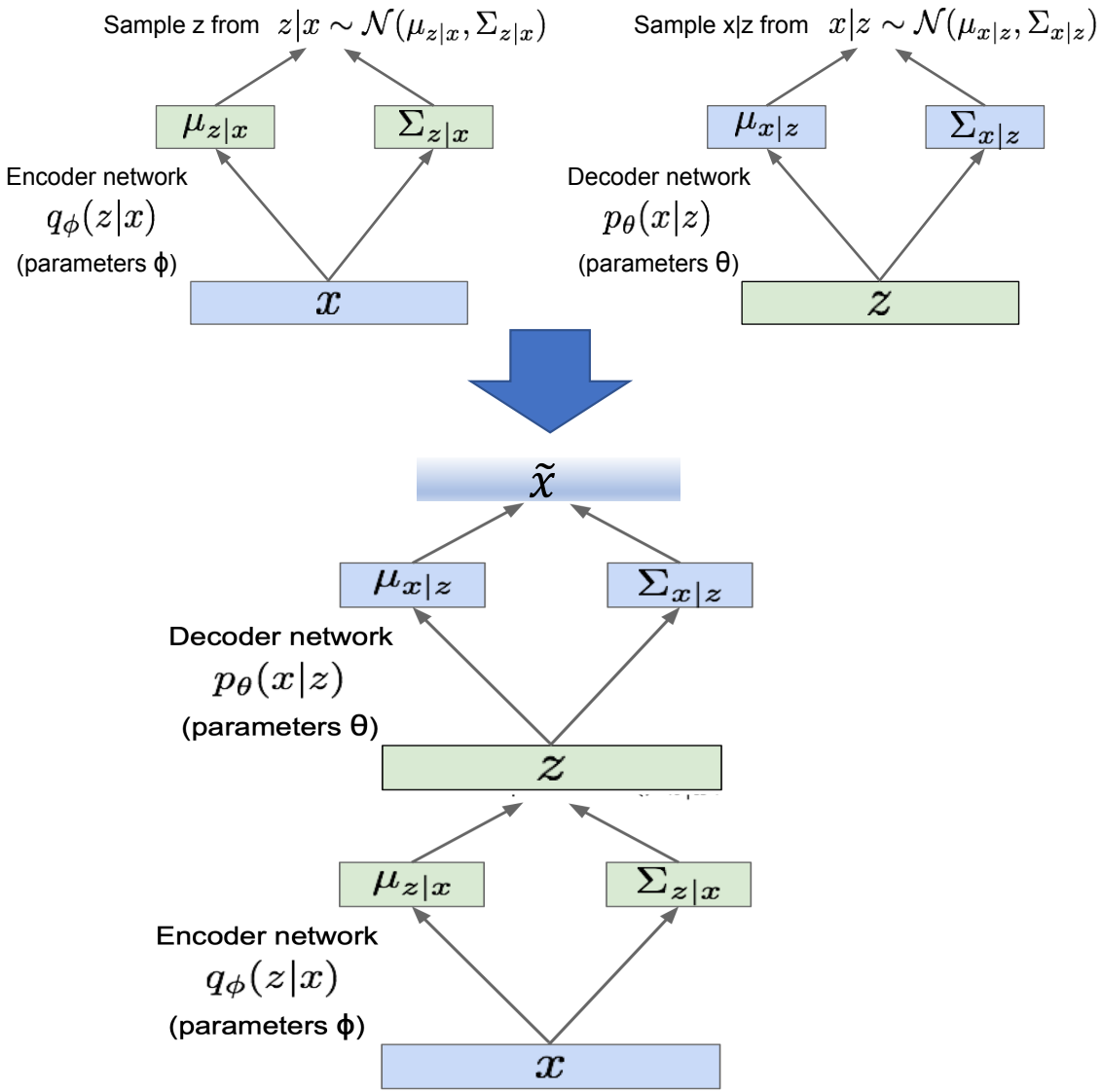
Encoder and decoder networks are probabilistic

- Encoder also called inference/recognition network.
- Decoder also called generation network.
- Amortized structure.



Want to match $q_\phi(z|x)$ and $p(z|x) \propto p(z)p(x|z)$.

Doesn't Look Like Autoencoder?

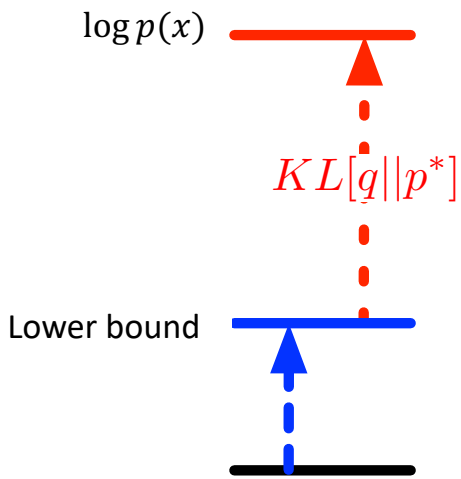


Variational Autoencoder: Log Data Likelihood

Derive from a different but equivalent way as in variational inference:

- Don't need to apply Jensen's inequality, or equivalently, prove Jensen's inequality.
- We will show: $\log p(x) = KL(q(z|x)||p(z|x)) + \text{lower bound}$

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$



Variational Autoencoder: Log Data Likelihood

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} \left[\log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

Taking expectation wrt. z (using encoder network) will come in handy later

Variational Autoencoder: Log Data Likelihood

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Bayes' Rule})$$

Variational Autoencoder: Log Data Likelihood

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} \left[\log p_\theta(x^{(i)}) \right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} \mid z) p_\theta(z)}{p_\theta(z \mid x^{(i)})} \right] && (\text{Bayes' Rule}) \\
&= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} \mid z) p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})} \right] && (\text{Multiply by constant})
\end{aligned}
$$

Variational Autoencoder: Log Data Likelihood

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)} \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)} \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}
\end{aligned}
$$

Variational Autoencoder: Log Data Likelihood

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

Variational Autoencoder: Log Data Likelihood

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

The expectation wrt. z (using encoder network) let us write nice KL terms

Variational Autoencoder: Log Data Likelihood

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad \text{(Multiply by constant)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad \text{(Logarithms)}$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.

Variational Autoencoder: Log Data Likelihood

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}
\end{aligned}
$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

Variational Autoencoder: Log Data Likelihood

$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] && (p_\theta(x^{(i)}) \text{ Does not depend on } z)\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] && \text{(Bayes' Rule)}\\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] && \text{(Multiply by constant)}\\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] && \text{(Logarithms)}\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{>0}
\end{aligned}
$$

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)},\theta,\phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)},\theta,\phi)$$

Training: Maximize lower bound

Variational Autoencoder: Log Data Likelihood

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{>0}$$

Reconstruct the input data

Make approximate posterior distribution close to prior

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)},\theta,\phi)$$

Variational lower bound ("ELBO")

$$\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)},\theta,\phi)$$

Training: Maximize lower bound

Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

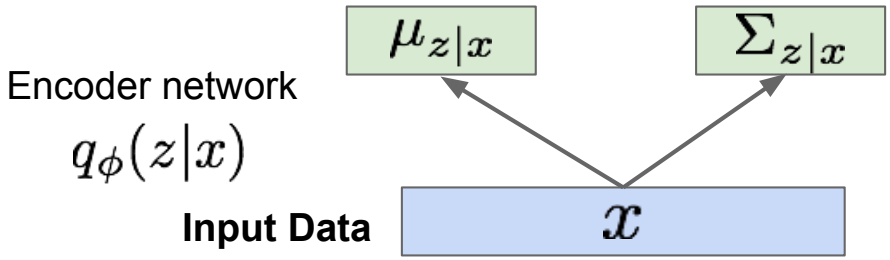
Let's look at computing the bound (forward pass) for a given minibatch of input data

Input Data x

Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

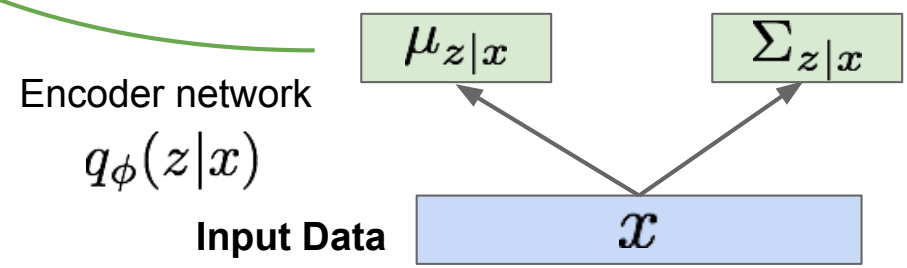


Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Make approximate posterior distribution close to prior

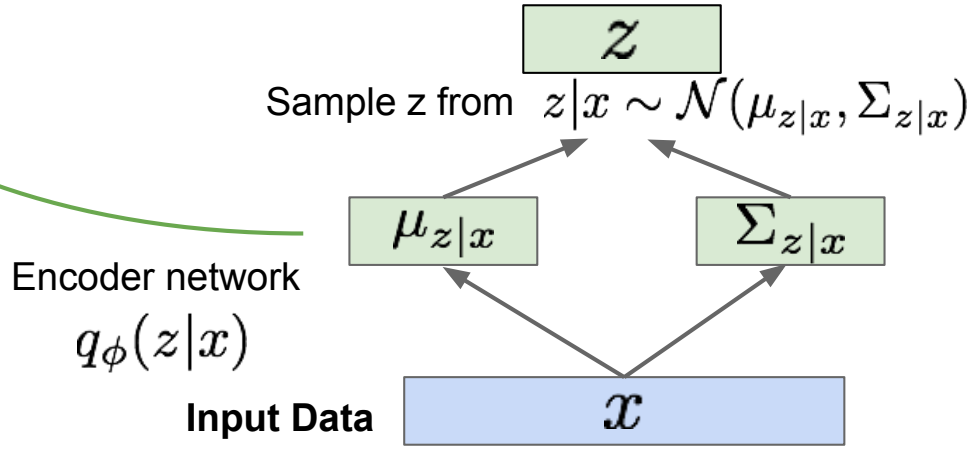


Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

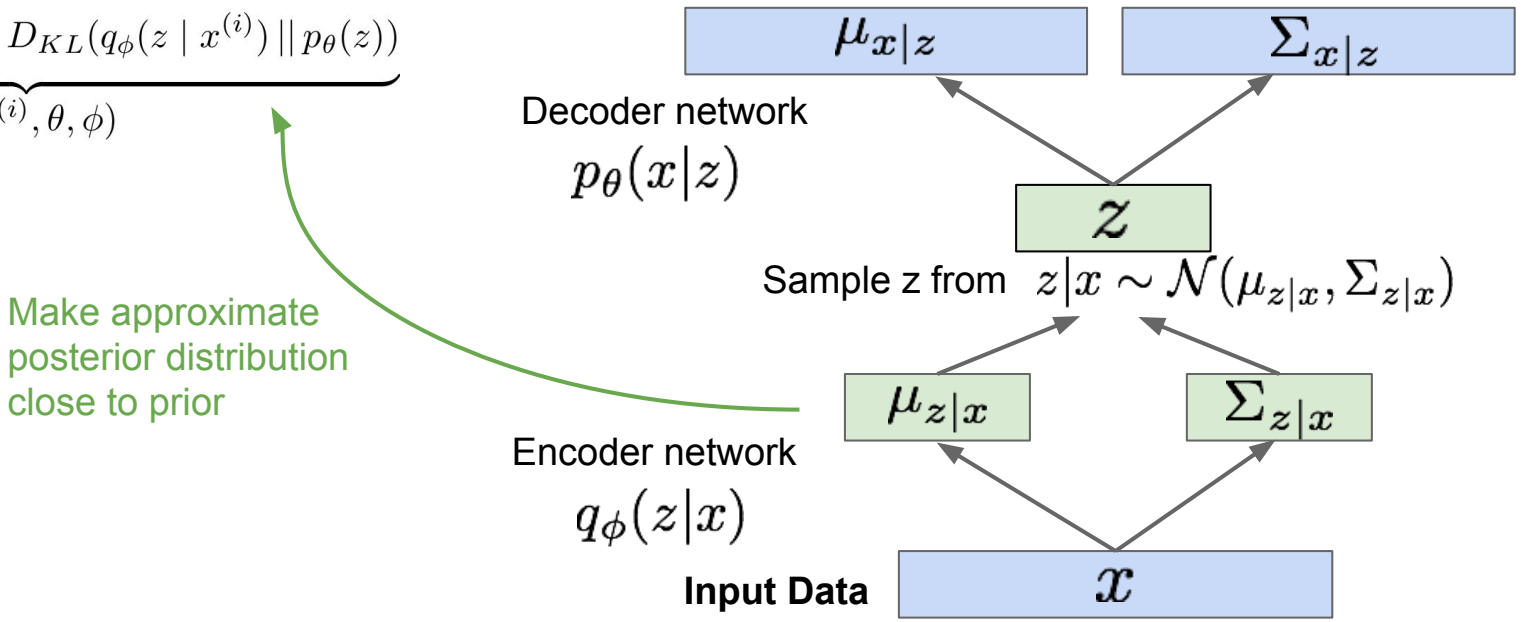
Make approximate posterior distribution close to prior



Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

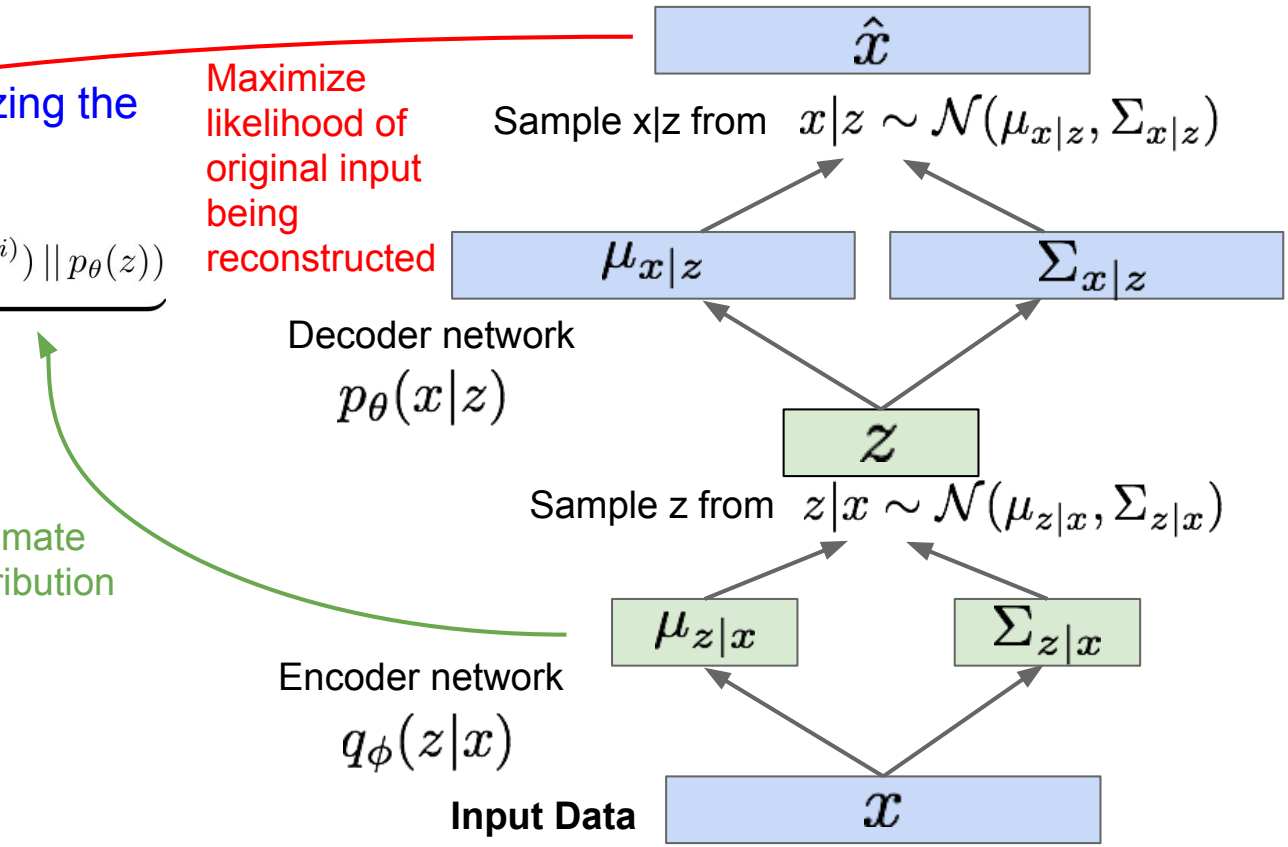


Variational Autoencoder: Log Data Likelihood

Putting it all together: maximizing the likelihood lower bound

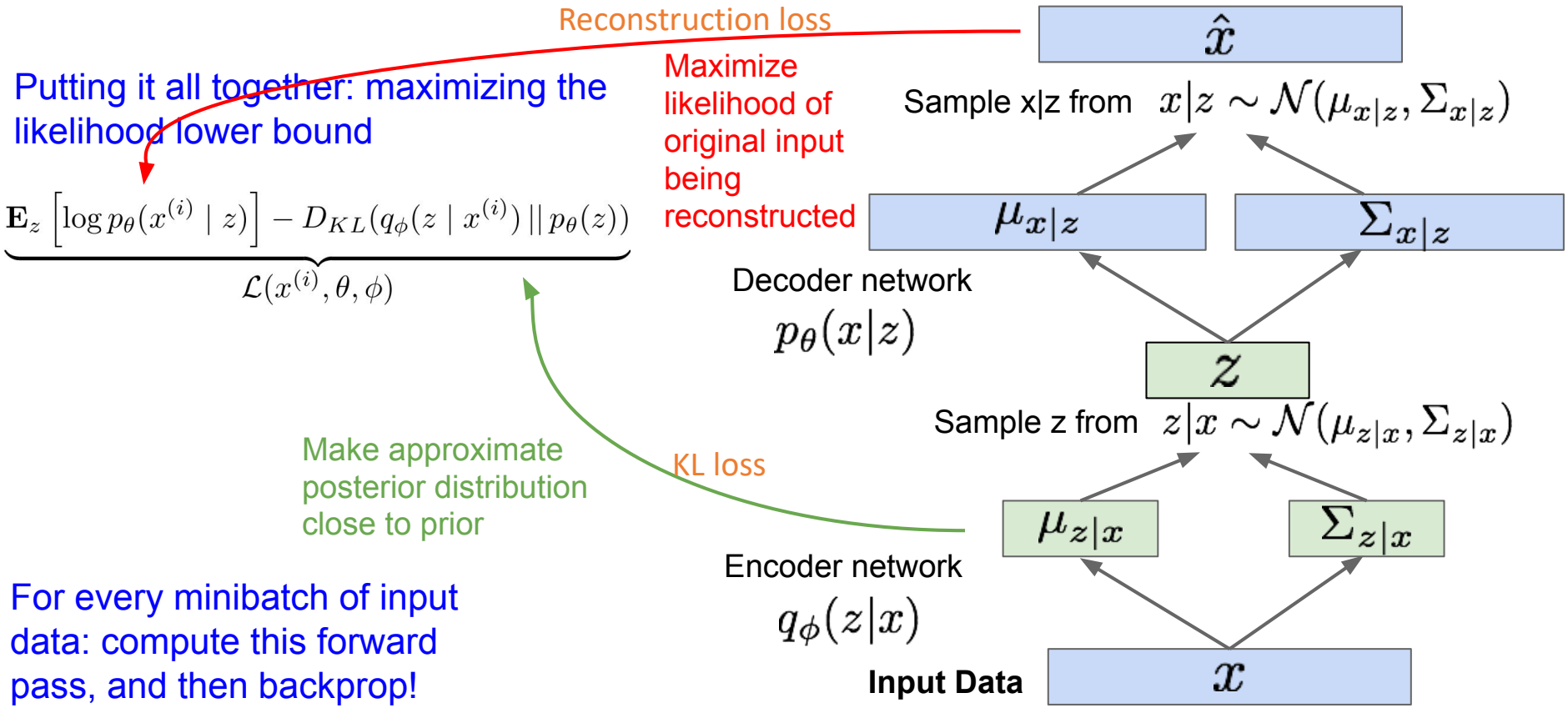
$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed



Make approximate posterior distribution close to prior

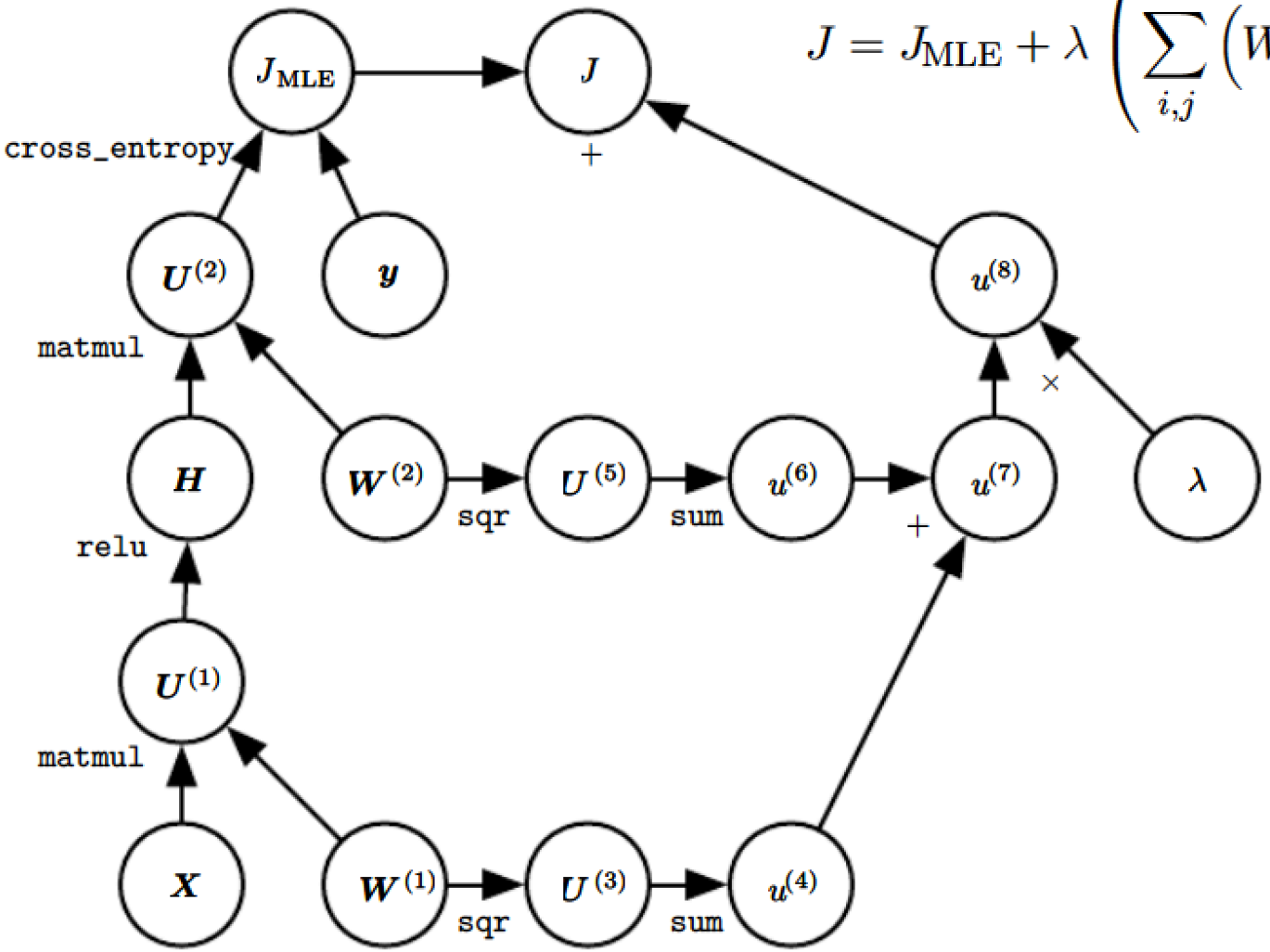
Variational Autoencoder: Log Data Likelihood



Recap: FNN Computational Graph

Total cost:

$$J = J_{\text{MLE}} + \lambda \left(\sum_{i,j} \left(W^{(1)}_{i,j} \right)^2 + \sum_{i,j} \left(W^{(2)}_{i,j} \right)^2 \right)$$

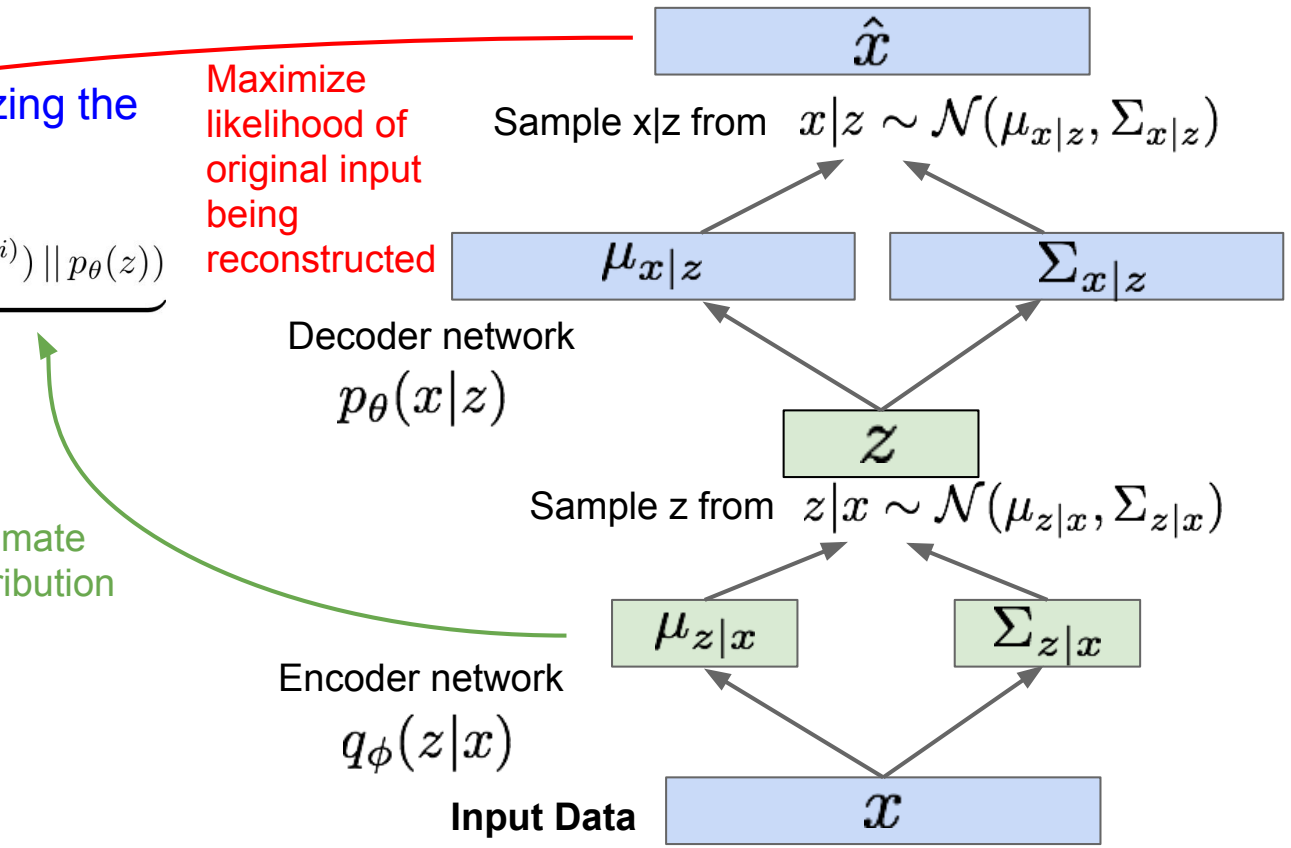


Recap: VAE Computational Graph

Putting it all together: maximizing the likelihood lower bound

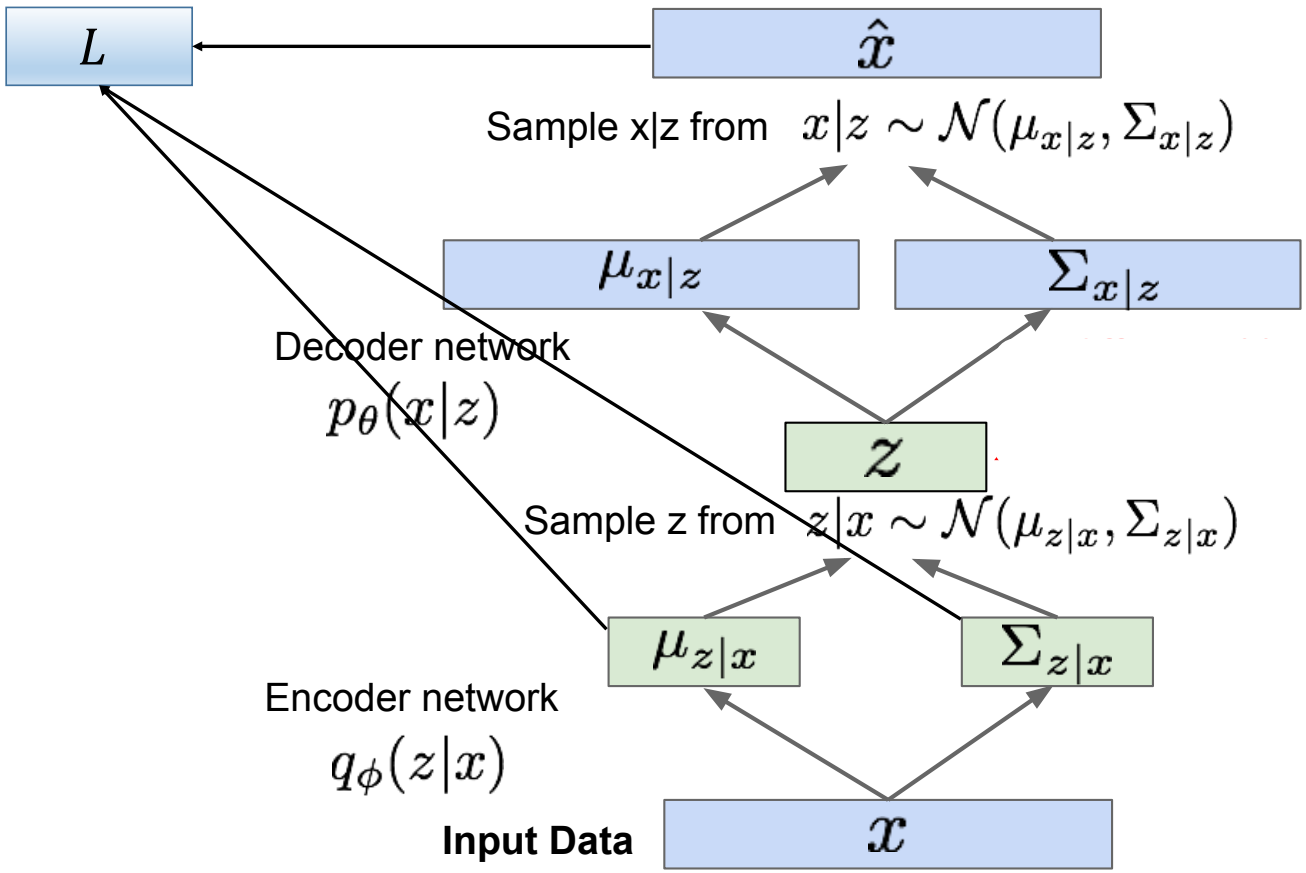
$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed



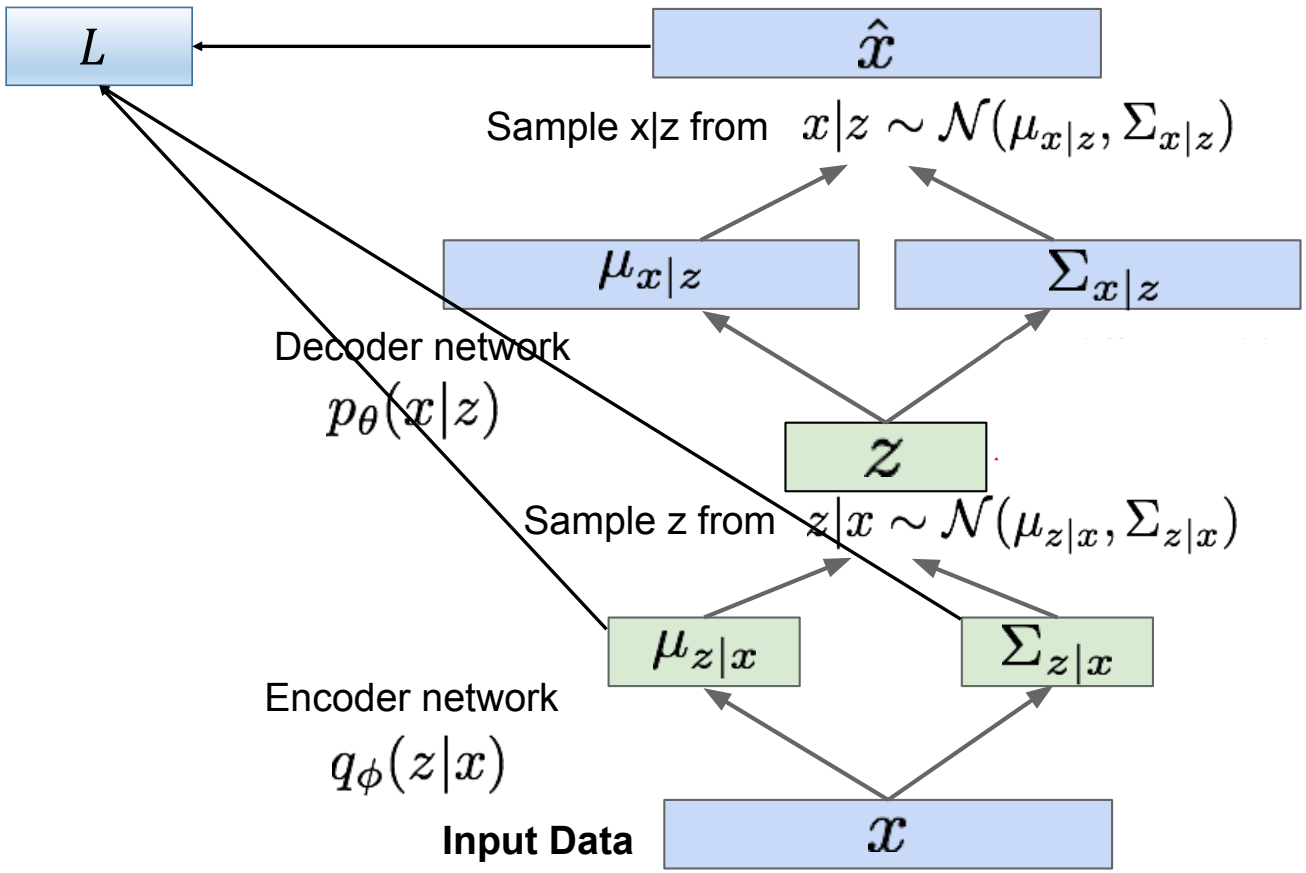
Make approximate posterior distribution close to prior

Variational Autoencoder: How to Calculate Gradients?



- Can we directly apply BP?

Variational Autoencoder: How to Calculate Gradients?



- Can we directly apply BP?

No!

Variational Autoencoder: Reparameterization Trick

- In the computational graph, we have

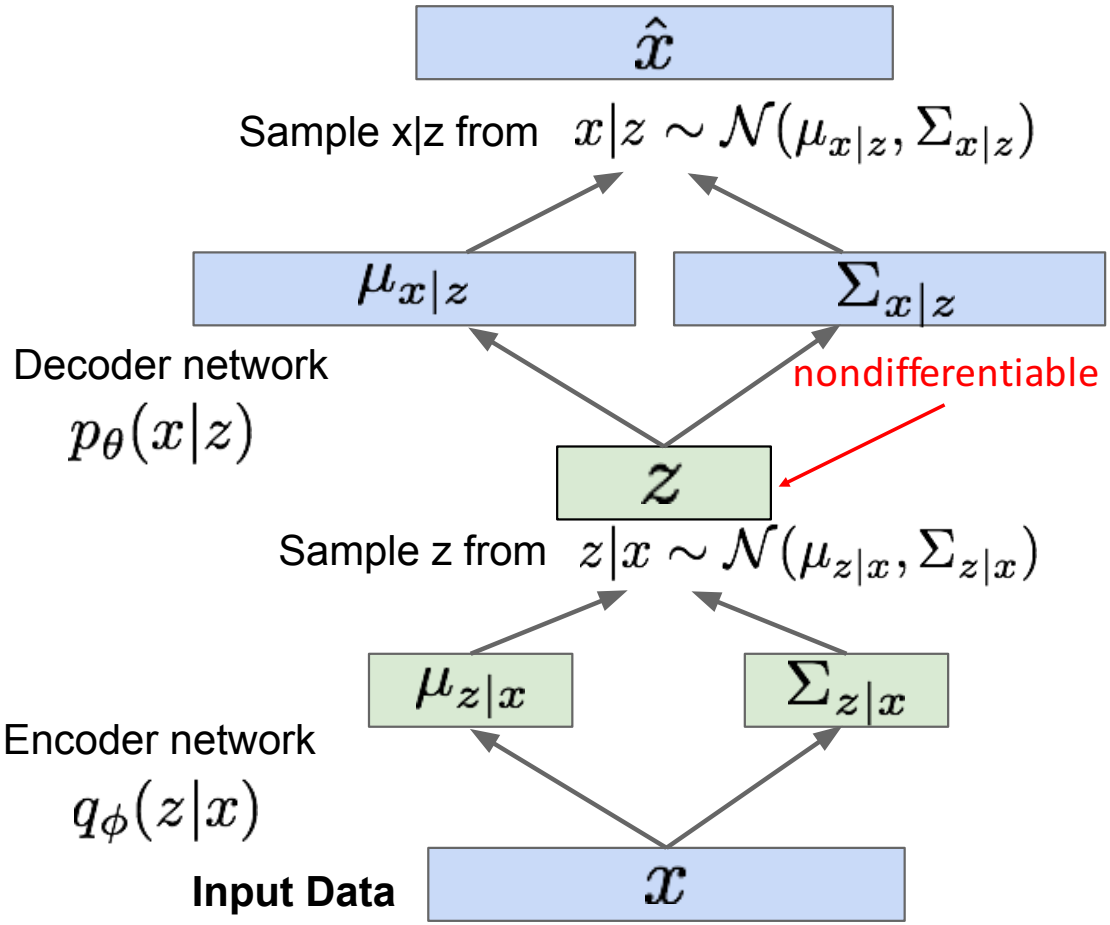
$$\mathbf{z} \sim \mathcal{N}\left(\mu_{\mathbf{z}|\mathbf{x}}, \Sigma_{\mathbf{z}|\mathbf{x}}\right) .$$

- To use BP, we need to compute $\frac{\partial \mathbf{z}}{\partial \mu_{\mathbf{z}|\mathbf{x}}}$ and $\frac{\partial \mathbf{z}}{\partial \Sigma_{\mathbf{z}|\mathbf{x}}}$:
  - nondifferentiable!
- Reparameterization:

$$\mathbf{z} = \mu_{\mathbf{z}|\mathbf{x}} + \Sigma_{\mathbf{z}|\mathbf{x}} \xi$$

$$\xi \sim \mathcal{N}(0, 1)$$

  - differentiable w.r.t. both $\mu_{\mathbf{z}|\mathbf{x}}$ and $\Sigma_{\mathbf{z}|\mathbf{x}}$



Variational Autoencoder: Reparameterization Trick

- In the computational graph, we have

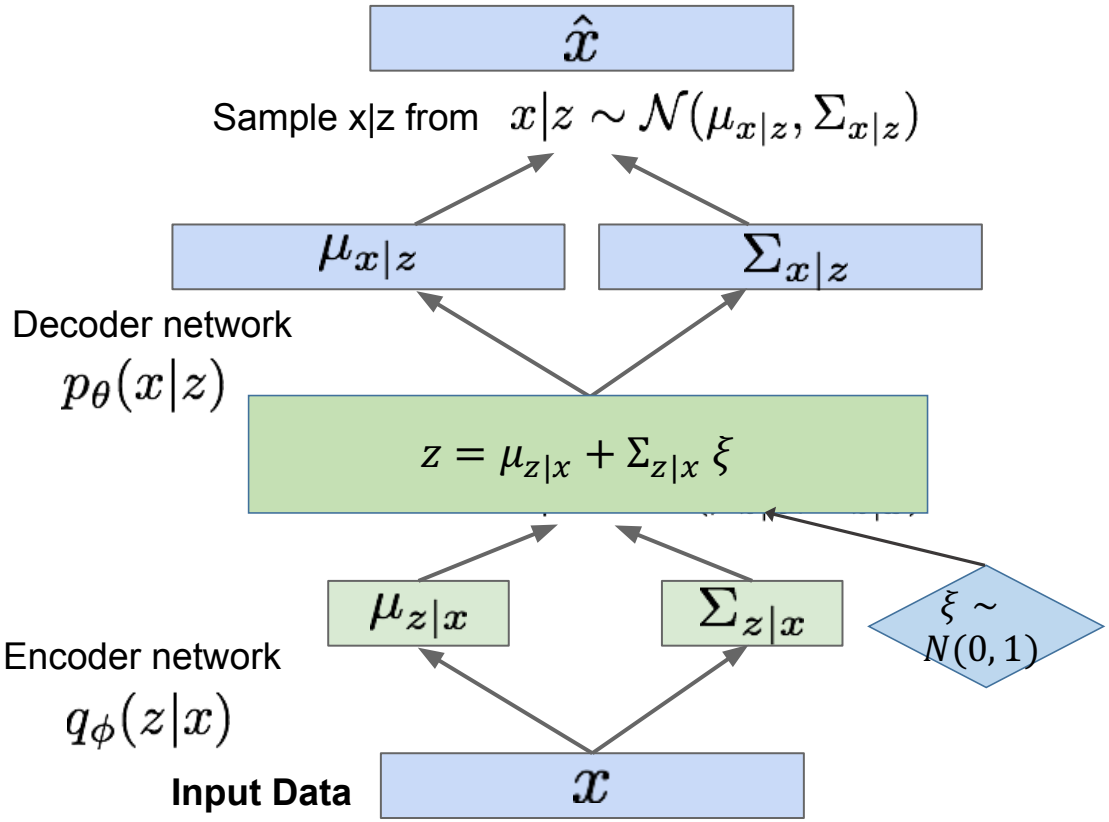
$$\mathbf{z} \sim \mathcal{N}\left(\mu_{\mathbf{z}|\mathbf{x}}, \Sigma_{\mathbf{z}|\mathbf{x}}\right) .$$

- To use BP, we need to compute $\frac{\partial \mathbf{z}}{\partial \mu_{\mathbf{z}|\mathbf{x}}}$ and $\frac{\partial \mathbf{z}}{\partial \Sigma_{\mathbf{z}|\mathbf{x}}}$:
  - nondifferentiable!
- Reparameterization:

$$\mathbf{z} = \mu_{\mathbf{z}|\mathbf{x}} + \Sigma_{\mathbf{z}|\mathbf{x}} \xi$$

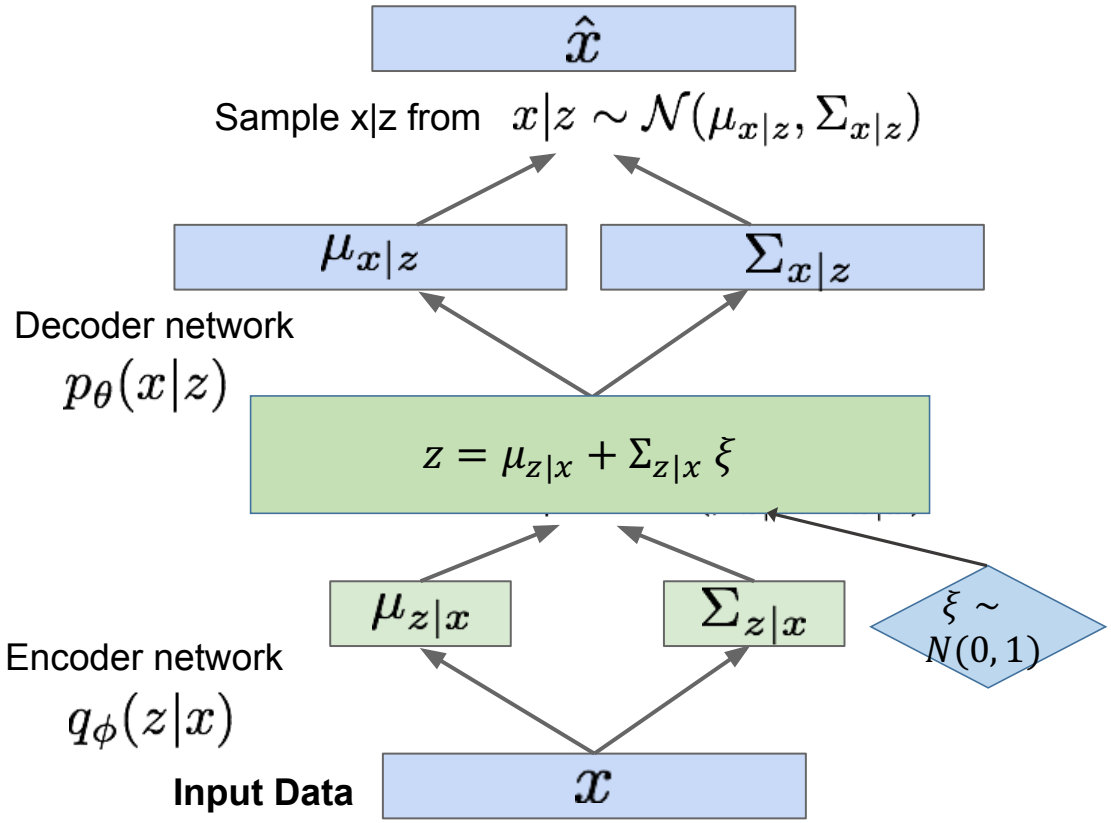
$$\xi \sim \mathcal{N}(0, 1)$$

  - differentiable w.r.t. both $\mu_{\mathbf{z}|\mathbf{x}}$ and $\Sigma_{\mathbf{z}|\mathbf{x}}$



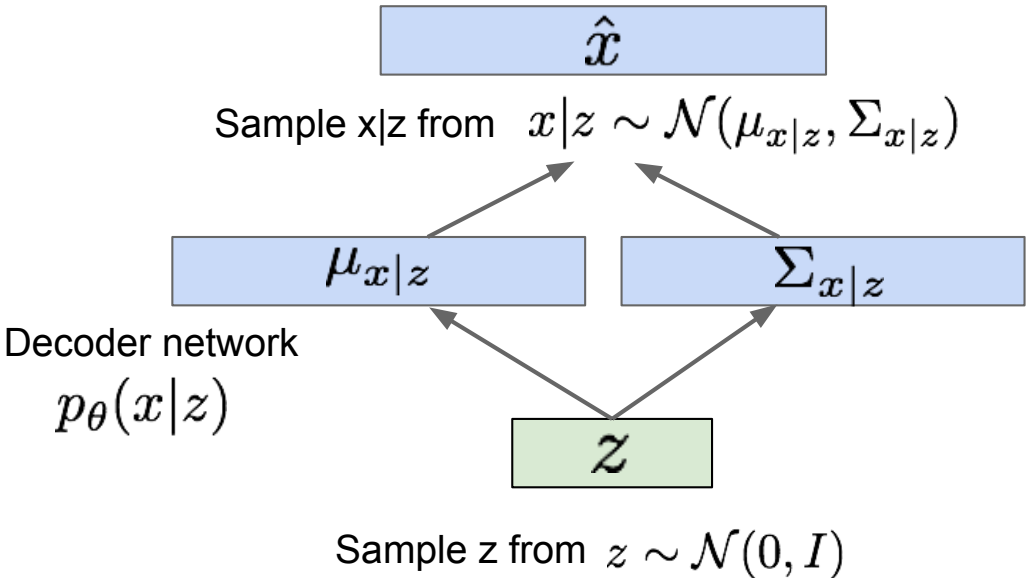
Variational Autoencoder: Implementation

- Define a forward graph as shown on the right.
- Define the loss as the ELBO.
- Associate the loss with an optimizer.



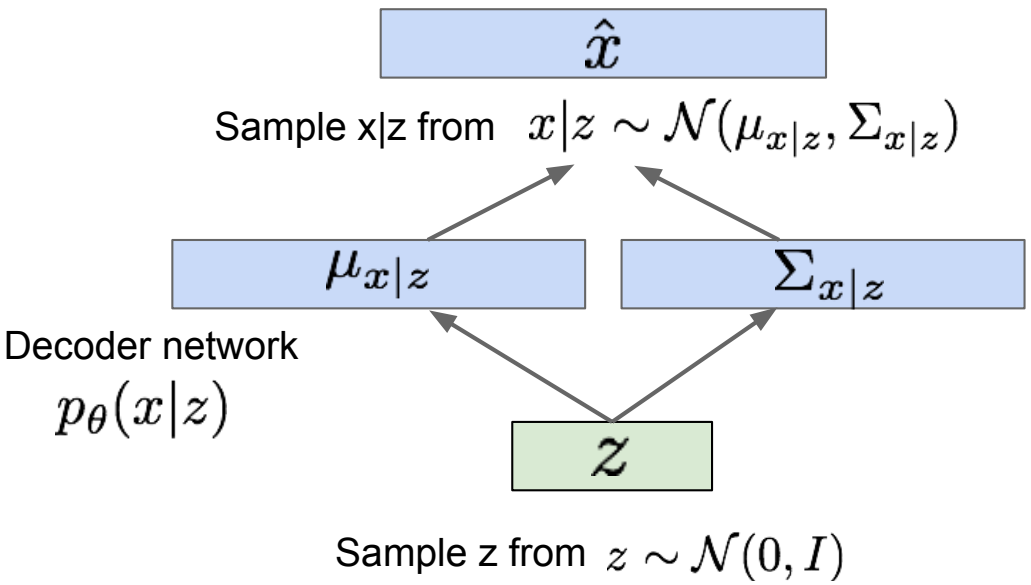
Variational Autoencoder: Generating Data

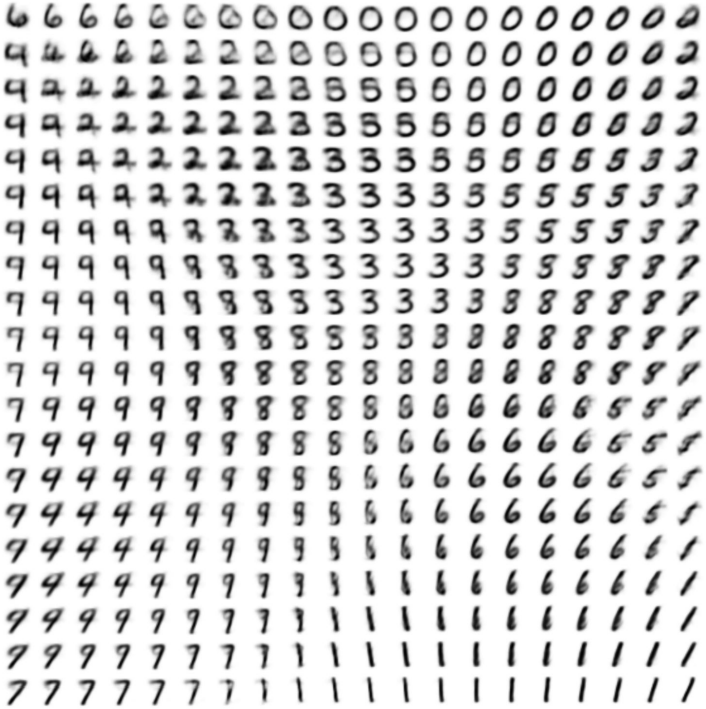
Use deconder network. Sample z from the prior.



Variational Autoencoder: Generating Data

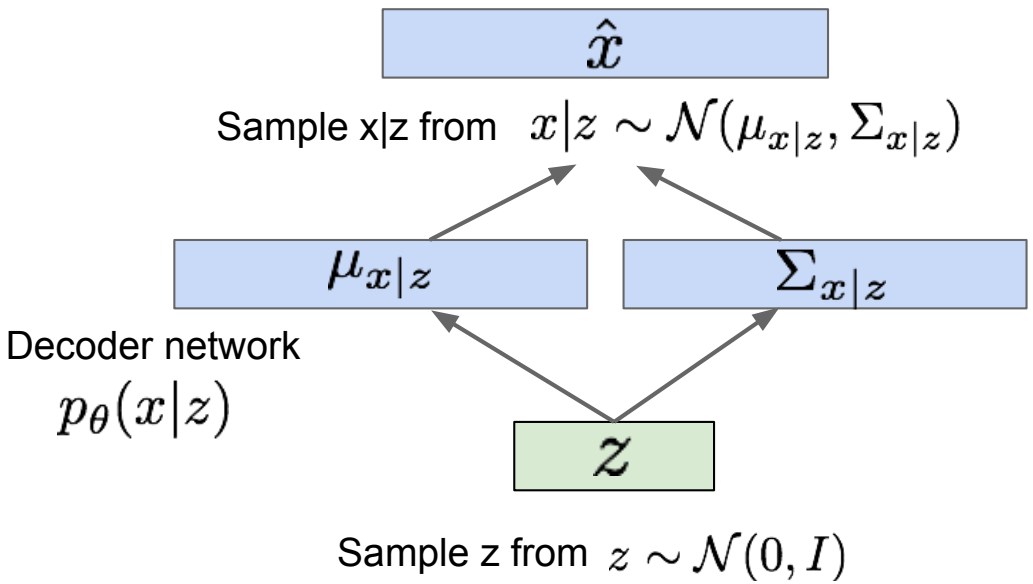
Use deconder network. Sample z from the prior.



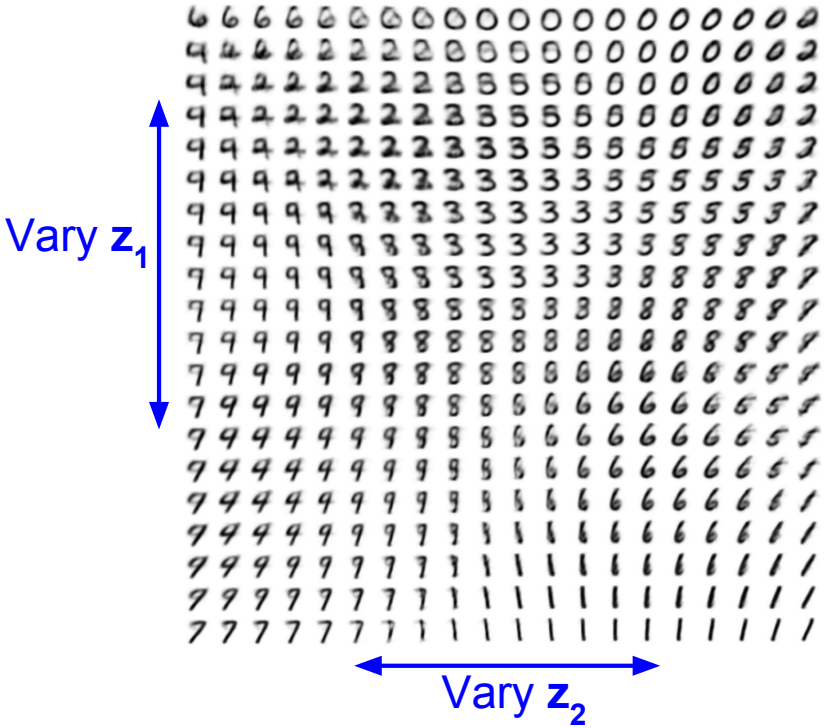


Variational Autoencoder: Generating Data

Use deconder network. Sample z from the prior.



Data manifold for 2-d **z**



Extension: Conditioned VAE

ELBO of standard VAE

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

- Can I ask VAE to generate a specific type of data, *e.g.*, generate an image of type "9"?
  - No!
- Solution: extend VAE to conditioned VAE.

Extension: Conditioned VAE

ELBO of standard VAE

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

- Can I ask VAE to generate a specific type of data, *e.g.*, generate an image of type "9"?
  - No!
- Solution: extend VAE to conditioned VAE.

Extension: Conditioned VAE

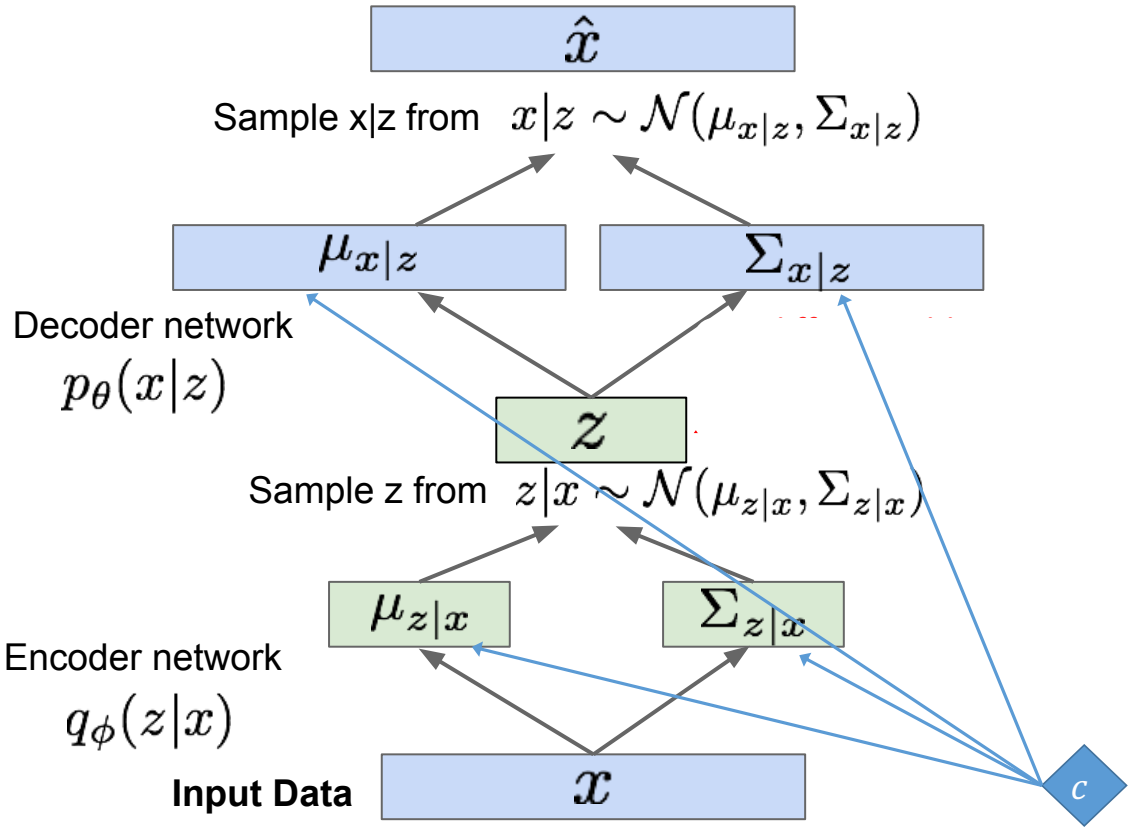
ELBO of standard VAE

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

- Can I ask VAE to generate a specific type of data, *e.g.*, generate an image of type "9"?
  - No!
- Solution: extend VAE to conditioned VAE.

Extension: Conditioned VAE

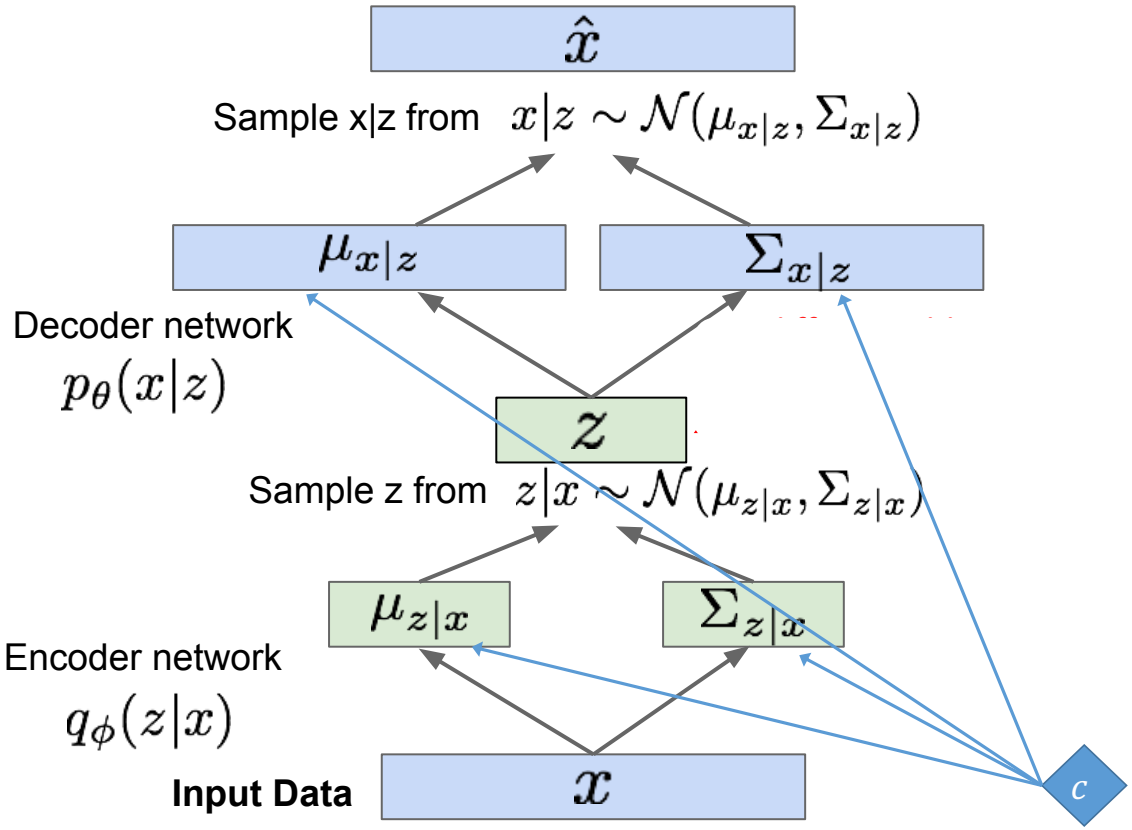
1. Implement by conditioning the encoder and decoder to something:
   - Represented by some variable **c**, *e.g.*, it could be a one-hot vector to represent a class.



Extension: Conditioned VAE

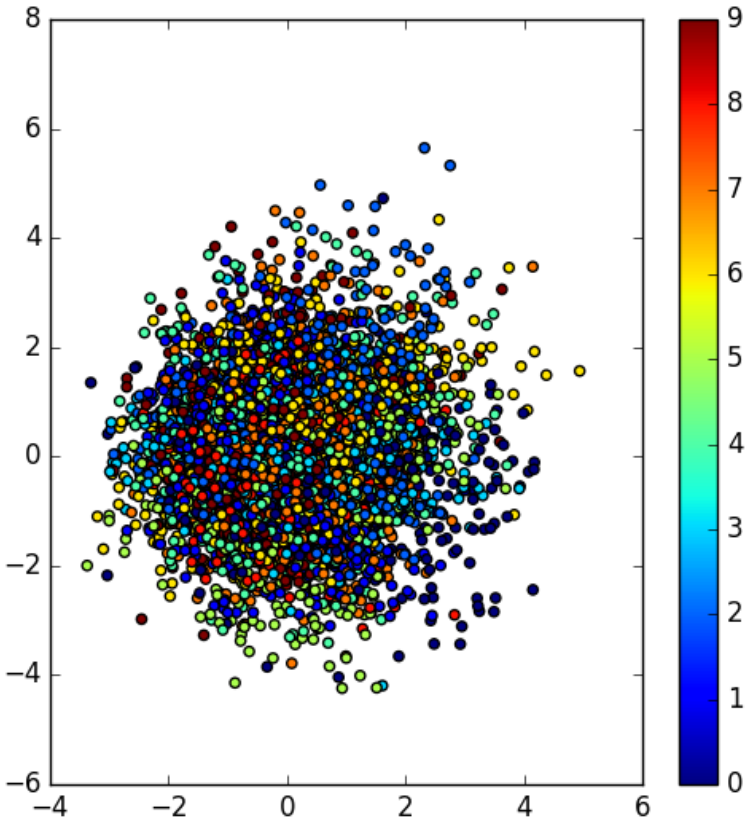
1. Implement by conditioning the encoder and decoder to something:

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



Conditional VAE on MNIST

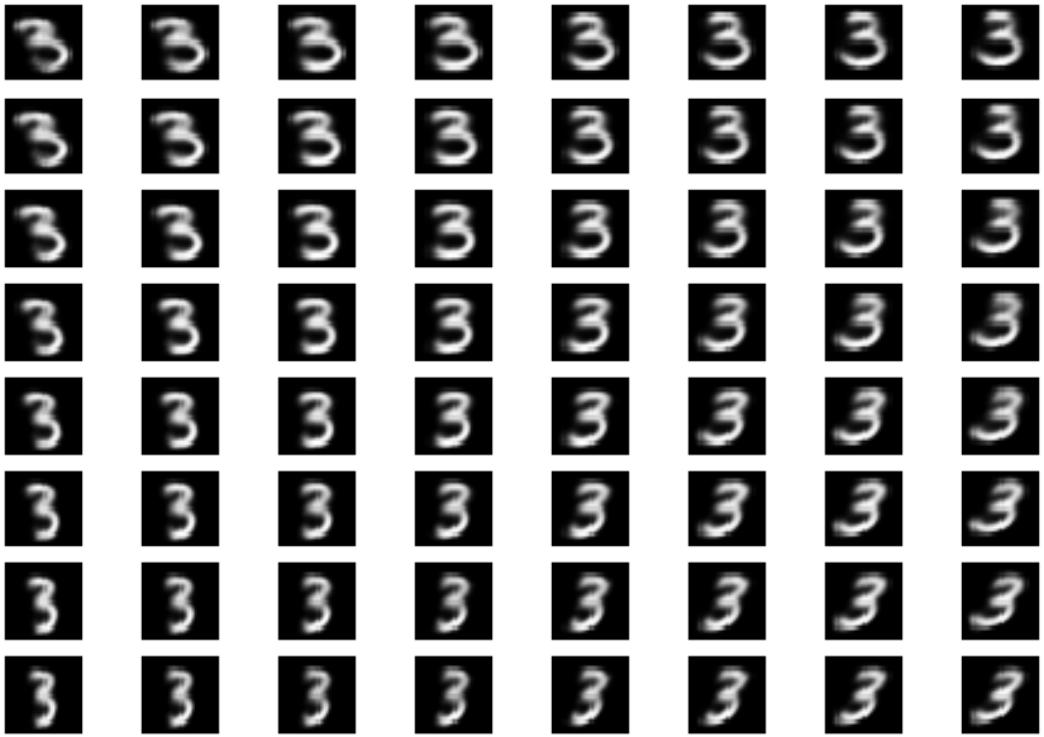
- Assume $p_\theta(\mathbf{z} \mid \mathbf{c}) = \mathcal{N}(0, 1)$ given $\mathbf{c}$.
- The figure shows the learned $q_\phi(\mathbf{z} \mid \mathbf{x}, \mathbf{c})$ for each class (different colors)[5].



[5] Image credit: https://wiseodd.github.io/techblog/2016/12/17/conditional-vae/

Conditional VAE on MNIST: Generating "3"[6]

- Set $\mathbf{c}$ to be a one-hot vector representing 3.
- Changing z_1 (on the y-axis) makes the digit style becomes narrower.
- Varying z_2 (on the x-axis) appears to rotate the digit slightly and elongate the lower portion in relation to the upper portion.



[6] Image credit: http://nnormandin.com/science/2017/07/01/cvae.html

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

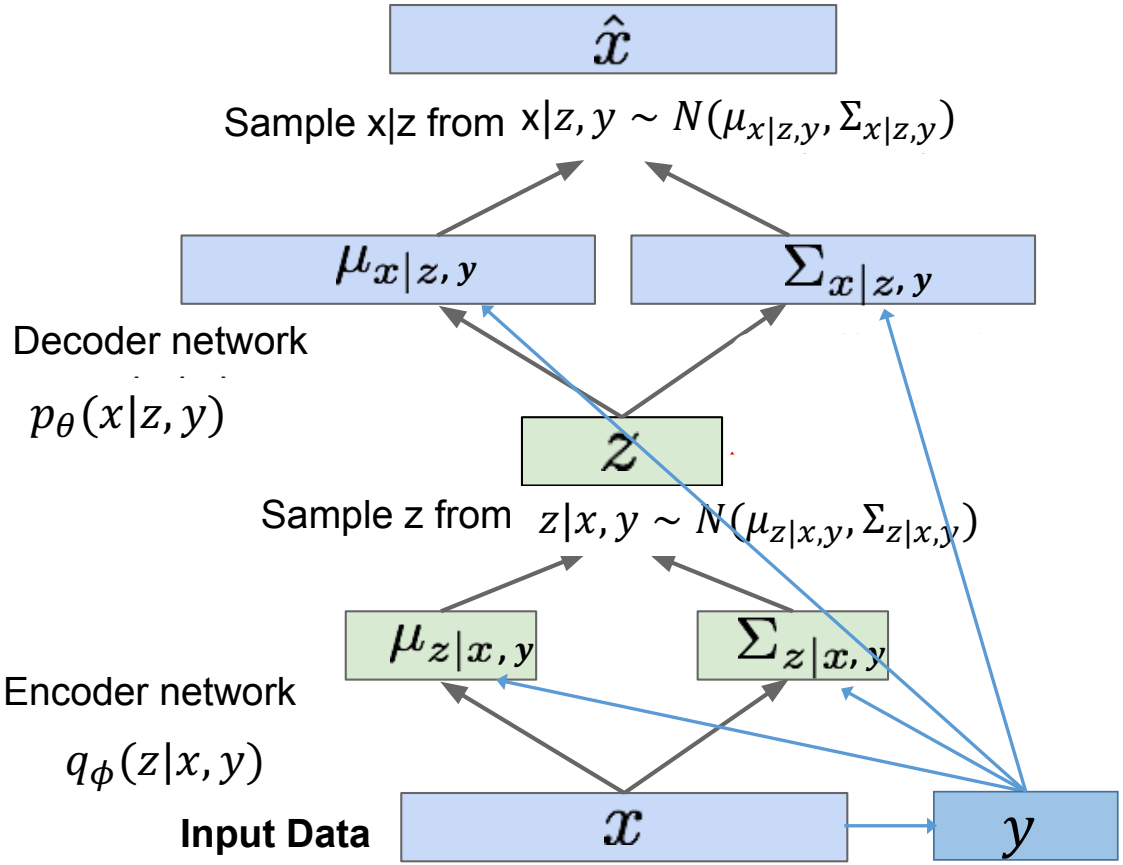
$$p(y) = \text{Cat}(y|\pi), \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}),$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_{\mathbf{x}|\mathbf{z},y}, \Sigma_{\mathbf{x}|\mathbf{z},y}\right)$$

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:



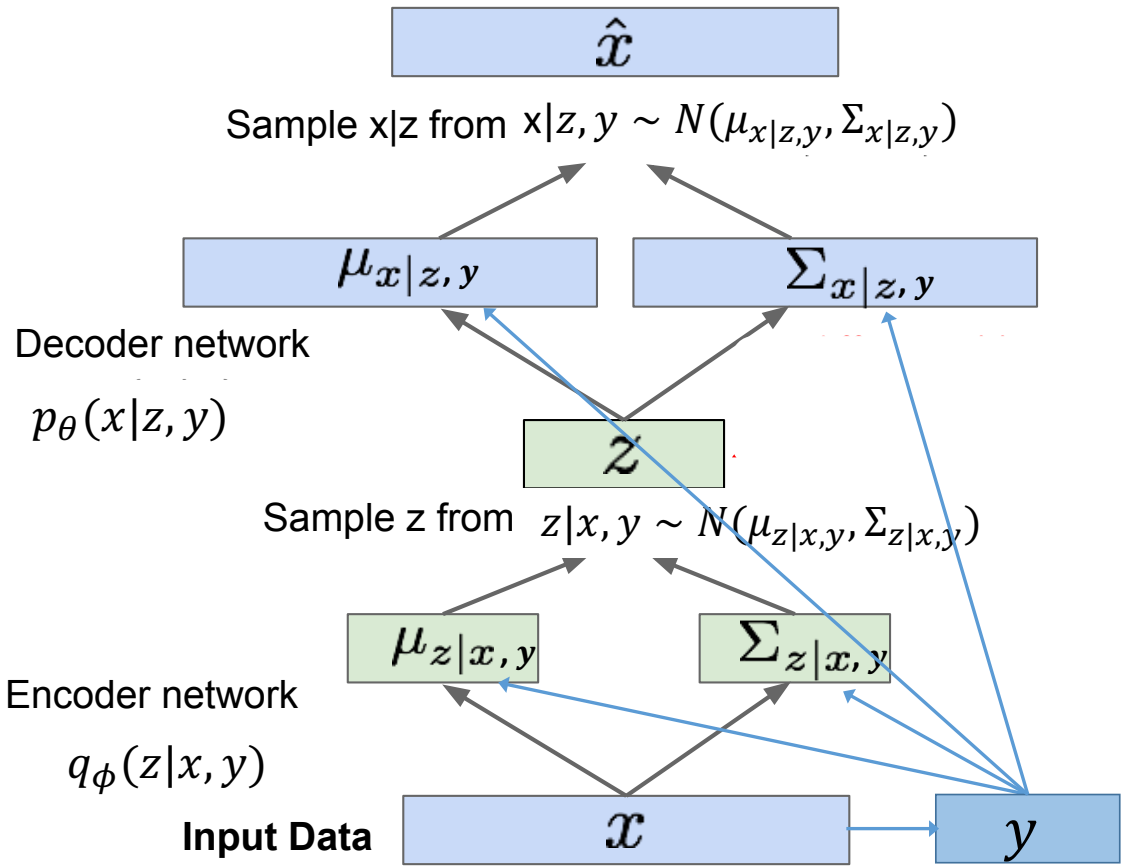
G: $p(y) = \text{Cat}(y|\pi), p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}), p_\theta(\mathbf{x}|y,\mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_{\mathbf{x}|\mathbf{z},y}, \Sigma_{\mathbf{x}|\mathbf{z},y}\right)$

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:

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



Extension: Semi-supervised VAE (Optional)

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 (Optional)

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 (Optional)

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:

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

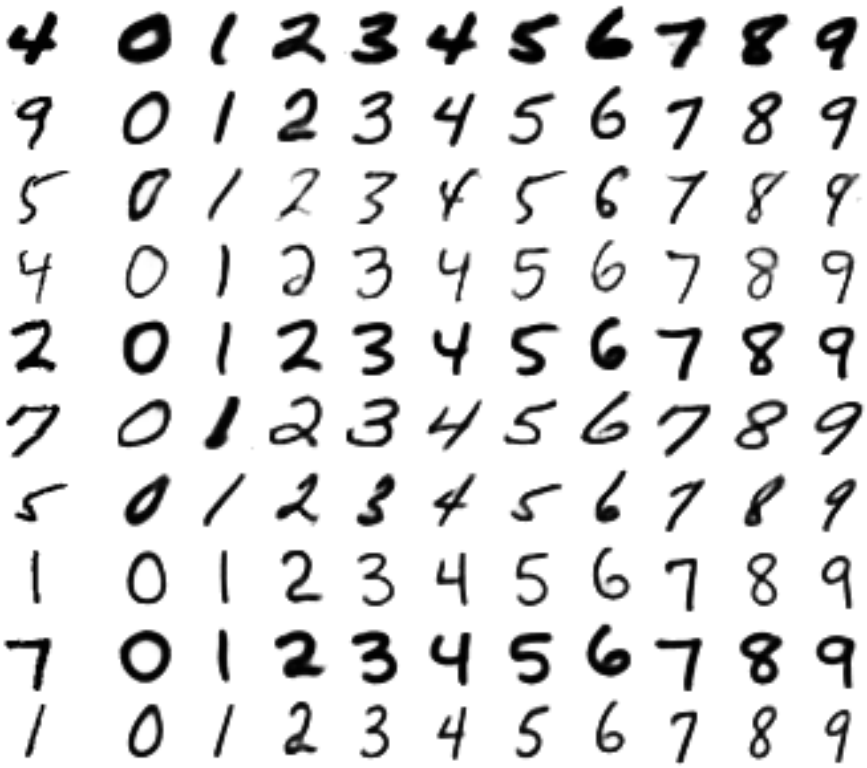
Total loss:

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

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)

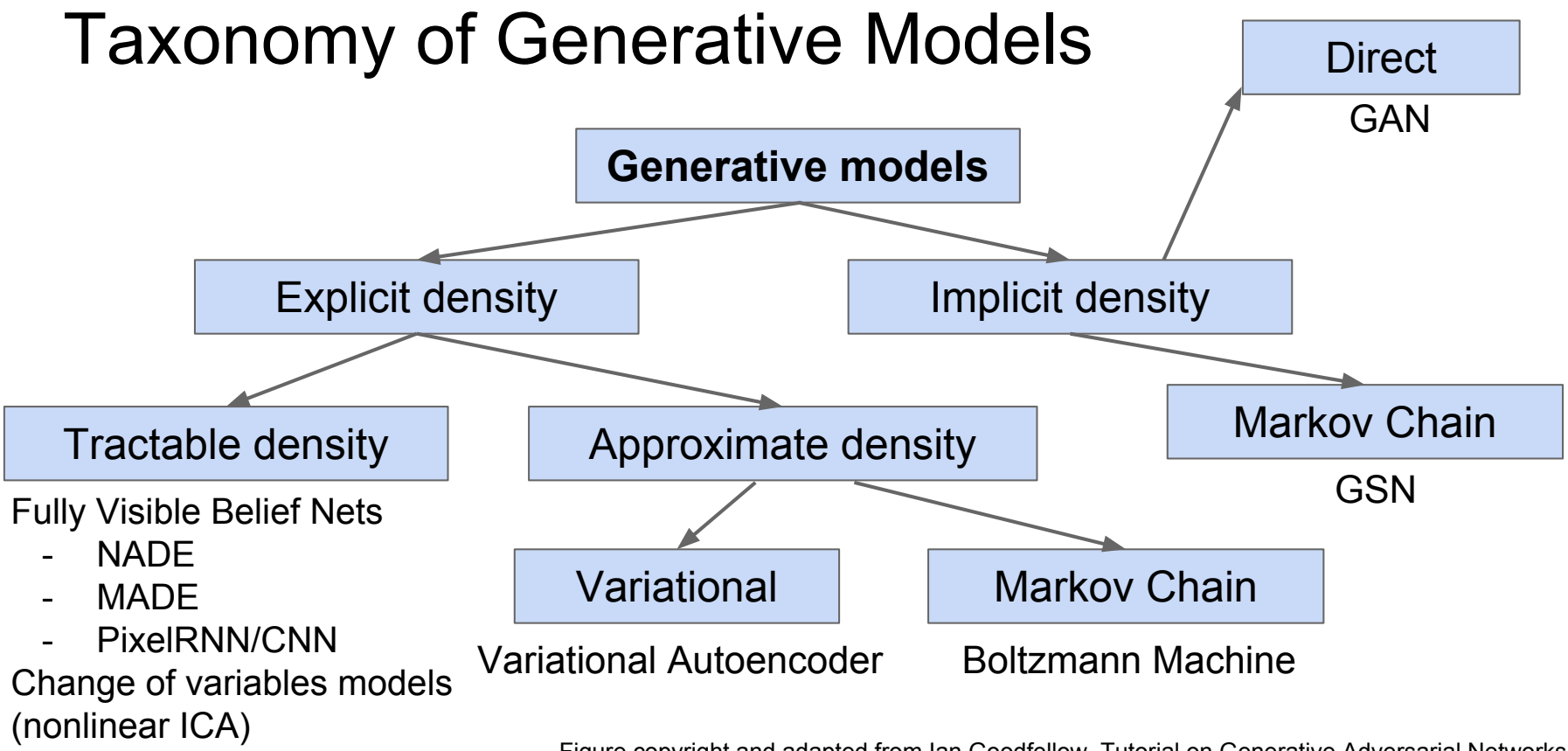


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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:

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

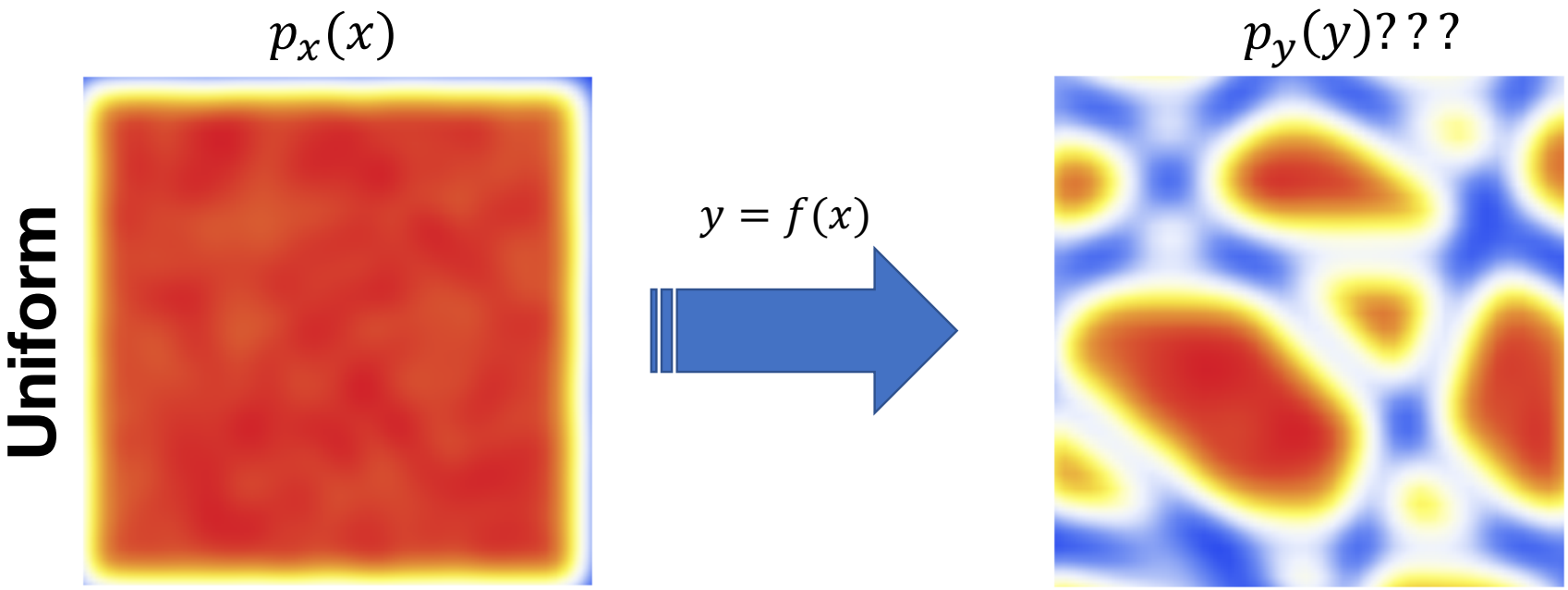
2. VAEs introduce latent variable $\mathbf{z}$ into the model; the likelihood is an intractable integration:

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

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?

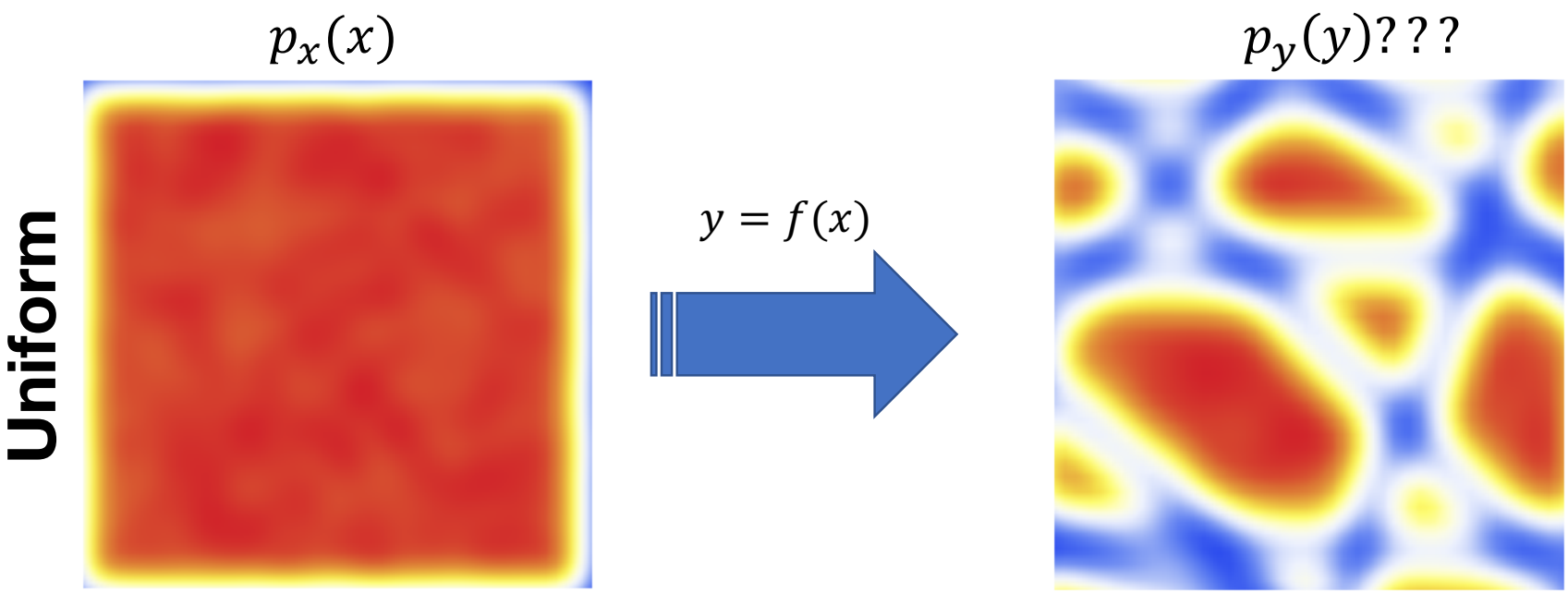
Solution: First sample from a simple distribution, *e.g.* random noise, then learn a transformation to the training distribution.



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

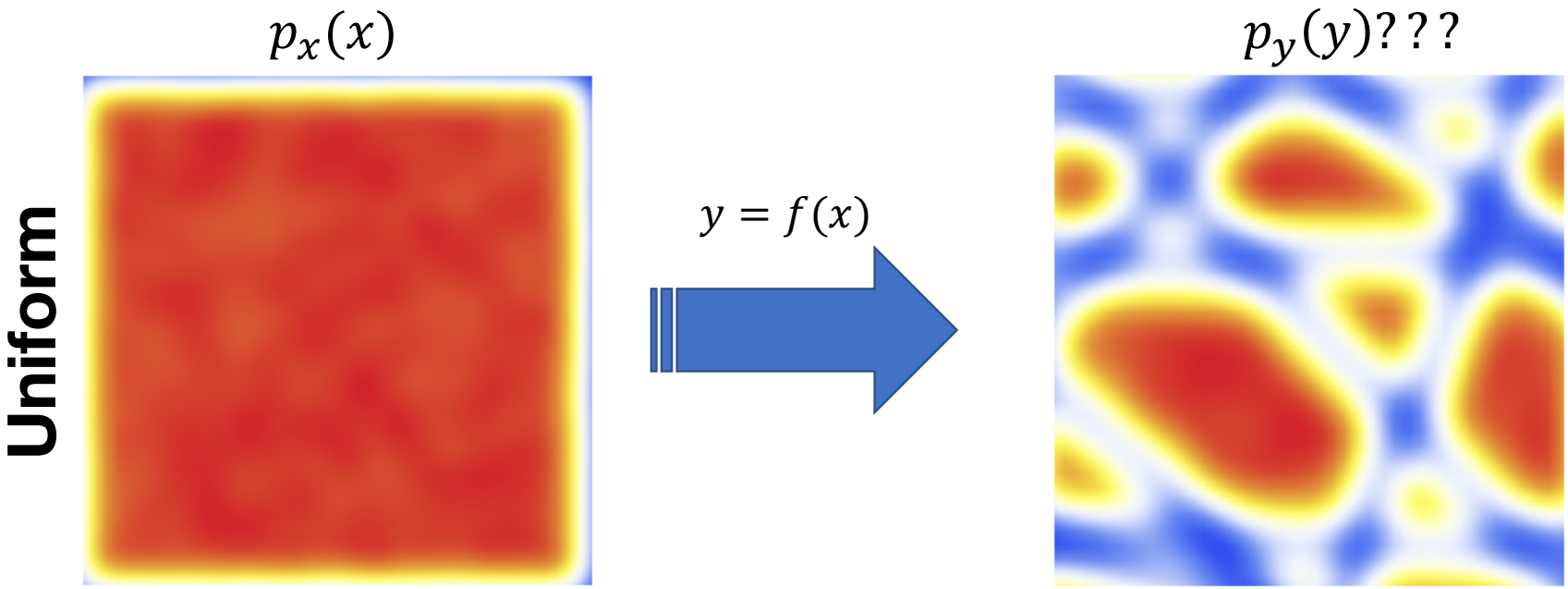
$$p_y(y) = p_x(f^{-1}(y)) \left| \frac{\mathrm{d}}{\mathrm{d}y} f^{-1}(y) \right| .$$



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

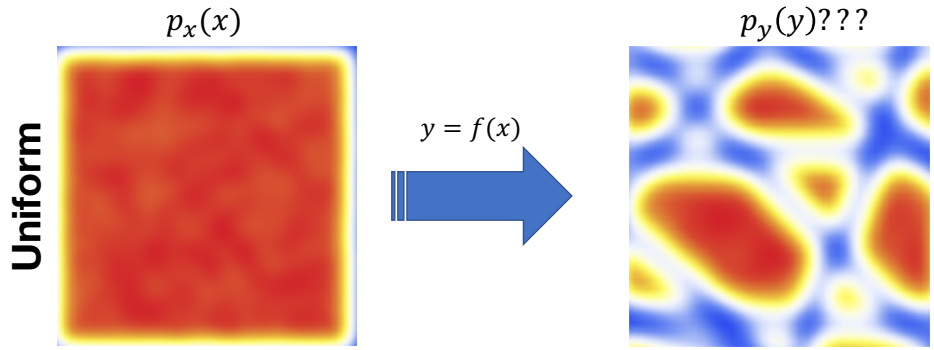


Generative Adversarial Networks

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!



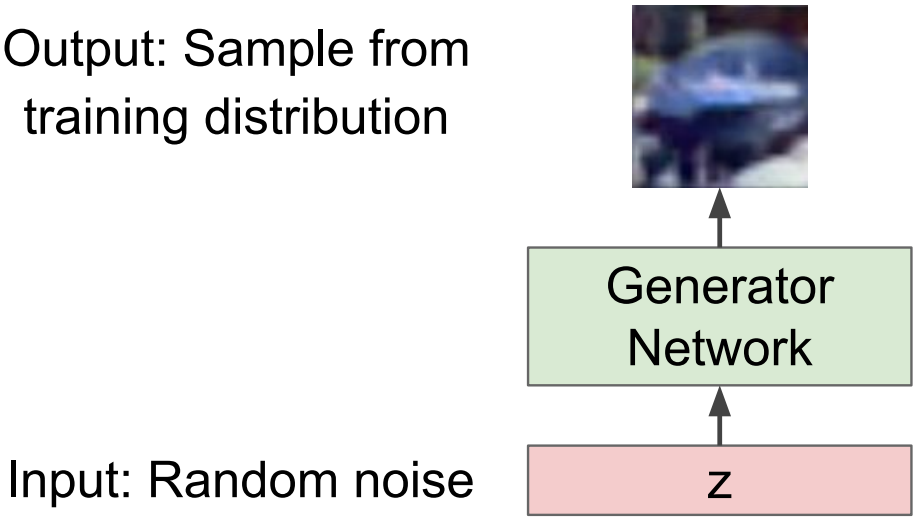
Goodfellow *et al.*, NIPS 2014

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!

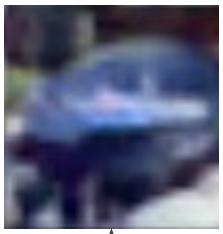


Goodfellow *et al.*, NIPS 2014

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?

Output: Sample from training distribution



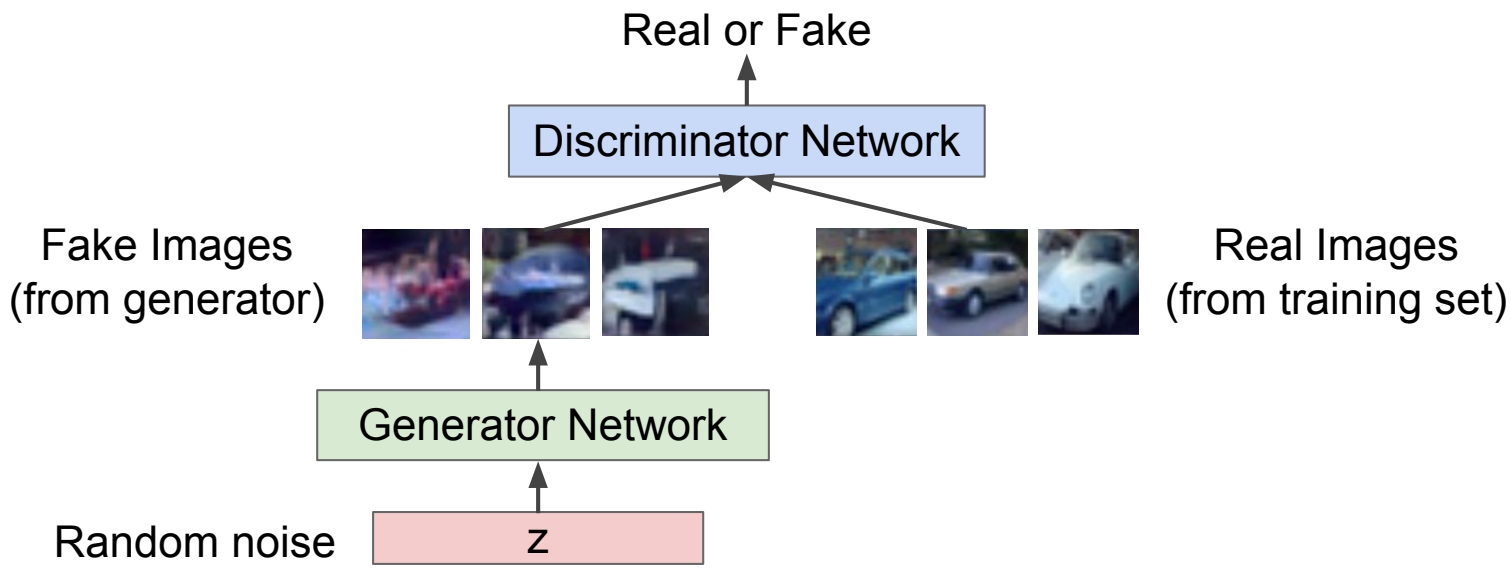
Generator Network

Input: Random noise

z

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.

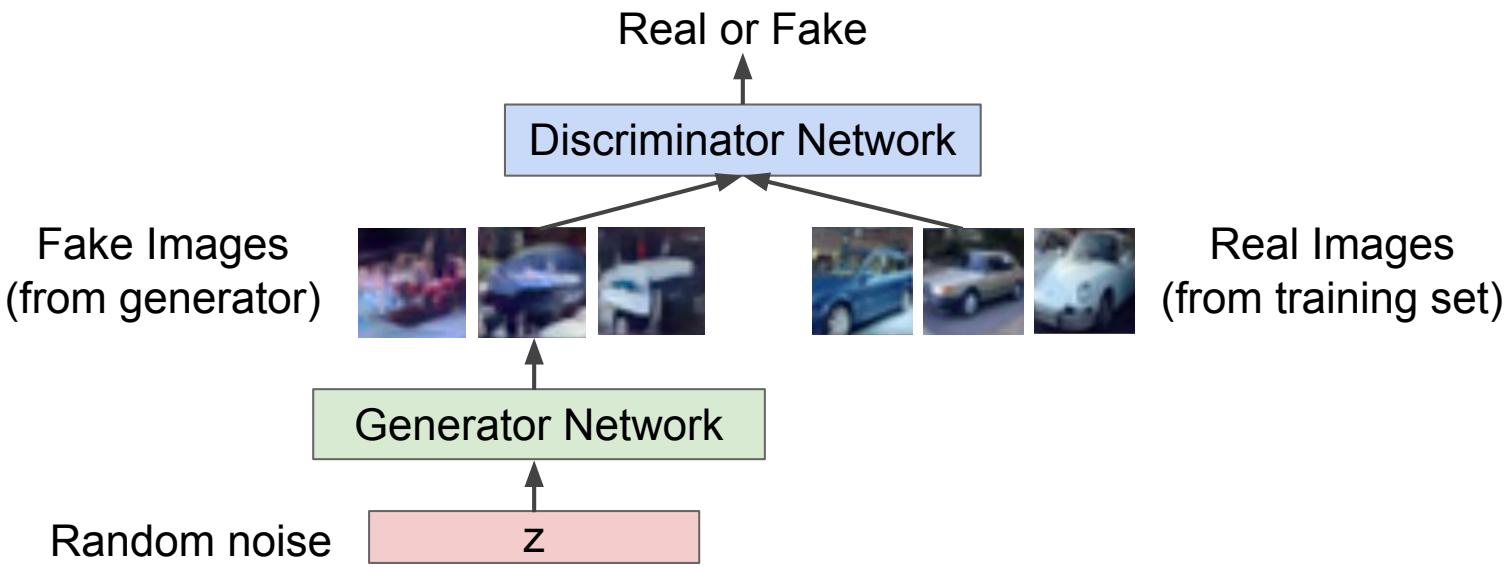
Why does this work?

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!

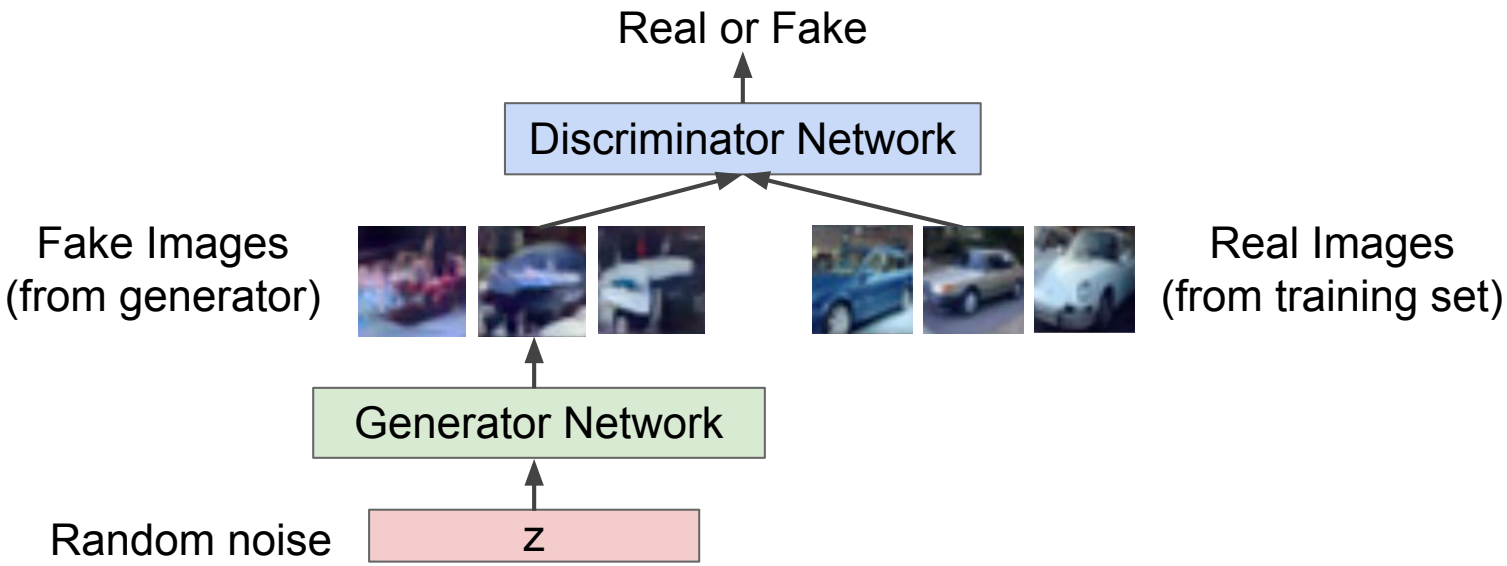
Training GANs: A Two-player Game

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.

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.

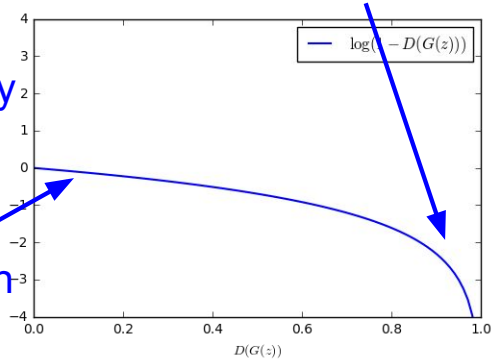
- How about changing it to:

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

  - 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:

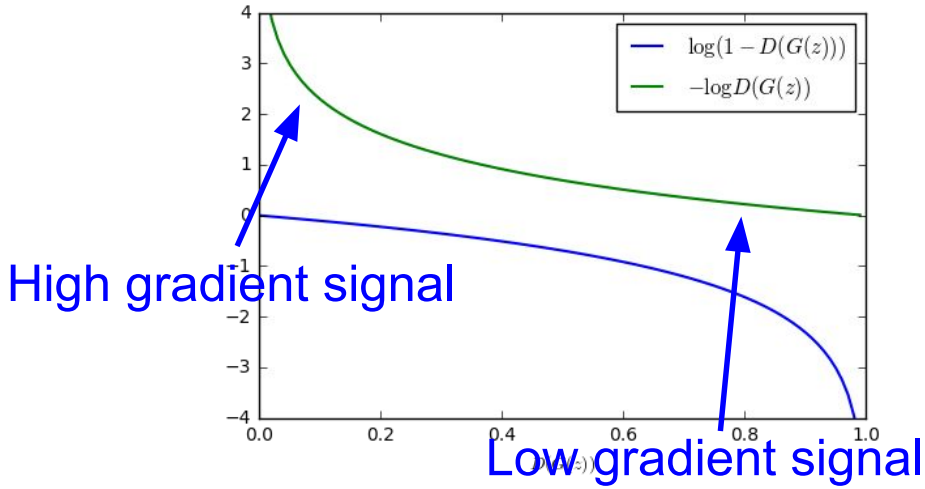
$$\min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - 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:

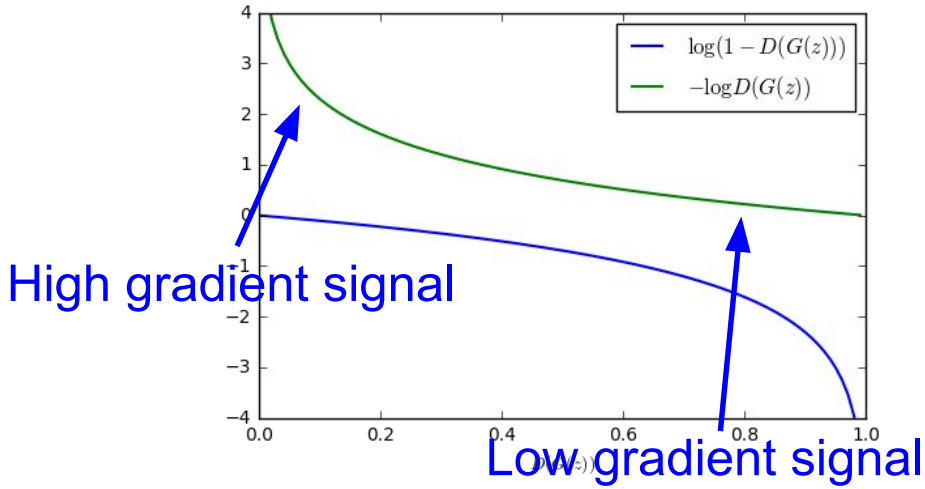
$$\min_{\theta_g} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}))\right)$$

  - 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:

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

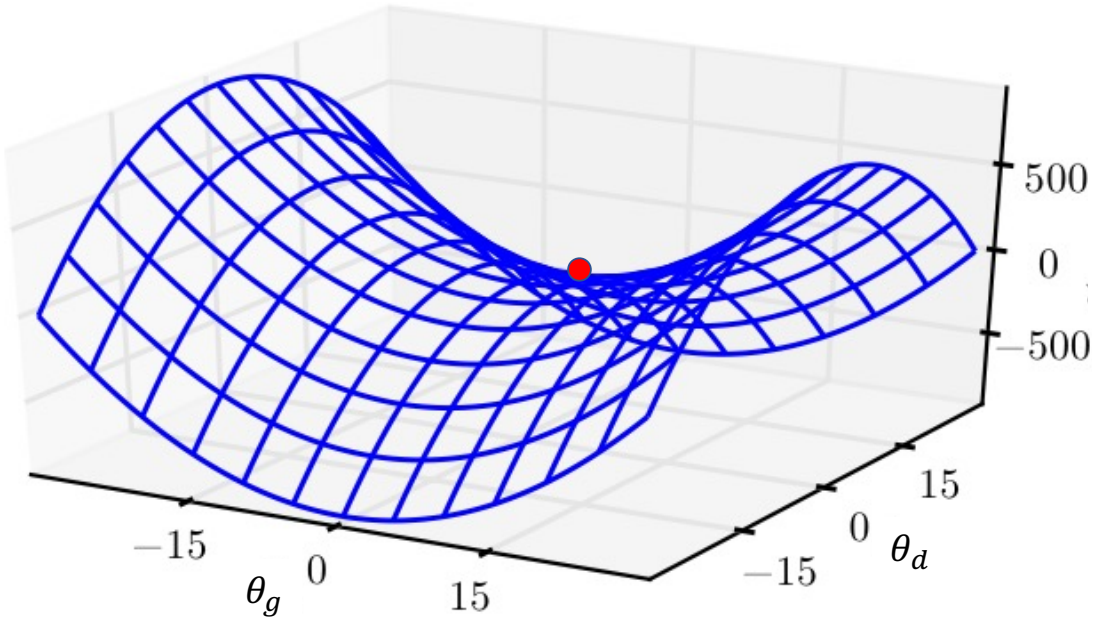
- What does an optimal solution of GAN look like?
  - A local minimum/maximum?
  - Or $\cdots$

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

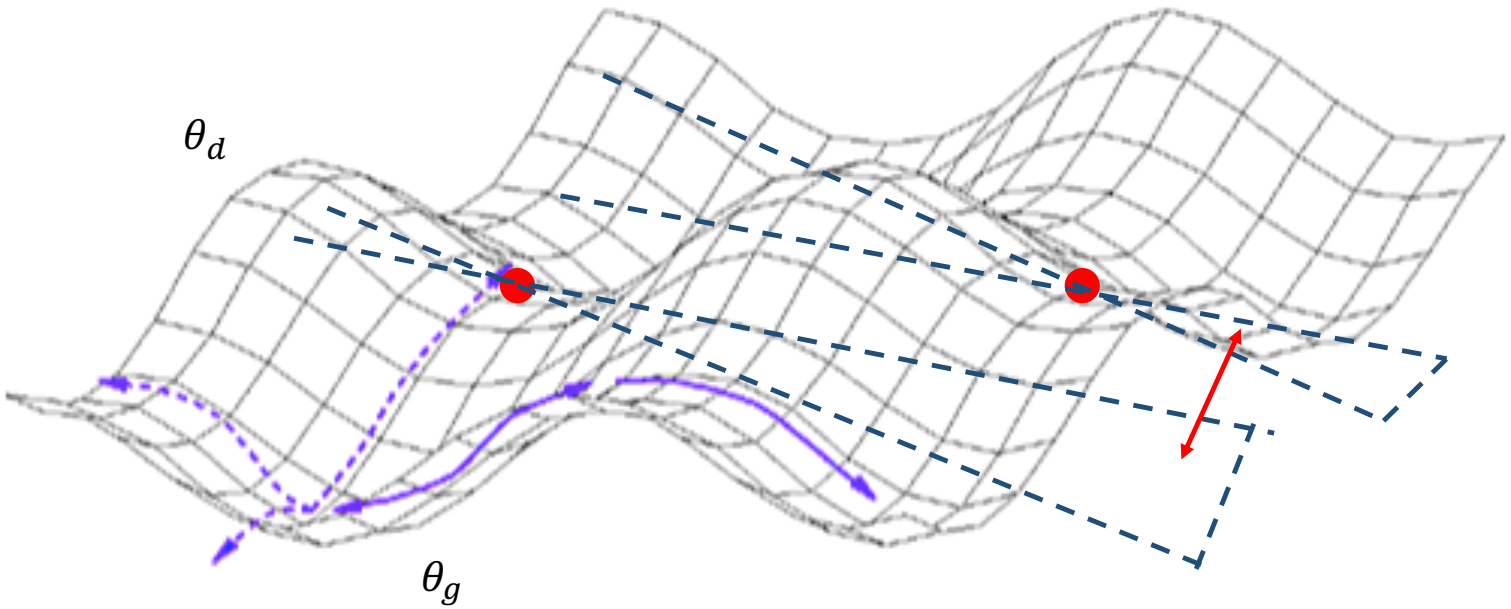
- No best rule for choosing k.

Training GANs: A Two-Player Game

A minimax objective function:

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

- Which one is the solution?

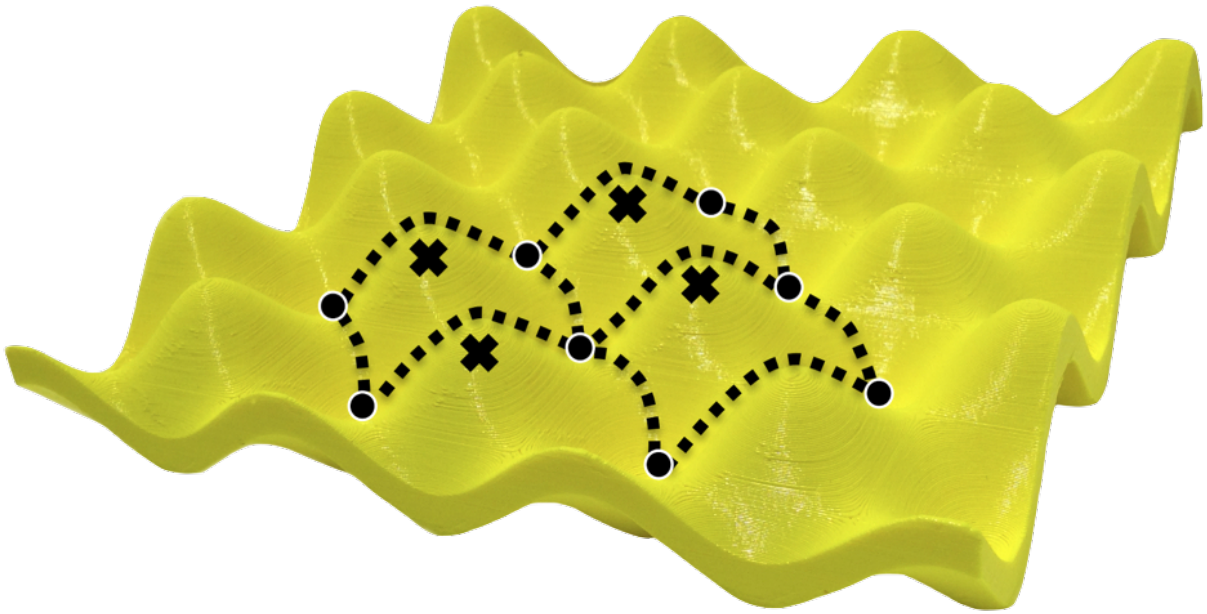


Training GANs: A Two-Player Game

A minimax objective function:

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

- Jointly training two networks is challenging, can be unstable:
  - Can have many saddle points $\Rightarrow$ many unstable sub-optima.

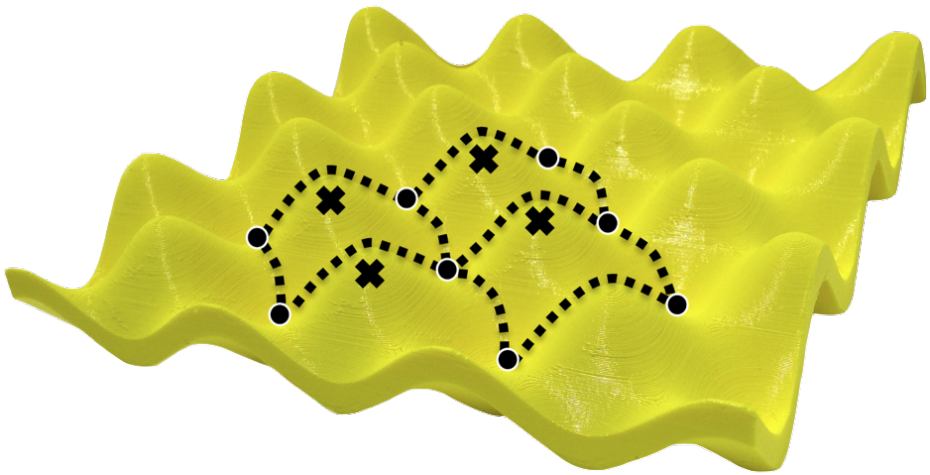


Training GANs: A Two-Player Game

A minimax objective function:

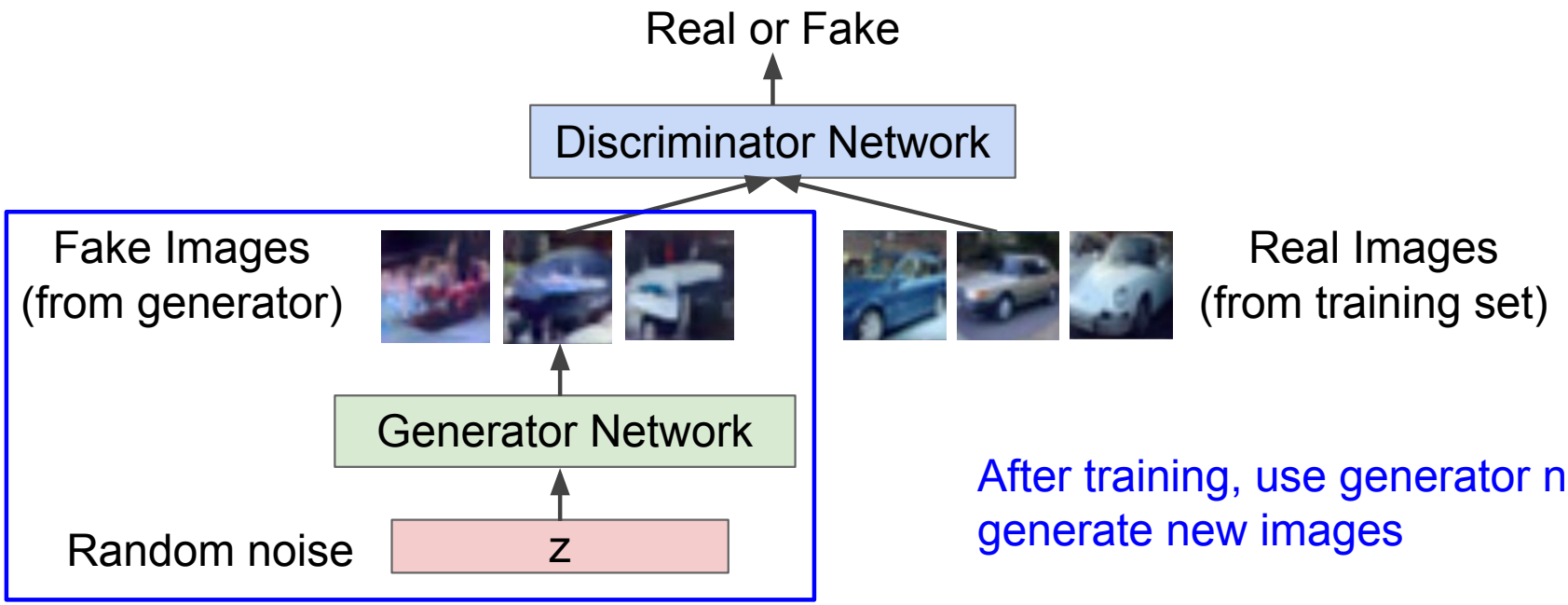
$$\min_{\boldsymbol{\theta}_g} \max_{\boldsymbol{\theta}_d} \left[\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D_{\boldsymbol{\theta}_d}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \log \left(1 - D_{\boldsymbol{\theta}_d}(G_{\boldsymbol{\theta}_g}(\mathbf{z}))\right) \right]$$

- Jointly training two networks is challenging, can be unstable:
  - Can have many saddle points $\Rightarrow$ many unstable sub-optima.
- Choosing objectives with better loss landscapes helps training, or designing better training algorithms, are active areas of research.



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.

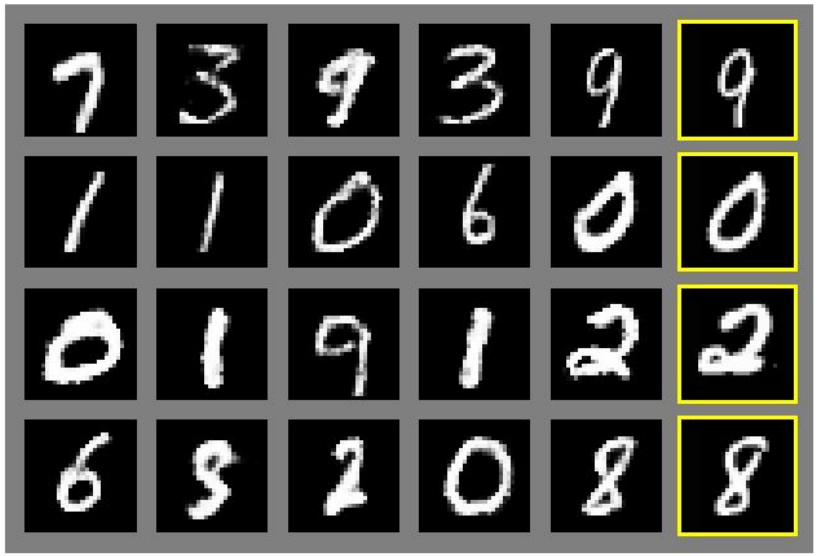


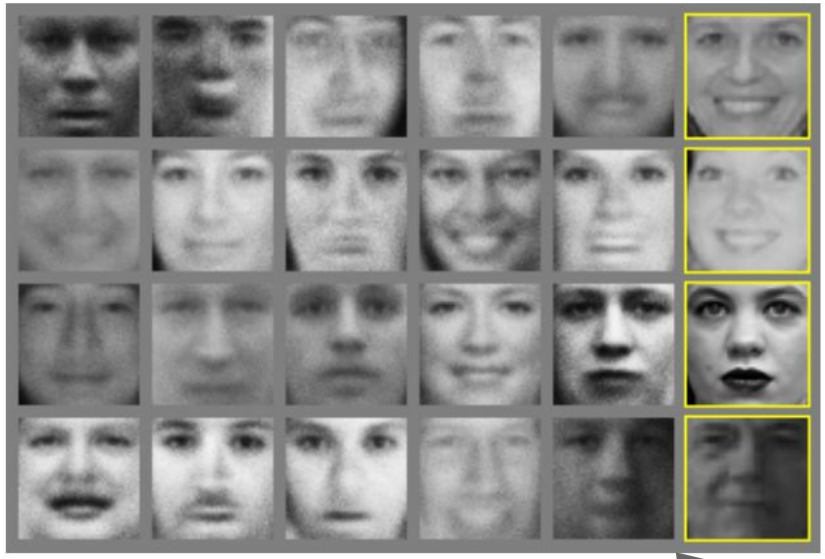
After training, use generator network to generate new images

Denton *et al*, 2015

GAN: Generated Samples

- Generator and discriminator as MLPs (left); Generator as deconvolutional NN, discriminator as a CNN (right).





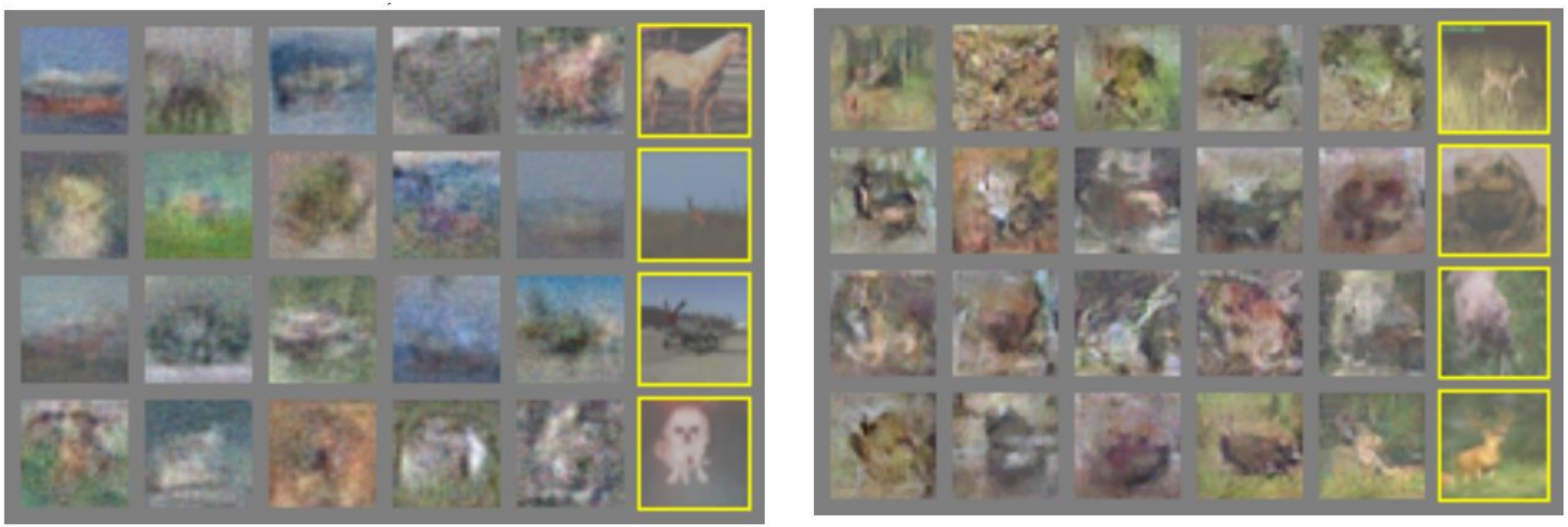
Nearest neighbor from training set

Goodfellow *et al*, NIPS 2014

GAN: Generated Samples

- Generator and discriminator as MLPs (left); Generator as deconvolutional NN, discriminator as a CNN (right).

Generated samples (CIFAR-10)



Nearest neighbor from training set

- Not great, as the generator and discriminator need to be carefully designed.

Goodfellow *et al*, NIPS 2014

Theoretical Properties of GANs

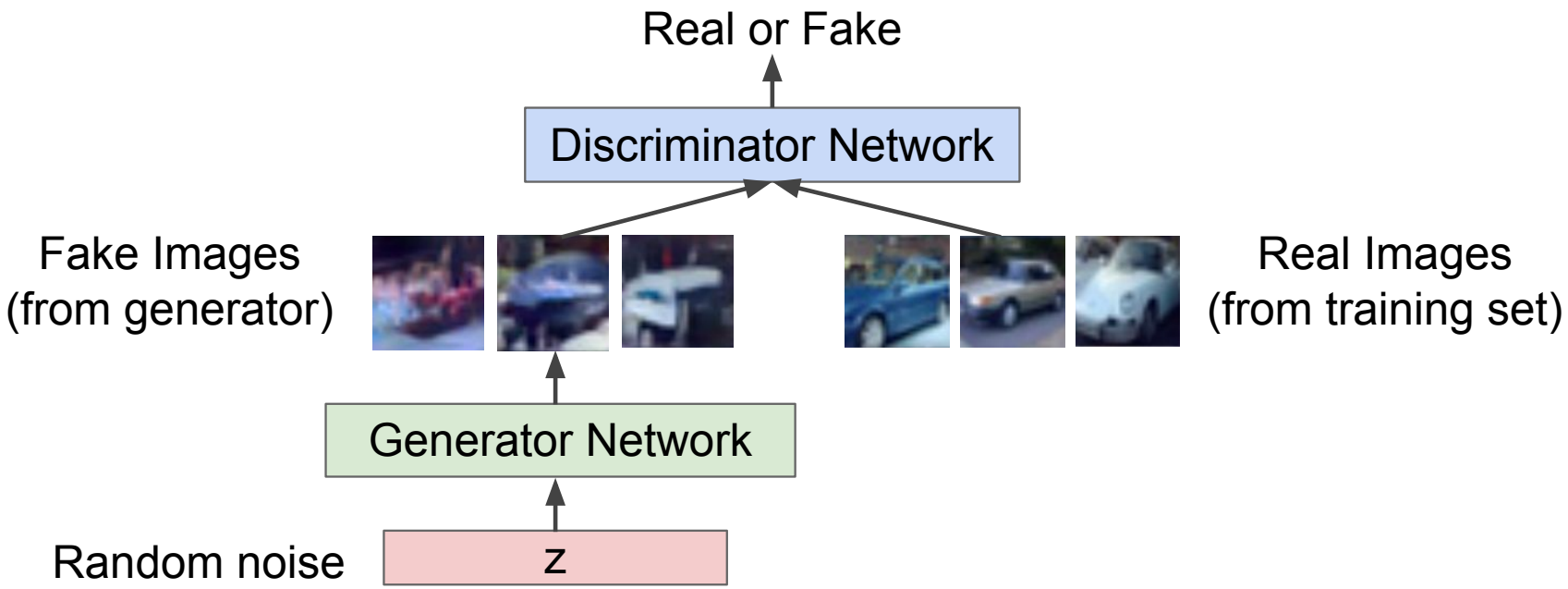
Recap: A Two-Player Game

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 is under going in GANs

Distribution matching between data distribution $p_{\text{data}}(\mathbf{x})$ and generator distribution $p_g(x)$.



GAN as Distribution Matching

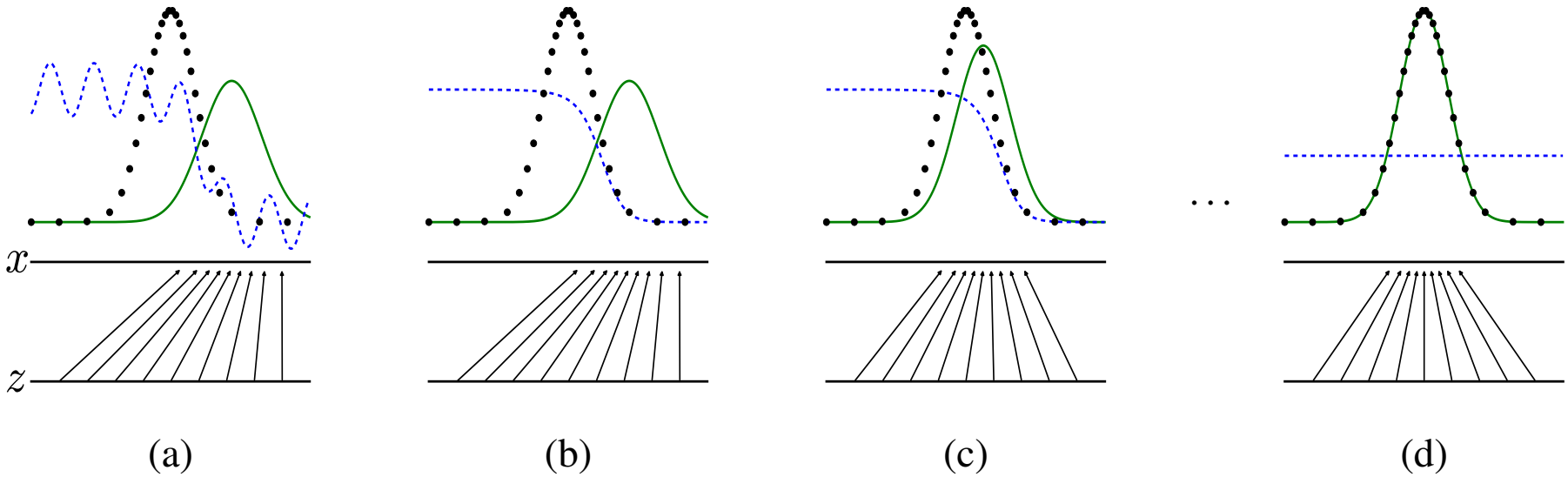


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Objective Reformulation

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

$$\Rightarrow V(G,D) = \int_x p_{\text{data}(\mathbf{x})} \log(D(\mathbf{x}))\mathrm{d}\,\mathbf{x} + \int_z p(\mathbf{z}) \log(1 - D(G(\mathbf{z})))\mathrm{d}\,\mathbf{z}$$

$$\overset{\text{change of r.v.}}{\Rightarrow} V(G,D) = \int_x \left(p_{\text{data}(\mathbf{x})} \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x}))\right) \mathrm{d}\,\mathbf{x}$$

Underlying assumption

The mapping G_{θ_g} is invertible:

- Not generally satisfied in DNN, but just use in practice.

Global Optimality of p_g

$$V(G, D) = \int_x \left(p_{\text{data}(\mathbf{x})} \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) \right) \mathrm{d}\,\mathbf{x}$$

Theorem

For G fixed, the optimal discriminator is

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

Objective Reformulation

$$V(G, D) = \int_x \left(p_{\text{data}(\mathbf{x})} \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) \right) \mathrm{d}\,\mathbf{x}$$

The GAN objective can be reformulated as

$$\min_G C(G) \triangleq \min_G \max_D V(G, D)$$

$$= \min_G \left\{ \underbrace{\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right] + \mathbb{E}_{\mathbf{x} \sim p_g} \left[\log \frac{p_g(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_g(\mathbf{x})} \right]}_{C(G)} \right\}$$

Global Optimality of GAN

Theorem

The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$, i.e.,

$$\min_G C(G) = -\log 4 \ .$$

Jensen-Shannon Divergence

Definition (Jensen-Shannon Divergence)

The Jensen-Shannon divergence between two distributions $p_1(x)$ and $p_2(x)$ is defined as:

$$JSD(p_1 \| p_2) \triangleq \frac{1}{2}\left(KL\left(p_1 \| \frac{p_1 + p_2}{2}\right) + KL\left(p_2 \| \frac{p_1 + p_2}{2}\right)\right)$$

Jensen-Shannon Divergence

Definition (Jensen-Shannon Divergence)

The Jensen-Shannon divergence between two distributions $p_1(x)$ and $p_2(x)$ is defined as:

$$JSD(p_1 \| p_2) \triangleq \frac{1}{2}\left(KL\left(p_1 \| \frac{p_1 + p_2}{2}\right) + KL\left(p_2 \| \frac{p_1 + p_2}{2}\right)\right)$$

The training criterion $C(G)$ of GAN can be written in terms of the Jensen-Shannon divergence:

$$C(G) = -\log(4) + 2JSD\left(p_{\text{data}} \| p_g\right)$$
$$\Rightarrow G^* = \arg\min_G C(G) = \arg\min_G JSD\left(p_{\text{data}} \| p_g\right)$$

Jensen-Shannon Divergence

Definition (Jensen-Shannon Divergence)

The Jensen-Shannon divergence between two distributions $p_1(x)$ and $p_2(x)$ is defined as:

$$JSD(p_1 \| p_2) \triangleq \frac{1}{2}\left(KL\left(p_1 \| \frac{p_1 + p_2}{2}\right) + KL\left(p_2 \| \frac{p_1 + p_2}{2}\right)\right)$$

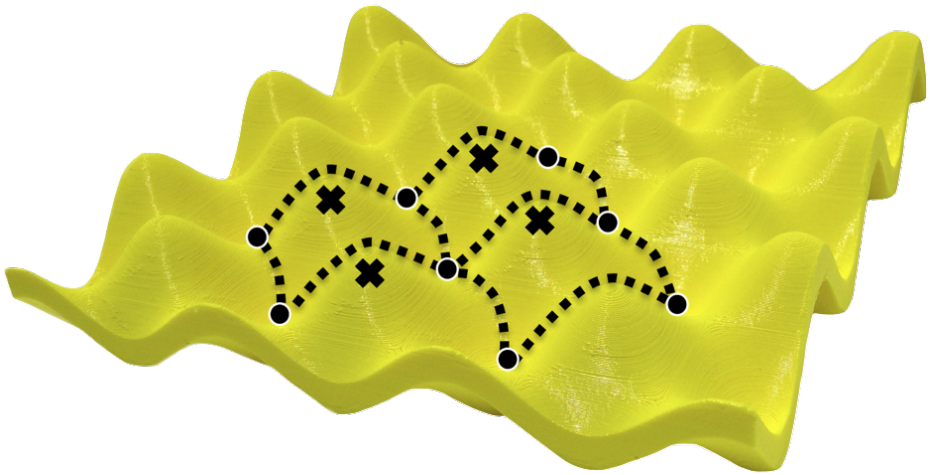
The training criterion $C(G)$ of GAN can be written in terms of the Jensen-Shannon divergence:

$$C(G) = -\log(4) + 2JSD\left(p_{\text{data}} \| p_g\right)$$
$$\Rightarrow G^* = \arg\min_G C(G) = \arg\min_G JSD\left(p_{\text{data}} \| p_g\right)$$

Distribution matching!

Convergence of the Algorithm

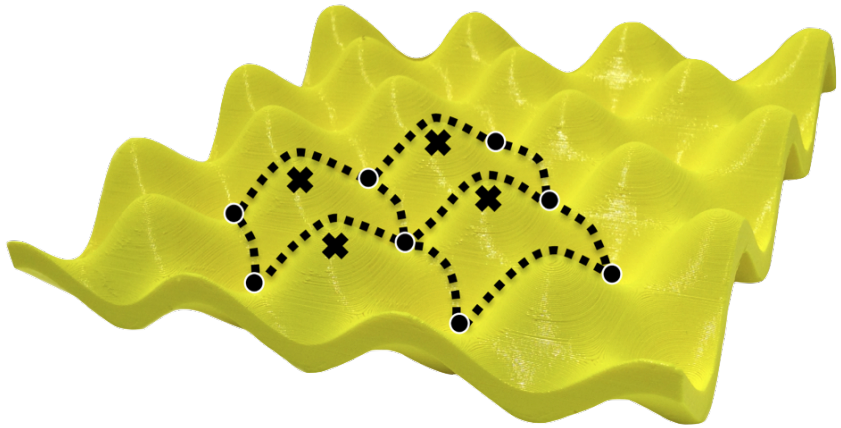
Intuitively not necessary converge



- Hard to say which one is the global optima.
- Not able to reach the globally optimal saddle point.

Convergence of the Algorithm

Intuitively not necessary converge



Theorem

If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{\mathbf{x}\sim p_{data}} \log D_G^*(\mathbf{x}) + \mathbb{E}_{\mathbf{x}\sim p_g} \log\left(1 - D_G^*(\mathbf{x})\right) \ ,$$

then p_g converges to p_{data}.

Proof Idea [Optional]

$$V(G, D) = \int_x \left(p_{\text{data}(\mathbf{x})} \log(D(\mathbf{x})) + p_g(\mathbf{x}) \log(1 - D(\mathbf{x})) \right) \mathrm{d}\,\mathbf{x}$$

GAN objective

$$\begin{aligned}
&\min_G C(G) \triangleq \min_G \max_D V(G, D) \\
&= -\log(4) + 2 \min_G JSD\left(p_{\text{data}} \| p_g\right) \\
&= -\log(4) + \min_G KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}} + p_g}{2}\right)
\end{aligned}$$

Proof Idea [Optional]

GAN objective

$$\begin{aligned}
&\min_G C(G) \triangleq \min_G \max_D V(G, D) \\
&= -\log(4) + 2 \min_G JSD\left(p_{\text{data}} \| p_g\right) \\
&= -\log(4) + \min_G KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}} + p_g}{2}\right)
\end{aligned}$$

Alternative GAN objective in the space of probability distributions

$$\min_{p_g} \mathcal{F}(p_g) \triangleq \min_{p_g} KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}} + p_g}{2}\right) .$$

Proof Idea [Optional]

$$\mathcal{F}(p_g) = KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}} + p_g}{2}\right) .$$

- To ensure global optima, we need to prove $\mathcal{F}(p_g)$ is convex w.r.t. p_g.
  - If convex, we can find the global optima (*i.e.*, $p_g = p_{\text{data}}$) by doing gradient descent on p_g.
  - Convexity in space of distributions!
- Specifically, we need to prove for two distributions p_1 and p_2 and $\lambda \in [0, 1]$:

$$\mathcal{F}(\lambda p_1 + (1 - \lambda) p_2) \leq \lambda \mathcal{F}(p_1) + (1 - \lambda) \mathcal{F}(p_2) .$$

Proof Idea [Optional]

$$\mathcal{F}(p_g) = KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}} + p_g}{2}\right) .$$

- To ensure global optima, we need to prove $\mathcal{F}(p_g)$ is convex w.r.t. p_g.
  - If convex, we can find the global optima (*i.e.*, $p_g = p_{\text{data}}$) by doing gradient descent on p_g.
  - Convexity in space of distributions!
- Specifically, we need to prove for two distributions p_1 and p_2 and $\lambda \in [0, 1]$:

$$\mathcal{F}(\lambda p_1 + (1 - \lambda)p_2) \leq \lambda \mathcal{F}(p_1) + (1 - \lambda)\mathcal{F}(p_2) .$$

Proof Idea [Optional]

$$\mathcal{F}(p_g) = KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}} + p_g}{2}\right) .$$

$$\Rightarrow \mathcal{F}(\underbrace{\lambda p_1 + (1-\lambda)p_2}_{:=p}) \leq \lambda \mathcal{F}(p_1) + (1-\lambda)\mathcal{F}(p_2)$$

- Let $p = \lambda p_1 + (1-\lambda)p_2$, we need to prove:

$$KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p}{2}\right) + KL\left(p \| \frac{p_{\text{data}} + p}{2}\right)$$

$$\leq \lambda KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_1}{2}\right) + (1-\lambda) KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_2}{2}\right)$$

$$+ \lambda KL\left(p_1 \| \frac{p_{\text{data}} + p_1}{2}\right) + (1-\lambda) KL\left(p_2 \| \frac{p_{\text{data}} + p_2}{2}\right)$$

Proof Idea [Optional]

$$KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p}{2}\right) + KL\left(p \| \frac{p_{\text{data}} + p}{2}\right)$$
$$\leq \lambda KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_1}{2}\right) + (1-\lambda) KL\left(p_{\text{data}} \| \frac{p_{\text{data}} + p_2}{2}\right)$$
$$+ \lambda KL\left(p_1 \| \frac{p_{\text{data}} + p_1}{2}\right) + (1-\lambda) KL\left(p_2 \| \frac{p_{\text{data}} + p_2}{2}\right)$$

Lemma (Convexity of KL divergence)

Let a_1, b_1 and a_2, b_2 be probability distributions over x, and $\lambda \in (0, 1)$. Define $a = \lambda a_1 + (1-\lambda) a_2$, $b = \lambda b_1 + (1-\lambda) b_2$. Then

$$KL(a \| b) \leq \lambda KL(a_1 \| b_1) + (1-\lambda) KL(a_2 \| b_2)$$

Proof Idea [Optional]

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1-\lambda)KL(a_2\|b_2)$$

$$p = \lambda p_1 + (1-\lambda)p_2$$

Proof of

$$KL\left(p_{\textbf{data}}\|\frac{p_{\textbf{data}}+p}{2}\right) \leq \lambda KL\left(p_{\textbf{data}}\|\frac{p_{\textbf{data}}+p_1}{2}\right) + (1-\lambda)KL\left(p_{\textbf{data}}\|\frac{p_{\textbf{data}}+p_2}{2}\right)$$

Proof Idea [Optional]

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1-\lambda)KL(a_2\|b_2)$$

$$p = \lambda p_1 + (1-\lambda)p_2$$

Proof of

$KL\left(p_{\text{data}}\|\frac{p_{\text{data}}+p}{2}\right) \leq \lambda KL\left(p_{\text{data}}\|\frac{p_{\text{data}}+p_1}{2}\right) + (1-\lambda)KL\left(p_{\text{data}}\|\frac{p_{\text{data}}+p_2}{2}\right)$

Proof.

Let $a_1 = a_2 = p_{\text{data}}$, $b_1 = \frac{p_{\text{data}}+p_1}{2}$, $b_2 = \frac{p_{\text{data}}+p_2}{2}$. Substituting these into the KL inequality, we get the conclusion. □

Proof Idea [Optional]

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1-\lambda)KL(a_2\|b_2)$$

$$p = \lambda p_1 + (1-\lambda)p_2$$

Proof of $KL\left(p\|\frac{p_{\text{data}}+p}{2}\right) \leq \lambda KL\left(p_1\|\frac{p_{\text{data}}+p_1}{2}\right) + (1-\lambda)KL\left(p_2\|\frac{p_{\text{data}}+p_2}{2}\right)$

Proof Idea [Optional]

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1-\lambda)KL(a_2\|b_2)$$

$$p = \lambda p_1 + (1-\lambda)p_2$$

Proof of $KL\left(p\|\frac{p_{\text{data}}+p}{2}\right) \leq \lambda KL\left(p_1\|\frac{p_{\text{data}}+p_1}{2}\right) + (1-\lambda)KL\left(p_2\|\frac{p_{\text{data}}+p_2}{2}\right)$

Proof.

Let $a_1 = p_1$, $a_2 = p_2$, $b_1 = \frac{p_{\text{data}}+p_1}{2}$, $b_2 = \frac{p_{\text{data}}+p_2}{2}$. Substituting these into the KL inequality, we get the conclusion. □

Proof Idea [Optional]

Conclusion

$\mathcal{F}(p_g) = KL\left(p_{\text{data}} \| \frac{p_{\text{data}}+p_g}{2}\right) + KL\left(p_g \| \frac{p_{\text{data}}+p_g}{2}\right)$ is convex w.r.t. p_g:

- Global optima of $p_g = p_{\text{data}}$ can be obtained by optimizing $\mathcal{F}(p_g)$ with sub-gradient descent on the space of probability distributions:
  - Sub-gradient descent on the space of probability distributions corresponds to gradient descent on the parameter space of G.

Important

- Before optimizing p_g from $\mathcal{F}(p_g)$, we have assume the discriminator D is optimal given G, *i.e.*, $D^* = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x})+p_g(\mathbf{x})}$.
- This is why we need to do SGD for several steps for discriminator in the algorithm in each round.

Proof Idea [Optional]

Convexity of KL divergence

Let a_1, b_1 and a_2, b_2 be probability distributions over x, and $\lambda \in (0, 1)$. Define $a = \lambda a_1 + (1 - \lambda) a_2$, $b = \lambda b_1 + (1 - \lambda) b_2$. Then

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1 - \lambda) KL(a_2\|b_2)$$

Proof Idea [Optional]

Convexity of KL divergence

Let a_1, b_1 and a_2, b_2 be probability distributions over x, and $\lambda \in (0, 1)$. Define $a = \lambda a_1 + (1 - \lambda) a_2$, $b = \lambda b_1 + (1 - \lambda) b_2$. Then

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1 - \lambda) KL(a_2\|b_2)$$

Lemma (Log sum inequality)

Let $a_1, \cdots, a_n$ and $b_1, \cdots, b_n$ be nonnegative numbers. The log sum inequality states that

$$\left(\sum_i a_i\right) \log \frac{\sum_i a_i}{\sum_i b_i} \leq \sum_i a_i \log \frac{a_i}{b_i}$$

Proof Idea [Optional]

$$KL(a\|b) \leq \lambda KL(a_1\|b_1) + (1-\lambda)KL(a_2\|b_2)$$

$$\left(\sum_i a_i\right) \log \frac{\sum_i a_i}{\sum_i b_i} \leq \sum_i a_i \log \frac{a_i}{b_i}$$

Proof.

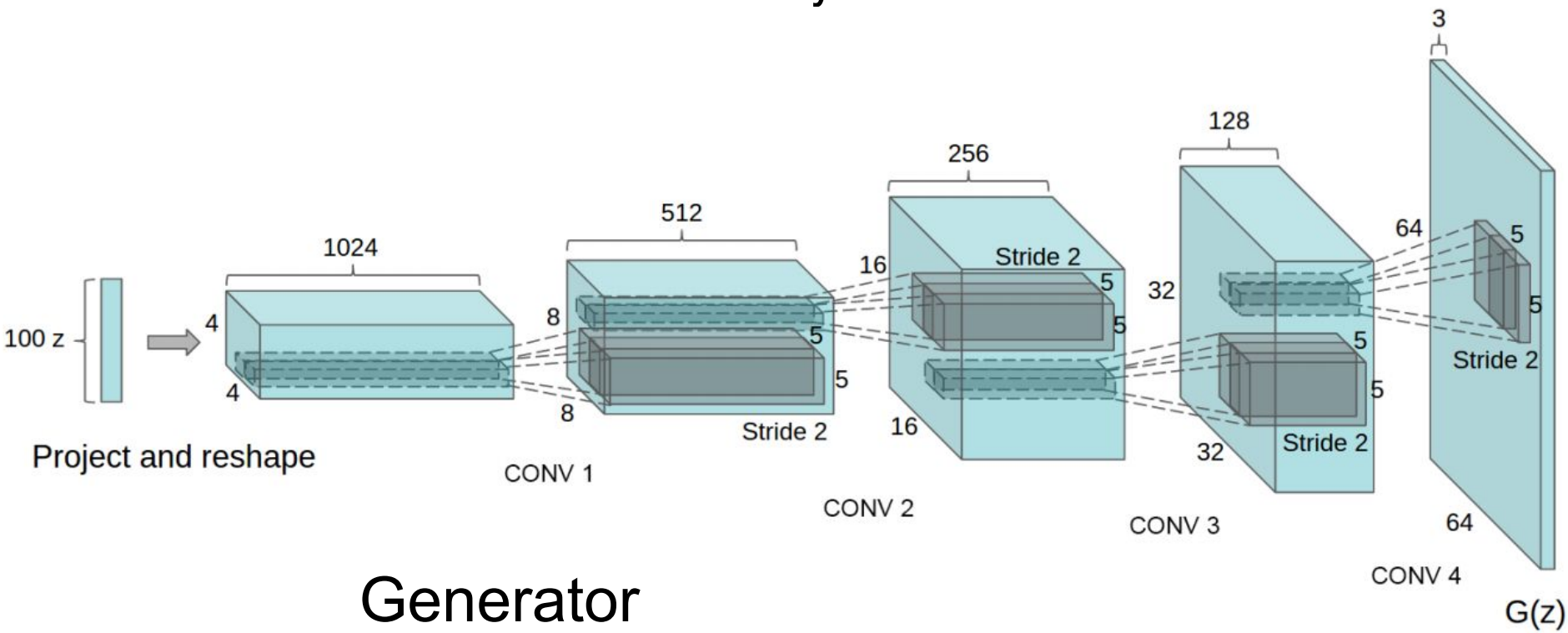
Let $p_1 = \lambda a_1$, $p_2 = (1-\lambda)a_2$, $q_1 = \lambda b_1$, $q_2 = (1-\lambda)b_2$.

$$\begin{aligned}
KL(a\|b) &= \int (\lambda a_1(x) + (1-\lambda)a_2(x)) \log \frac{\lambda a_1(x) + (1-\lambda)a_2(x)}{\lambda b_1(x) + (1-\lambda)b_2(x)} \mathrm{d}x \\
&= \int (p_1(x) + p_2(x)) \log \frac{p_1(x) + p_2(x)}{q_1(x) + q_2(x)} \mathrm{d}x \\
&\leq \int \left(p_1(x) \log \frac{p_1(x)}{q_1(x)} + p_2(x) \log \frac{p_2(x)}{q_2(x)} \right) \mathrm{d}x \\
&= \int \left(\lambda a_1(x) \log \frac{\lambda a_1(x)}{\lambda b_1(x)} + (1-\lambda)a_2(x) \log \frac{(1-\lambda)a_2(x)}{(1-\lambda)b_2(x)} \right) \mathrm{d}x \\
&= \lambda KL(a_1\|b_1) + (1-\lambda)KL(a_2\|b_2)
\end{aligned}$$

Deep Convolutional Generative Adversarial Networks (DCGAN)

Generative Adversarial Nets: Convolutional Architectur

- Generator is a deconvolutional neural network (upsampling network with deconvolutions).
- Discriminator is a convolutional network.
- Called deconvolutional generative adversarial nets (DCGAN).
- Discriminator: 4 convolutional layers.



Generator

Radford *et al*, ICLR 2016

Generative Adversarial Nets: Convolutional Architectur

- Generator is a deconvolutional neural network (upsampling network with deconvolutions).
- Discriminator is a convolutional network.
- Called deconvolutional generative adversarial nets (DCGAN).
- Discriminator: 4 convolutional layers.

Architecture guidelines for stable Deep Convolutional GANs

- No pooling layers.
- Use batchnorm in both the generator and discriminator.
- Remove fully connected layers.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in discriminator for all layers.

Radford *et al*, ICLR 2016

Generating Samples

Groundtruth MNIST | GAN | DCGAN (ours)

Generating Samples

Generating Samples

Generating Samples

Walking in the Latent Space

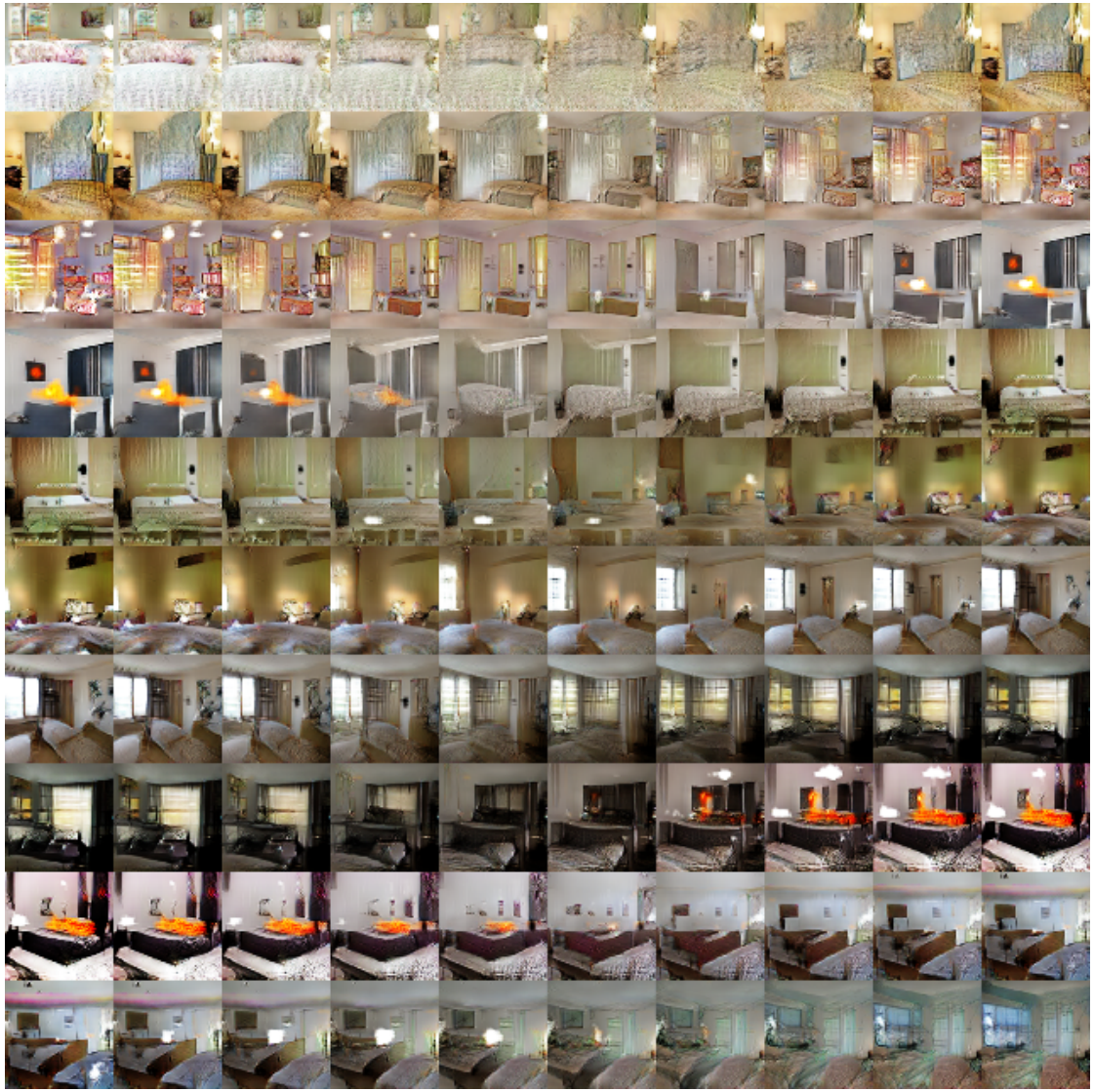
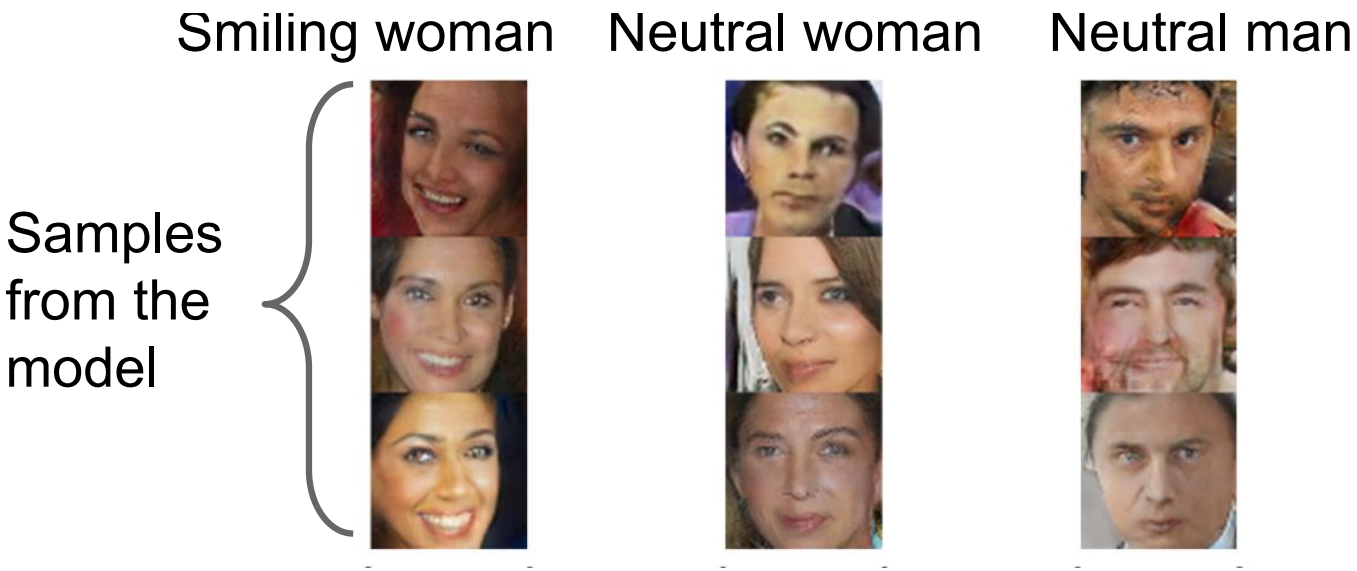


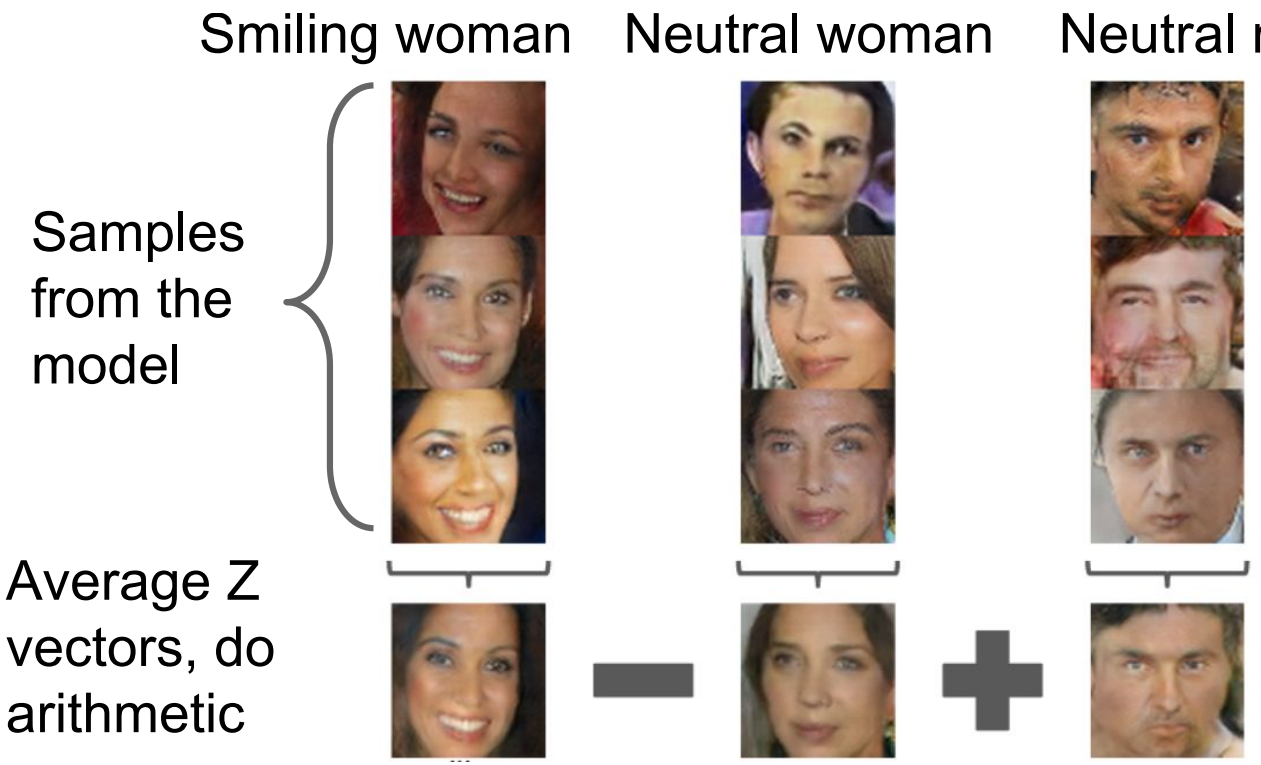
Figure 4: Top rows: Interpolation between a series of 9 random points in Z show that the space learned has smooth transitions, with every image in the space plausibly looking like a bedroom. In the 6th row, you see a room without a window slowly transforming into a room with a giant window. In the 10th row, you see what appears to be a TV slowly being transformed into a window.

Generative Adversarial Nets: Interpre



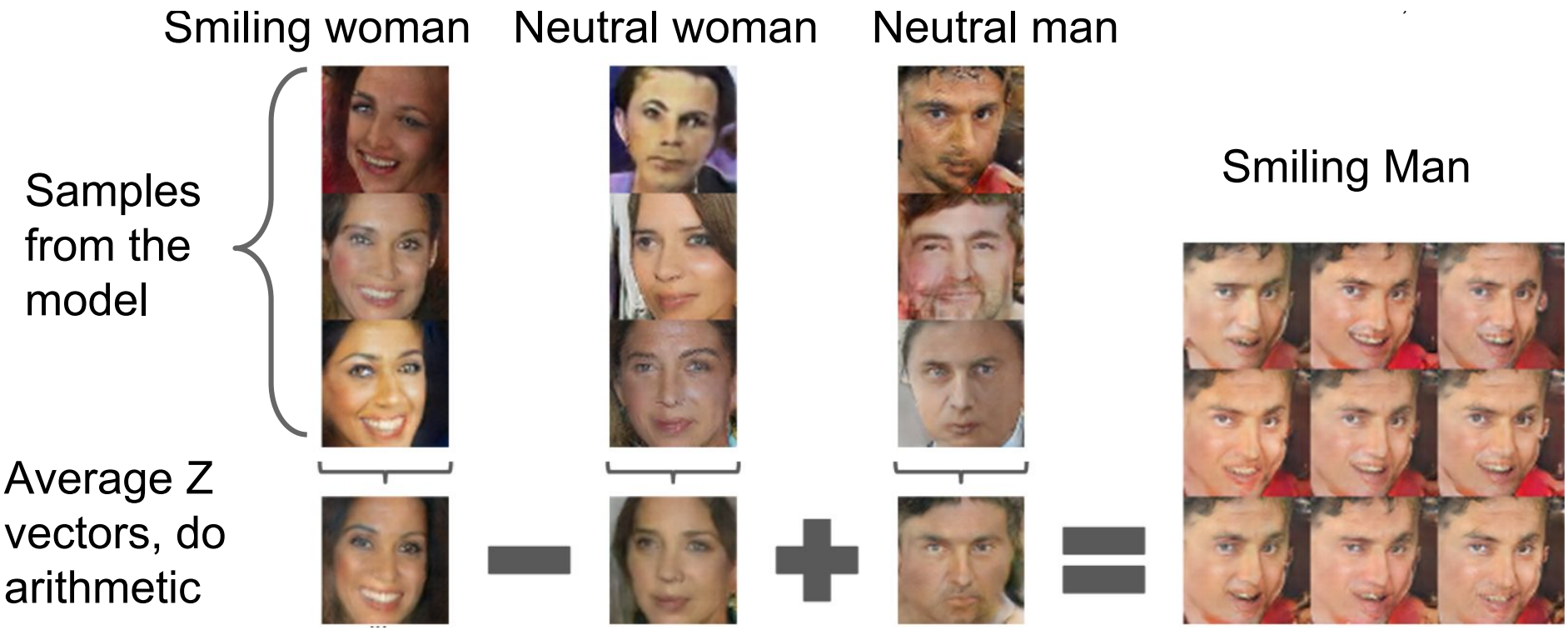
Radford *et al*, ICLR 2016

Generative Adversarial Nets: Interpre



Radford *et al*, ICLR 2016

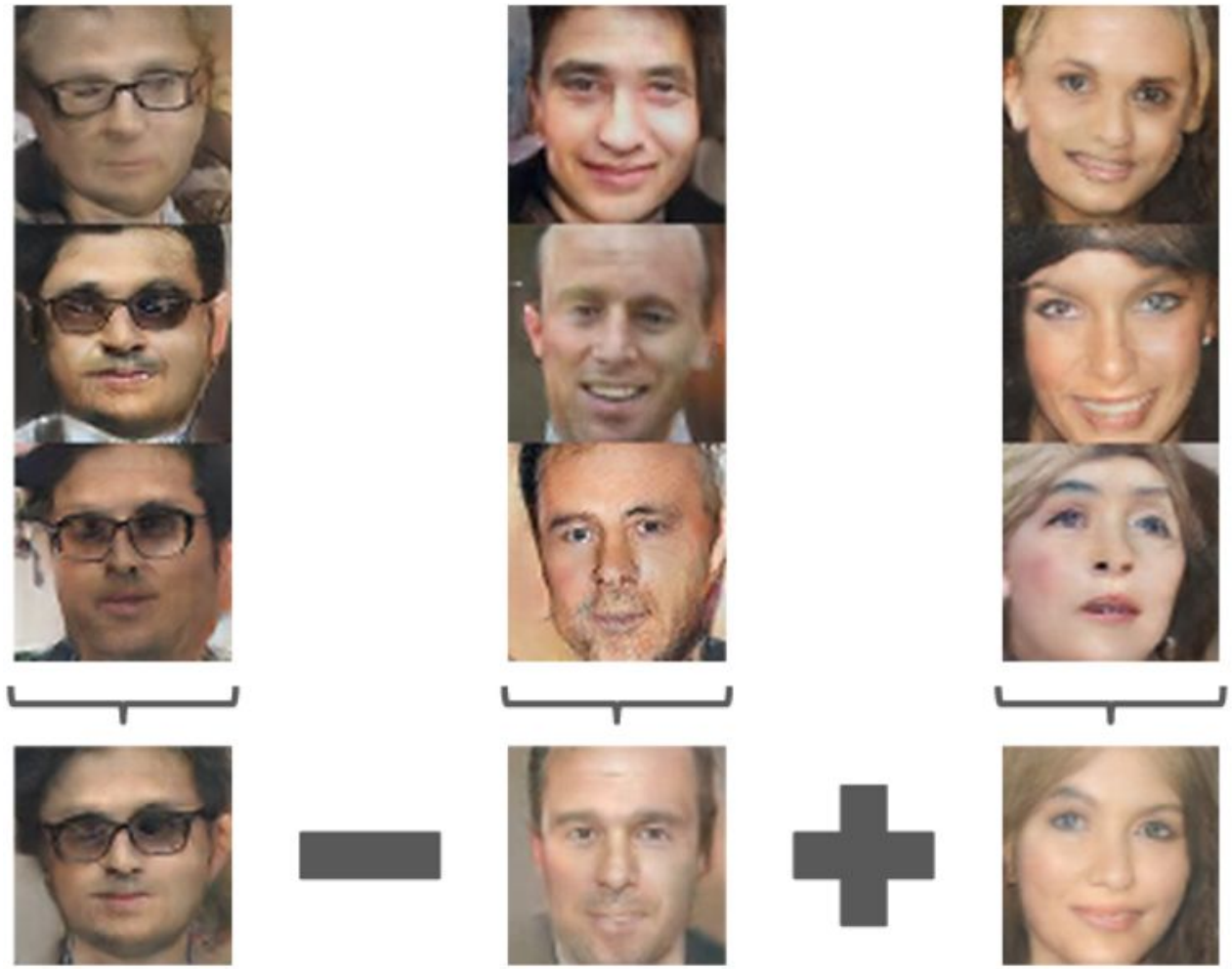
Generative Adversarial Nets: Interpre



Radford *et al*, ICLR 2016

Generative Adversarial Nets: Interpre

Glasses man — No glasses man — No glasses woman



Radford *et al*, ICLR 2016

Generative Adversarial Nets: Interpre

Glasses man

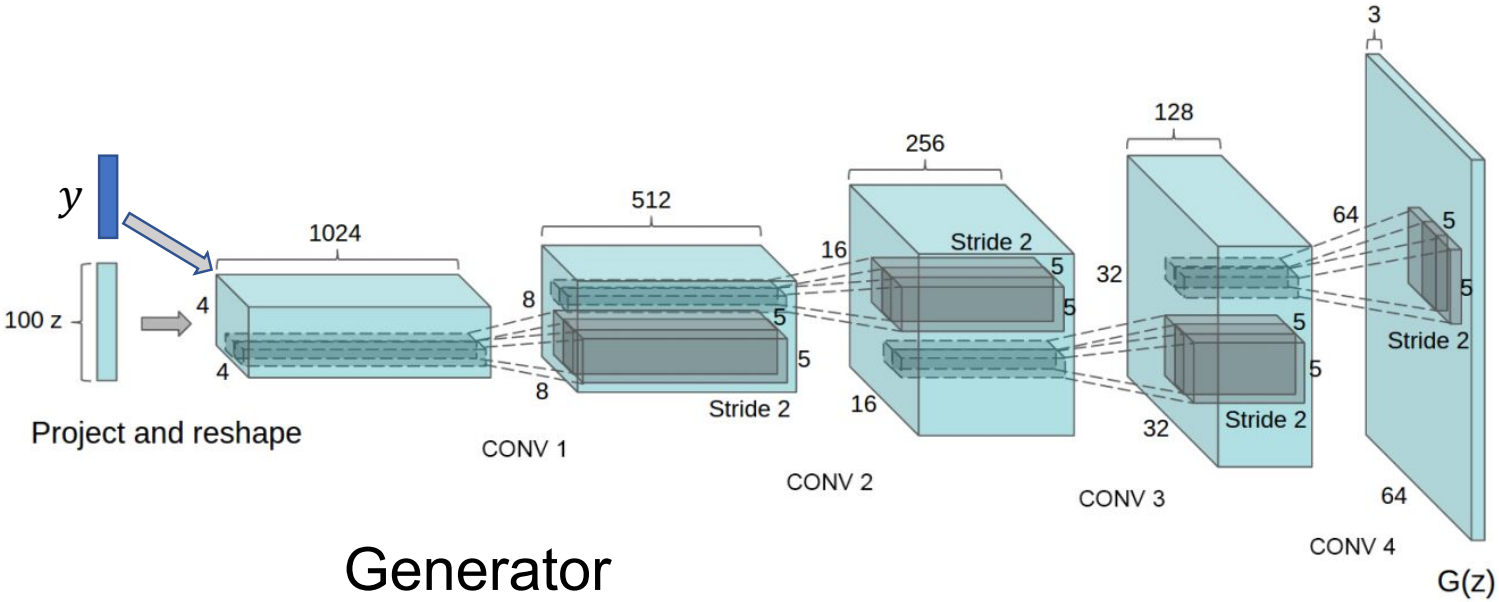
No glasses man

No glasses woman

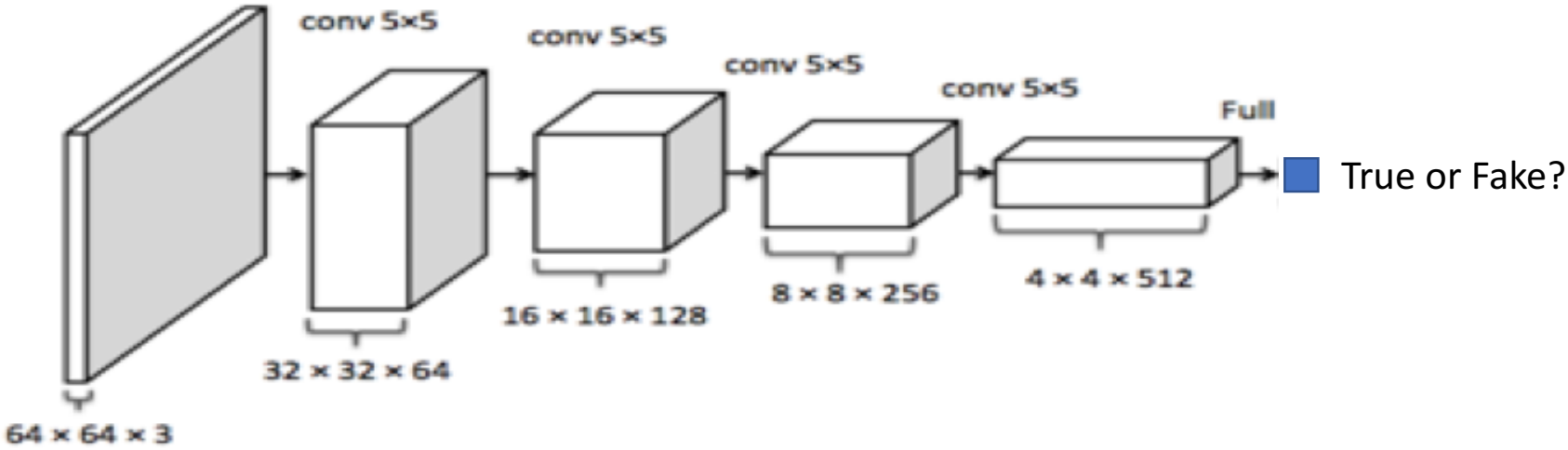
Woman with glasses

Radford *et al*, ICLR 2016

Extension: Adding Class Information

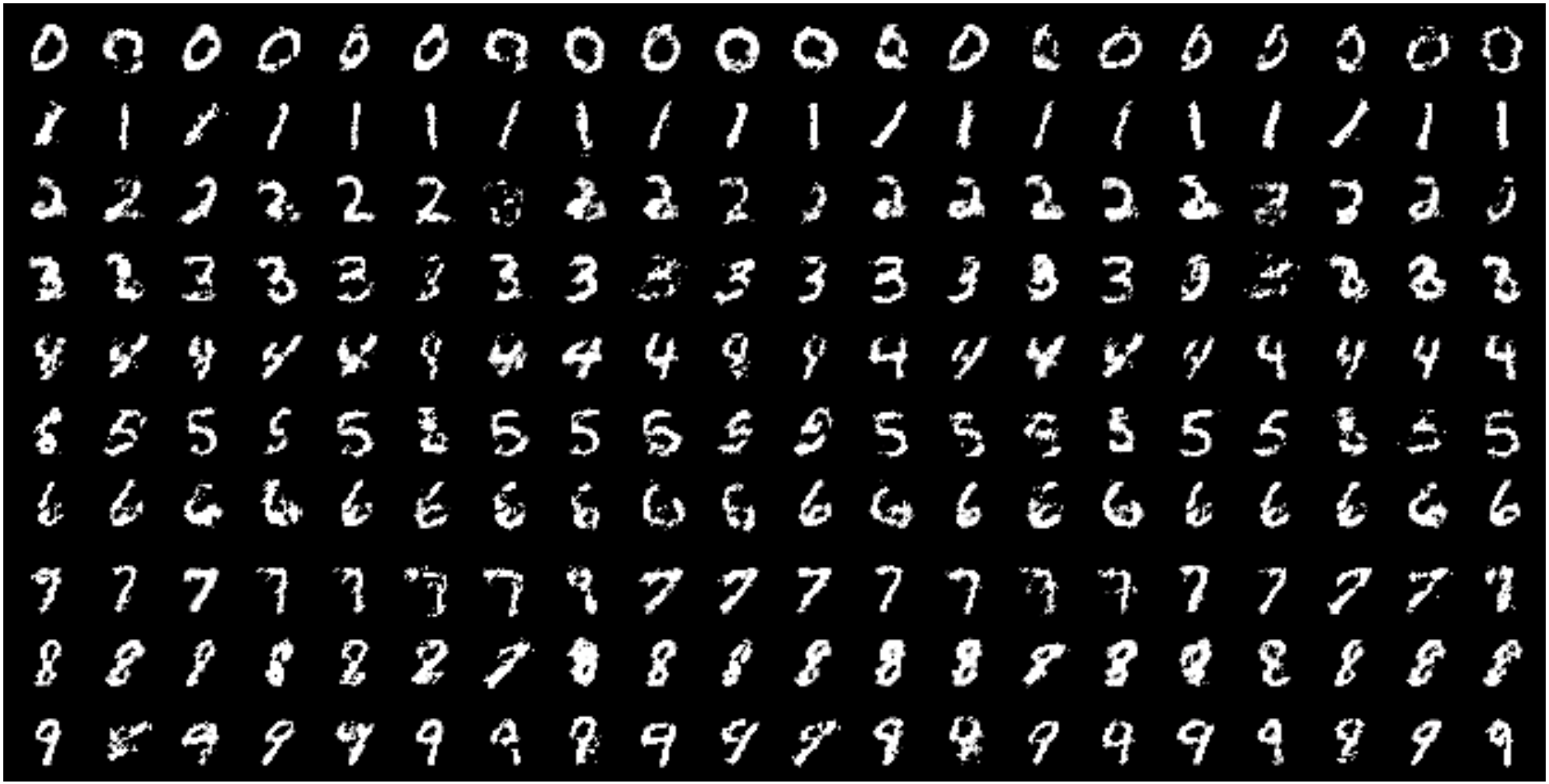


Generator

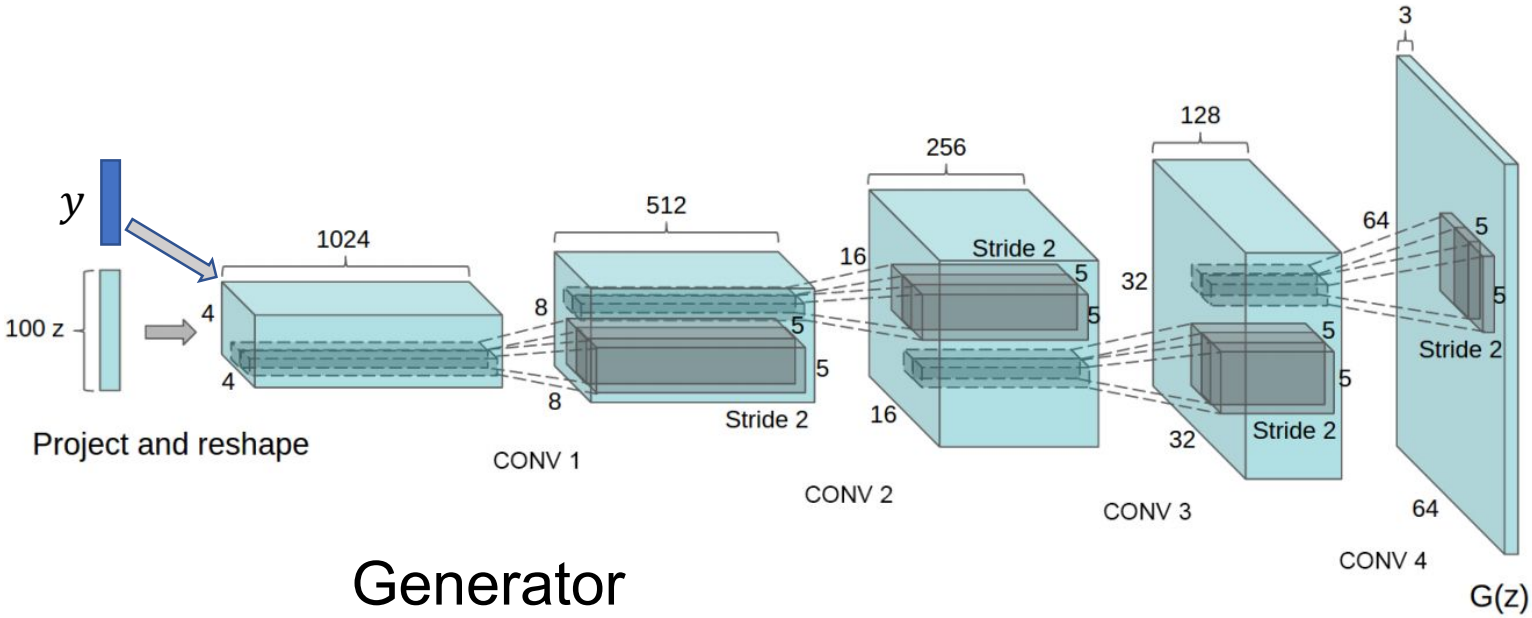


Discriminator

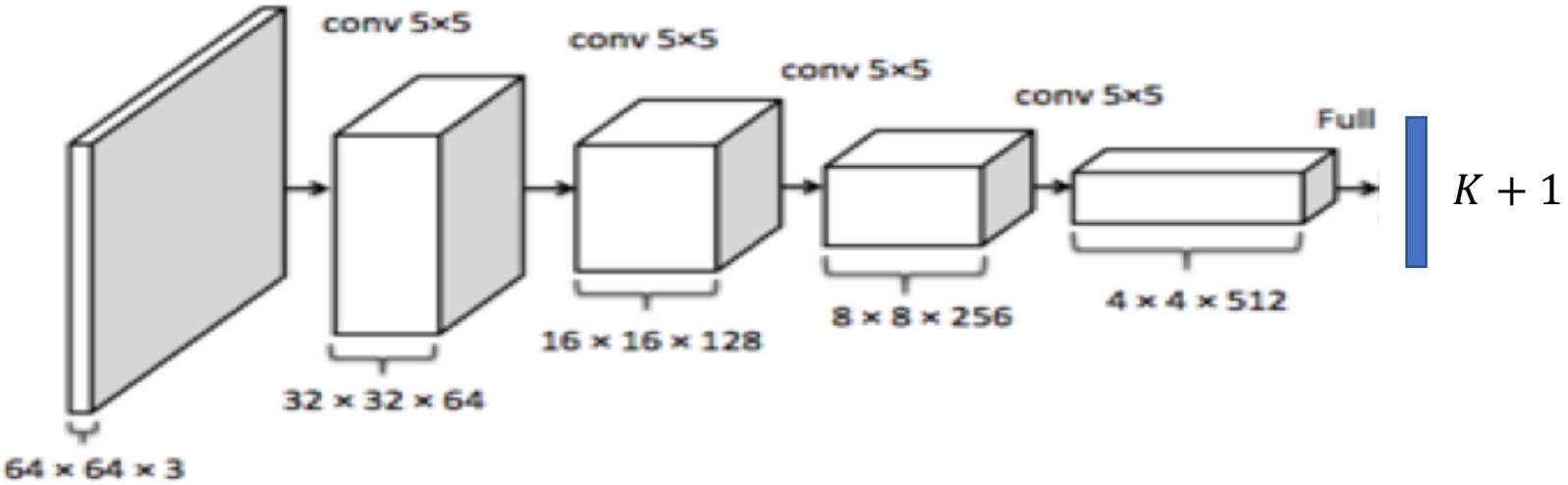
Extension: Adding Class Information



Extension: Multi-Class GAN



Generator



Discriminator

Extension: Multi-Class GAN

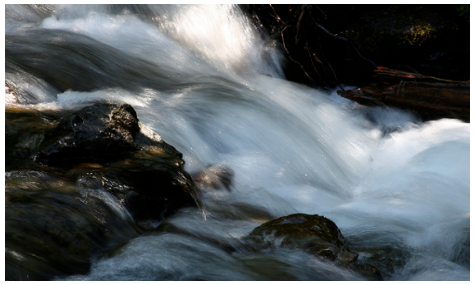
	User tags + annotations	Generated tags
	montanha, trem, inverno, frio, people, male, plant life, tree, structures, transport, car	taxi, passenger, line, transportation, railway station, passengers, railways, signals, rail, rails
	food, raspberry, delicious, homemade	chicken, fattening, cooked, peanut, cream, cookie, house made, bread, biscuit, bakes
	water, river	creek, lake, along, near, river, rocky, treeline, valley, woods, waters
	people, portrait, female, baby, indoor	love, people, posing, girl, young, strangers, pretty, women, happy, life

Image Credit: M. Mirza & S. Osindero

Wasserstein GANs

Recap: GANs

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

Theorem

If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{\mathbf{x} \sim p_{data}} \log D_G^*(\mathbf{x}) + \mathbb{E}_{\mathbf{x} \sim p_g} \log \left(1 - D_G^*(\mathbf{x})\right) \ ,$$

then p_g converges to p_{data}.

Recap: GANs

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

Theorem

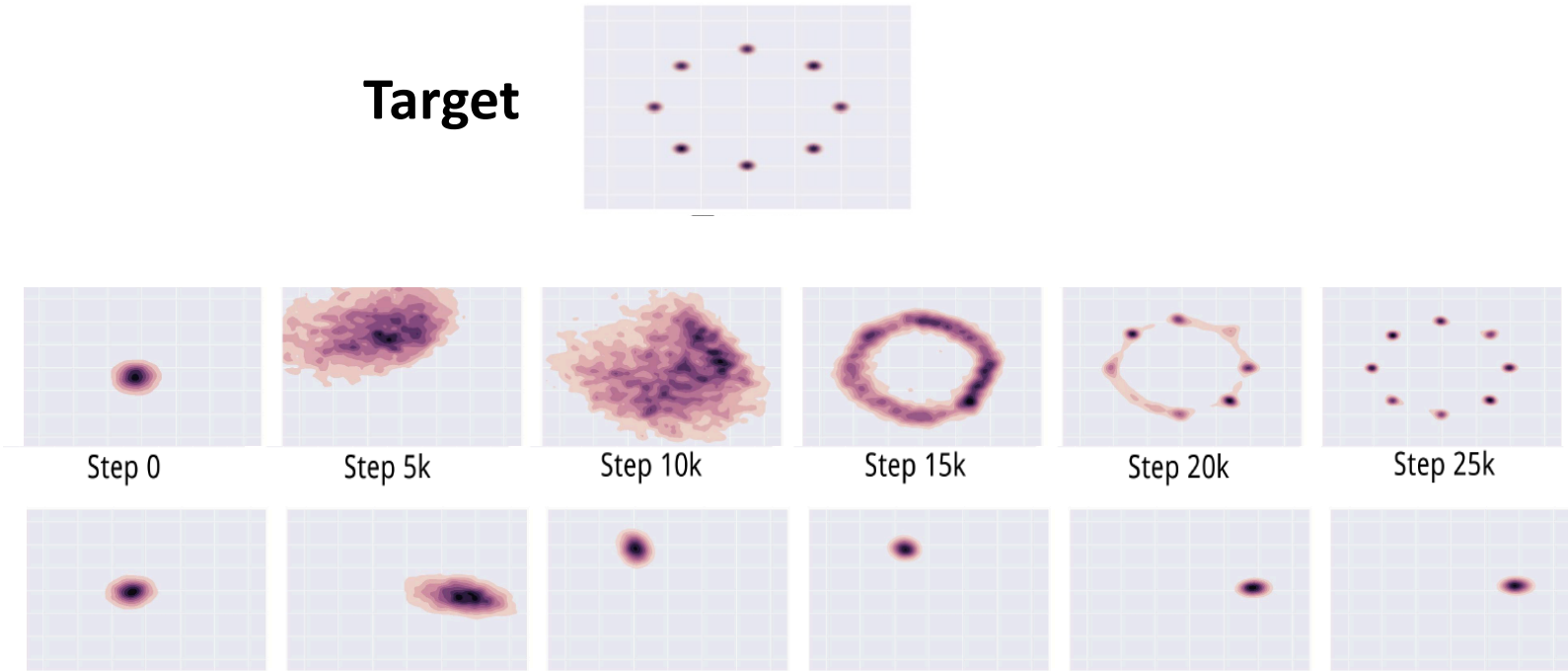
If G and D have enough capacity, and at each step of Algorithm 1, the discriminator is allowed to reach its optimum given G, and p_g is updated so as to improve the criterion

$$\mathbb{E}_{\mathbf{x} \sim p_{data}} \log D_G^*(\mathbf{x}) + \mathbb{E}_{\mathbf{x} \sim p_g} \log \left(1 - D_G^*(\mathbf{x})\right) \ ,$$

then p_g converges to p_{data}.

- Usually doesn't work in practice.

Mode Collapse in GAN



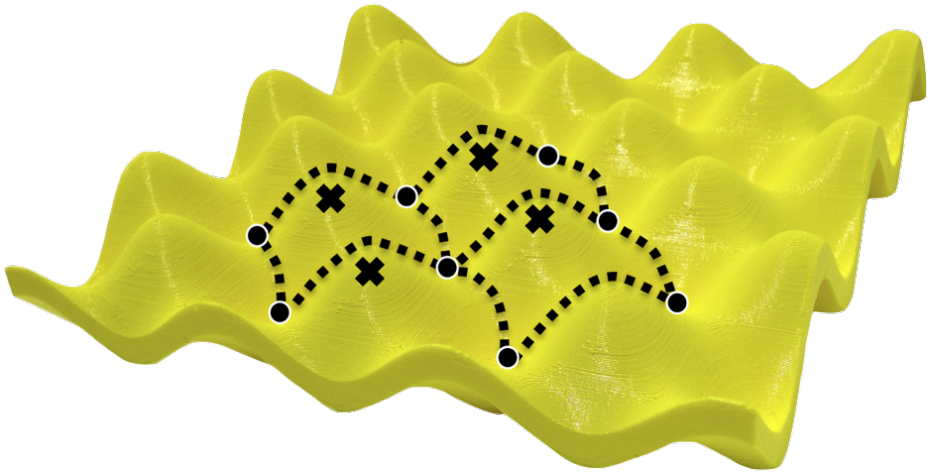
Metz, Luke, et al. **"Unrolled Generative Adversarial Networks."** arXiv preprint arXiv:1611.02163 (2016).

Mode Collapse in GAN

Reed, S., et al. ***Generating interpretable images with controllable structure.*** Technical report, 2016. 2, 2016.

Why Mode Collapse?

- Bad saddle points.



- Vanishing gradients in discriminator:
  - Wasserstein GAN.

GANs are about Distribution Matching

Minimizing Jensen-Shanon divergence between data distribution and generator distribution

$$G^* = \arg\min_G C(G) = \arg\min_G JSD(p_{\text{data}} \| p_g)$$

- Optimality when $p_g = p_{\text{data}}$.

GANs are about Distribution Matching

Minimizing Jensen-Shanon divergence between data distribution and generator distribution

$$G^* = \arg\min_G C(G) = \arg\min_G JSD(p_{\text{data}} \| p_g)$$

- Optimality when $p_g = p_{\text{data}}$.

What is wrong with the JSD?

Probability Measure

Definition (Probability Measure)

A probability measure is a real-valued function $\mathbb{P}$ defined on a set of events in a probability space $(\mathcal{X}, \Sigma)$ such that

- $0 \leq \mathbb{P}(A) \leq 1$ for all $A \in \Sigma$.
- $\mathbb{P}(\emptyset) = 0; \mathbb{P}(\mathcal{X}) = 1$.
- $\mathbb{P}(\cup_i A_i) = \sum_i \mathbb{P}(A_i)$ for all countable collections $\{A_i\}$ of pairwise disjoint sets.

Probability Measure

Definition (Probability Measure)

A probability measure is a real-valued function $\mathbb{P}$ defined on a set of events in a probability space $(\mathcal{X}, \Sigma)$ such that

- $0 \leq \mathbb{P}(A) \leq 1$ for all $A \in \Sigma$.
- $\mathbb{P}(\emptyset) = 0; \mathbb{P}(\mathcal{X}) = 1$.
- $\mathbb{P}(\cup_i A_i) = \sum_i \mathbb{P}(A_i)$ for all countable collections $\{A_i\}$ of pairwise disjoint sets.

- For a continuous distribution on space $\mathcal{X}$ with probability density function $p(\mathbf{x})$, the corresponding probability measure $\mathbb{P}$ is related to p, for $\mathbf{X} \in \mathcal{X}$, as

$$\mathbb{P}(\mathbf{X}) = \int_{\mathbf{X}} p(\mathbf{x}) \mathrm{d}\,\mathbf{x} \ .$$

Distance between Probability Measures

Let $\mathbb{P}_r$ and $\mathbb{P}_\theta$ be two probability measures:

- The Total Variation (TV) distance

$$\delta(\mathbb{P}_r, \mathbb{P}_\theta) \triangleq \sup_{A \in \Sigma} |\mathbb{P}_r(A) - \mathbb{P}_\theta(A)| \ .$$

- The KL divergence:

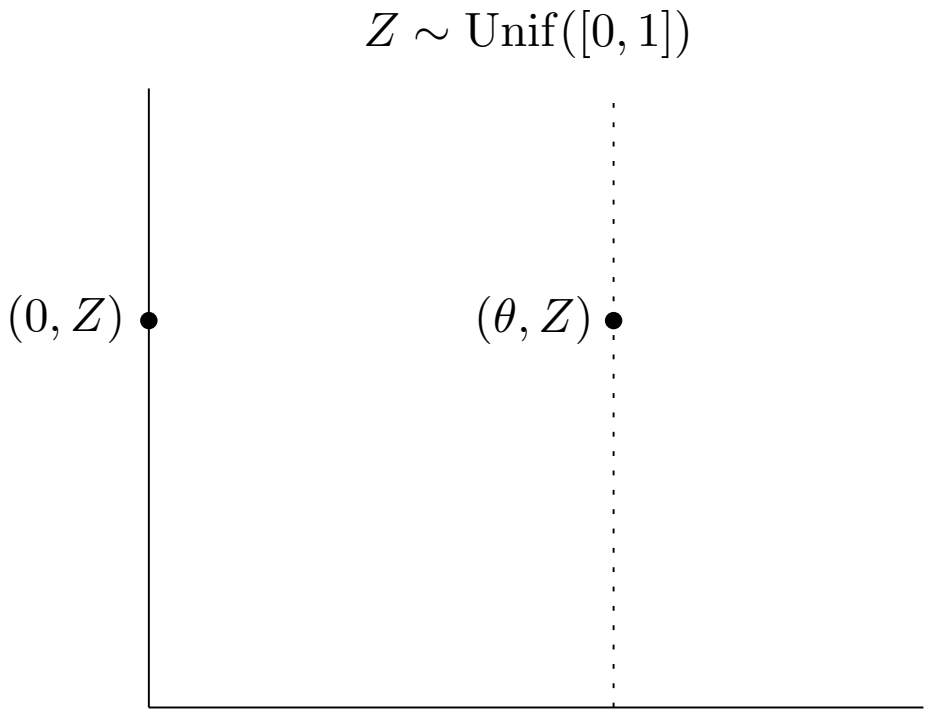
$$KL(\mathbb{P}_r \| \mathbb{P}_\theta) \triangleq \int \log \frac{\mathbb{P}_r(\mathbf{x})}{\mathbb{P}_\theta(\mathbf{x})} \mathbb{P}_r(\mathbf{x}) \mathrm{d}\,\mathbf{x}$$

- The Jensen-Shannon (JS) divergence (with $\mathbb{P}_m \triangleq (\mathbb{P}_r + \mathbb{P}_\theta)/2$):

$$JS(\mathbb{P}_r, \mathbb{P}_\theta) \triangleq \frac{1}{2} KL(\mathbb{P}_r \| \mathbb{P}_m) + \frac{1}{2} KL(\mathbb{P}_\theta \| \mathbb{P}_m) \ .$$

Distance between Probability Measures (Optional)

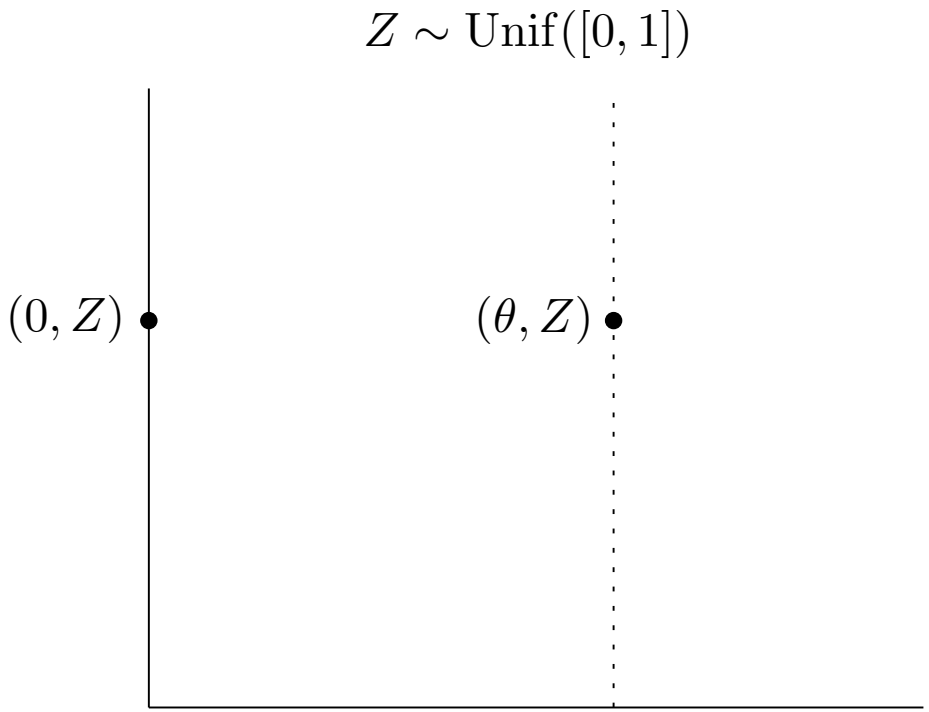
- Space $\mathcal{X}$ is the collection of lines in the two dimensional space $\mathbb{R} \times \mathbb{R}$.
- $\mathbb{P}_r$ only assigns uniform probabilities for $(0, Z)$, all others have probabilities 0.
- $\mathbb{P}_\theta$ only assigns uniform probabilities for (θ, Z) for some fixed θ, all others have probabilities 0.



- $\delta(\mathbb{P}_r, \mathbb{P}_\theta) = ?$
- $KL(\mathbb{P}_r || \mathbb{P}_\theta) = ?$
- $JS(\mathbb{P}_r, \mathbb{P}_\theta) = ?$

Distance between Probability Measures (Optional)

- Space $\mathcal{X}$ is the collection of lines in the two dimensional space $\mathbb{R} \times \mathbb{R}$.
- $\mathbb{P}_r$ only assigns uniform probabilities for $(0, Z)$, all others have probabilities 0.
- $\mathbb{P}_\theta$ only assigns uniform probabilities for (θ, Z) for some fixed θ, all others have probabilities 0.



- $\delta(\mathbb{P}_r, \mathbb{P}_\theta) = \begin{cases} 1, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$
- $KL(\mathbb{P}_r \| \mathbb{P}_\theta) = \begin{cases} \infty, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$, $JS(\mathbb{P}_r, \mathbb{P}_\theta) = \begin{cases} \log 2, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$

What is the Problem with these Metrics

- $\delta(\mathbb{P}_r, \mathbb{P}_\theta) = \begin{cases} 1, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$
- $KL(\mathbb{P}_r \| \mathbb{P}_\theta) = \begin{cases} \infty, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$, $JS(\mathbb{P}_r, \mathbb{P}_\theta) = \begin{cases} \log 2, & \text{if } \theta \neq 0 \\ 0, & \text{if } \theta = 0 \end{cases}$

They are independent of θ

- Derivatives w.r.t. θ equal to zero.
- Gradient vanishing!

Instability of GAN

- Original objective

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

- The optimal discriminator

$$D^*(\mathbf{x}) = \frac{\mathbb{P}_d(\mathbf{x})}{\mathbb{P}_d(\mathbf{x}) + \mathbb{P}_g(\mathbf{x})}$$

$$\text{and } L(D^*, G_{\theta_g}) = 2JS(\mathbb{P}_d, \mathbb{P}_g) - 2\log 2 \,.$$

- $\mathbb{P}_r = \mathbb{P}_d$ and $\mathbb{P}_\theta = \mathbb{P}_g$ in GANs.

Instability of GAN

Theorem

Let $\mathbb{P}_d$ and $\mathbb{P}_g$ be two distributions that have support contained in two closed manifolds $\mathcal{M}$ and $\mathcal{P}$ that don't perfectly align and don't have full dimension. We further assume that $\mathcal{M}$ and $\mathcal{P}$ are continuous in their respective manifolds. Then, there exists an optimal discriminator $D^* : \mathcal{X} \to [0, 1]$ that has accuracy 1 and for almost any $\mathbf{x}$ in $\mathcal{M}$ or $\mathcal{P}$, D^* is smooth in a neighborhood of $\mathbf{x}$ and $\nabla_{\mathbf{x}} D^*(\mathbf{x}) = 0$.

Instability of GAN

Theorem

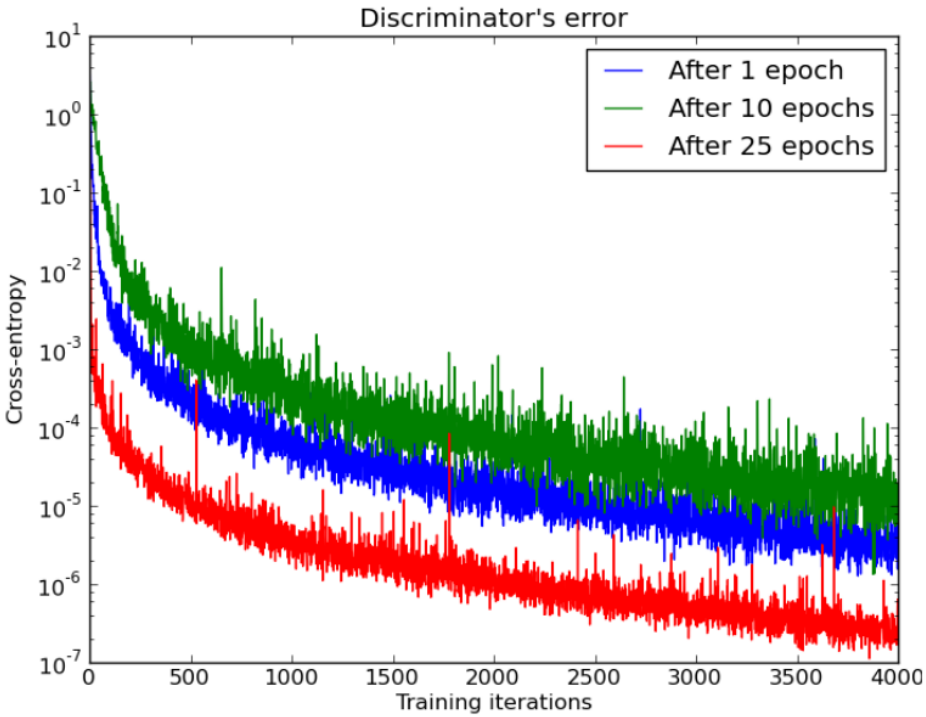
Let $\mathbb{P}_d$ and $\mathbb{P}_g$ be two distributions that have support contained in two closed manifolds $\mathcal{M}$ and $\mathcal{P}$ that don't perfectly align and don't have full dimension. We further assume that $\mathcal{M}$ and $\mathcal{P}$ are continuous in their respective manifolds. Then, there exists an optimal discriminator $D^* : \mathcal{X} \to [0, 1]$ that has accuracy 1 and for almost any $\mathbf{x}$ in $\mathcal{M}$ or $\mathcal{P}$, D^* is smooth in a neighborhood of $\mathbf{x}$ and $\nabla_{\mathbf{x}} D^*(\mathbf{x}) = 0$.

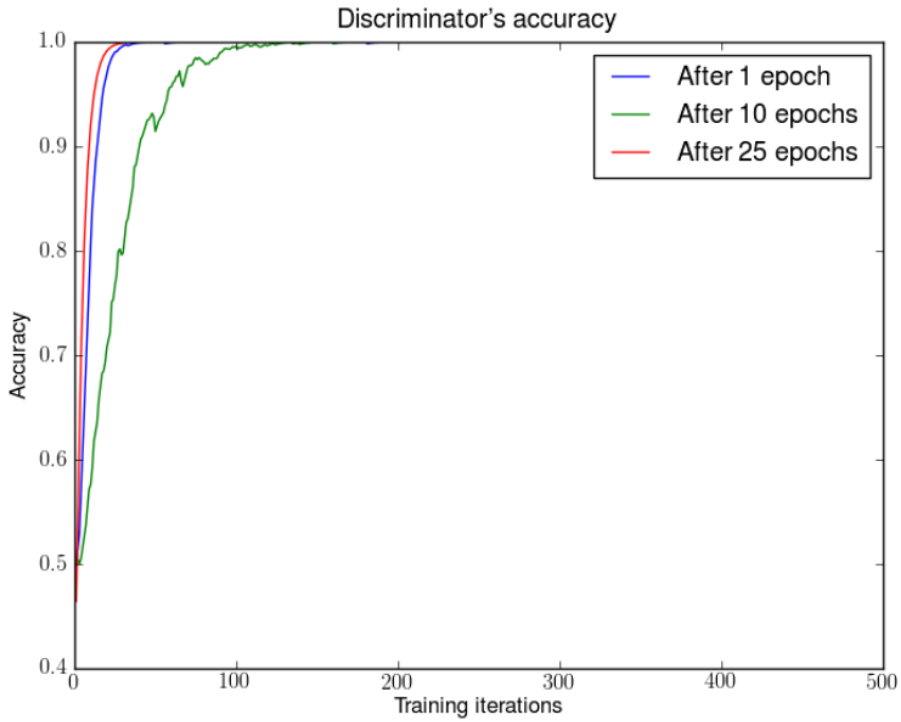
Theorem (Vanishing gradients on the generator)

If $\|D - D^*\| < \epsilon$, then

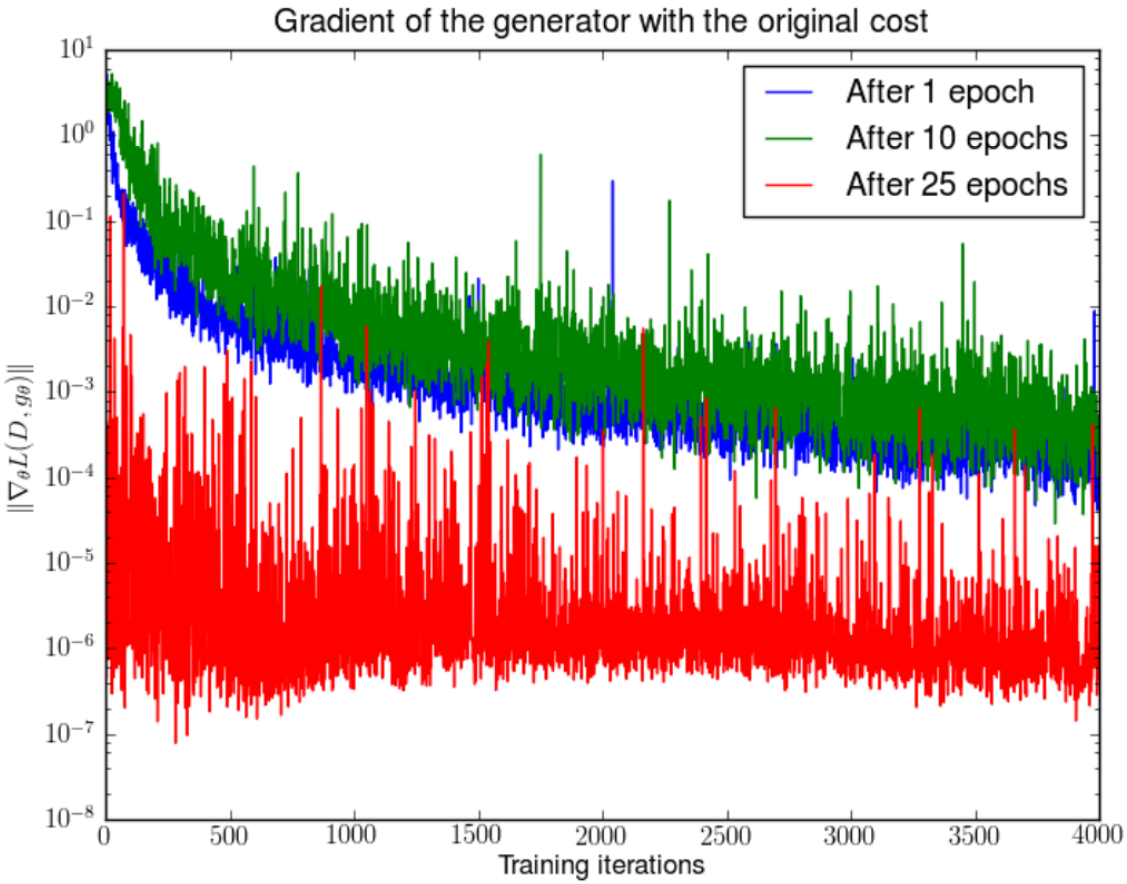
$$\left\|\nabla_{\theta_g} \mathbb{E}_{z \sim p(z)} \left[\log\left(1 - D(G(z))\right)\right]\right\|_2 < M \frac{\epsilon}{1 - \epsilon} .$$

Instability of GAN





Instability of GAN



Wasserstein Distance

- The Earth-Mover's (EM) distance or Wasserstein-1 distance

$$W(\mathbb{P}_r, \mathbb{P}_\theta) \triangleq \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma} \left| x - y \right| \ ,$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_\theta)$ denotes the set of all joint distributions $\gamma(x, y)$ such whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_\theta$, respectively.

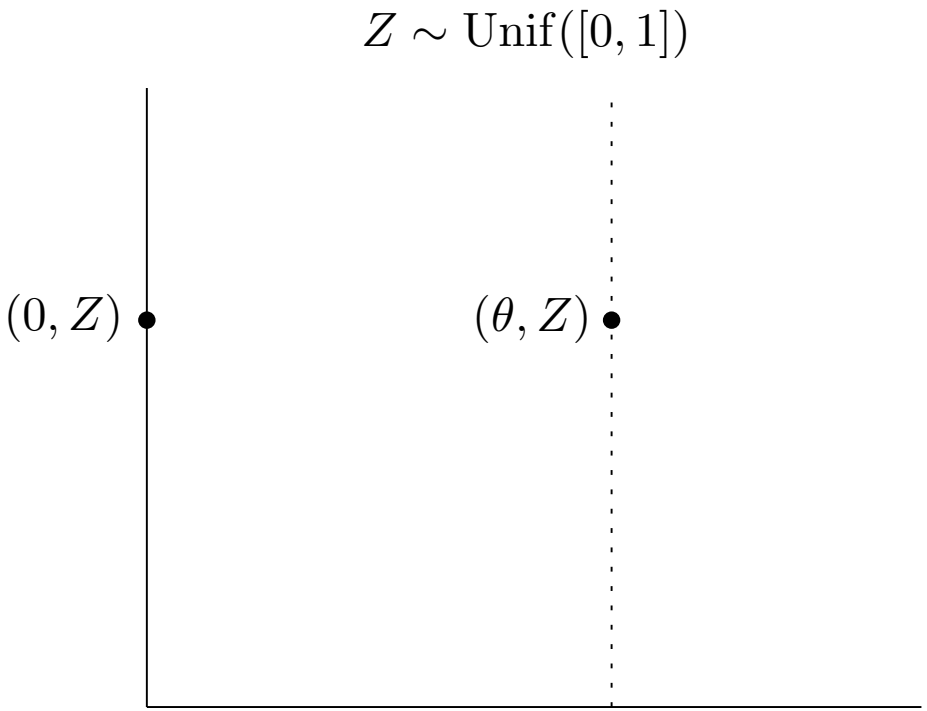
Wasserstein Distance

- The Earth-Mover's (EM) distance or Wasserstein-1 distance

$$W(\mathbb{P}_r, \mathbb{P}_\theta) \triangleq \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma} |x - y| \ ,$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_\theta)$ denotes the set of all joint distributions $\gamma(x, y)$ such whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_\theta$, respectively.

- $W(\mathbb{P}_r, \mathbb{P}_\theta) = ?$



Wasserstein Distance

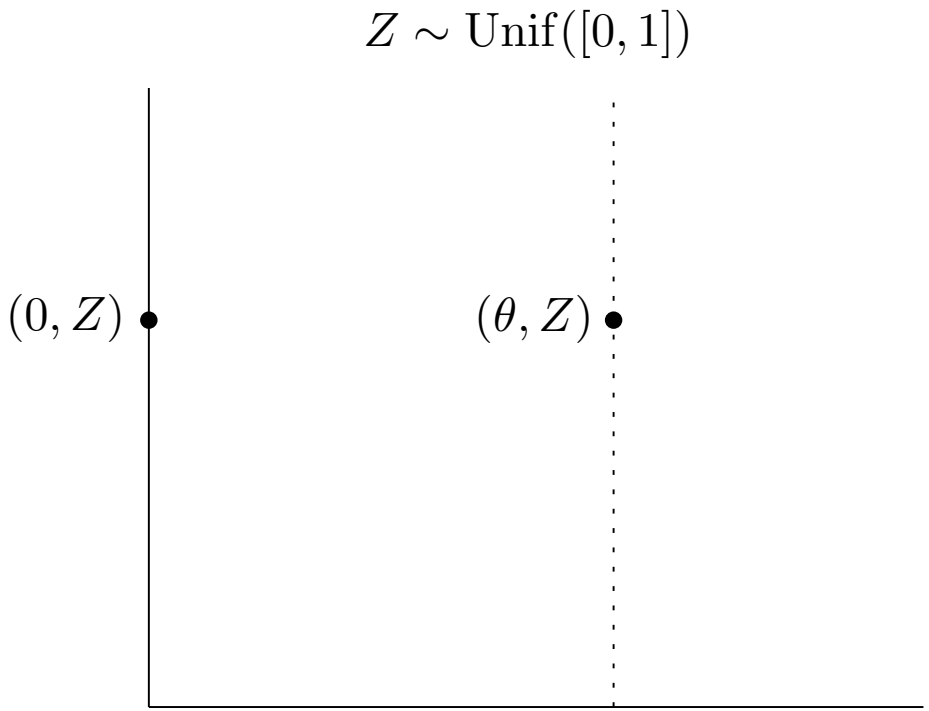
- The Earth-Mover's (EM) distance or Wasserstein-1 distance

$$W(\mathbb{P}_r, \mathbb{P}_\theta) \triangleq \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma} |x - y| \ ,$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_\theta)$ denotes the set of all joint distributions $\gamma(x, y)$ such whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_\theta$, respectively.

- $W(\mathbb{P}_r, \mathbb{P}_\theta) = |\theta|$

Continuous and differentiable w.r.t. θ.



Using Wasserstein Distance in GANs

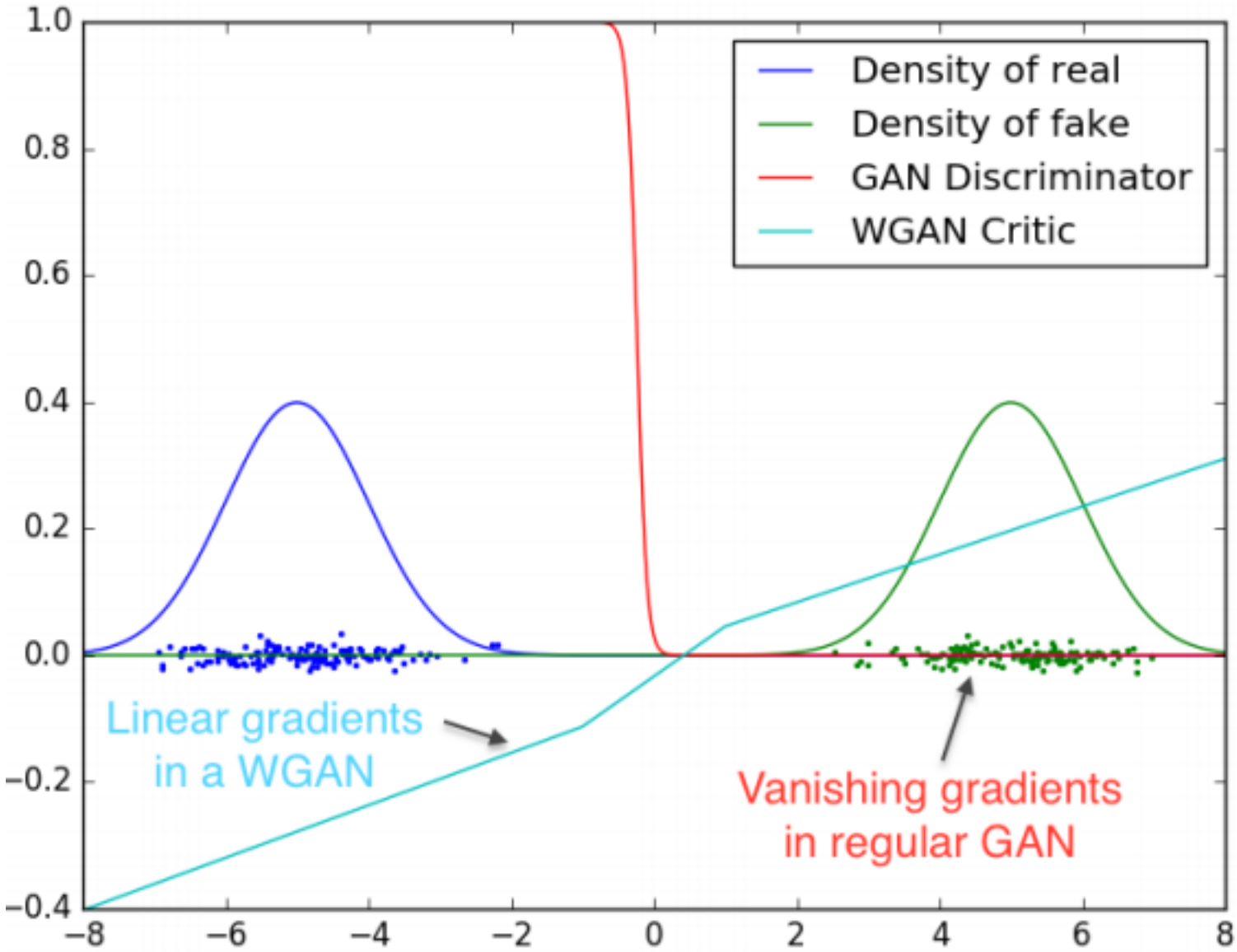
Theorem

When replacing the JS divergence with the Wasserstein distance in GAN:

1. *If G_{θ_g} is continuous in θ_g, so is $W(\mathbb{P}_d, \mathbb{P}_g)$.*
2. *If G_{θ_g} is locally Lipschitz and satisfies some regularity assumption, then $W(\mathbb{P}_d, \mathbb{P}_g)$ is continuous everywhere, and differentiable almost everywhere.*
3. *1 and 2 are false for the Jensen-Shannon and KL divergences.*

- If we choose G_{θ_g} to be any feedforward neural network parametrized by θ_g, and $p(z)$ to be $\mathbb{E}[\|z\|] < \infty$, then the regularity assumption is satisfied.

Using Wasserstein Distance in GANs



Implementing WGAN

$$W(\mathbb{P}_r, \mathbb{P}_{\boldsymbol{\theta}}) \triangleq \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_{\boldsymbol{\theta}})} \mathbb{E}_{(x,y)\sim\gamma} \left| x - y \right| \ ,$$

Implementing WGAN

$$W(\mathbb{P}_r, \mathbb{P}_{\boldsymbol{\theta}}) \triangleq \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_{\boldsymbol{\theta}})} \mathbb{E}_{(x,y)\sim\gamma} \left| x - y \right| \ ,$$

- By the Kantorovich-Rubinstein duality:

$$W(\mathbb{P}_r, \mathbb{P}_{\boldsymbol{\theta}}) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_d} \left[f(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_g} \left[f(\mathbf{x}) \right] \ ,$$

  where the supreme is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$.

Implementing WGAN

$$W(\mathbb{P}_r, \mathbb{P}_\theta) \triangleq \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_\theta)} \mathbb{E}_{(x,y)\sim\gamma} |x - y| \ ,$$

- By the Kantorovich-Rubinstein duality:

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_d} [f(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_g} [f(\mathbf{x})] \ ,$$

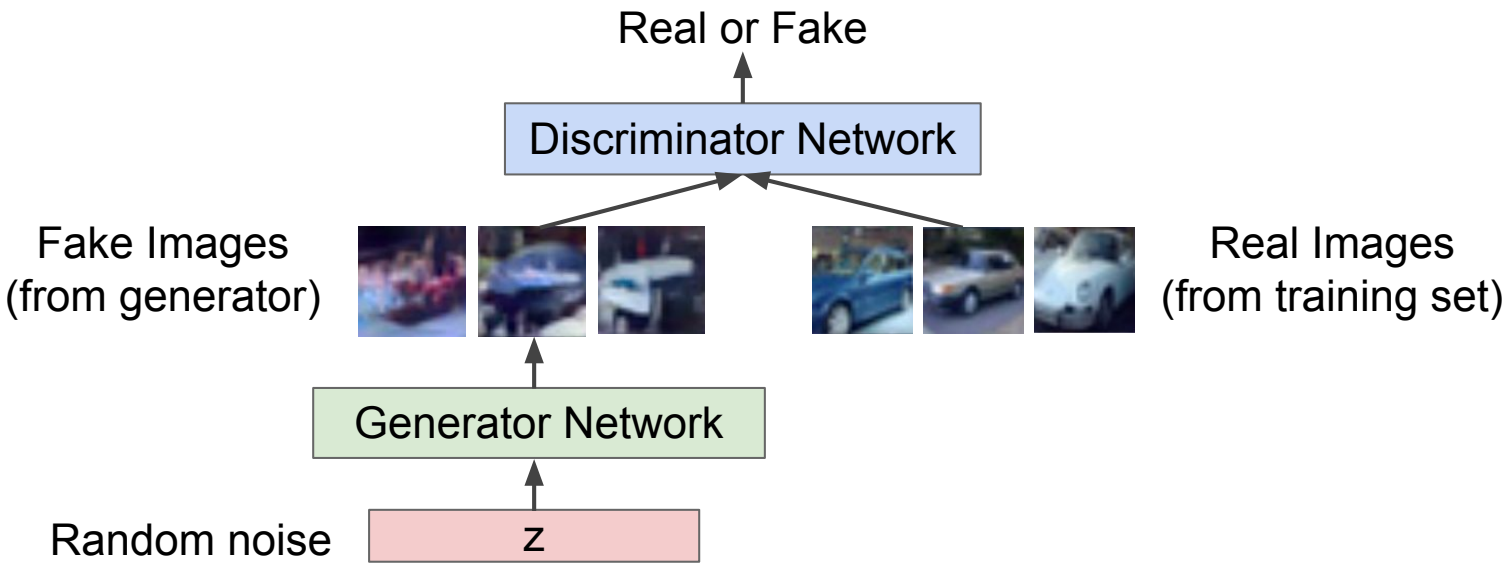
  where the supreme is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$.

Theorem

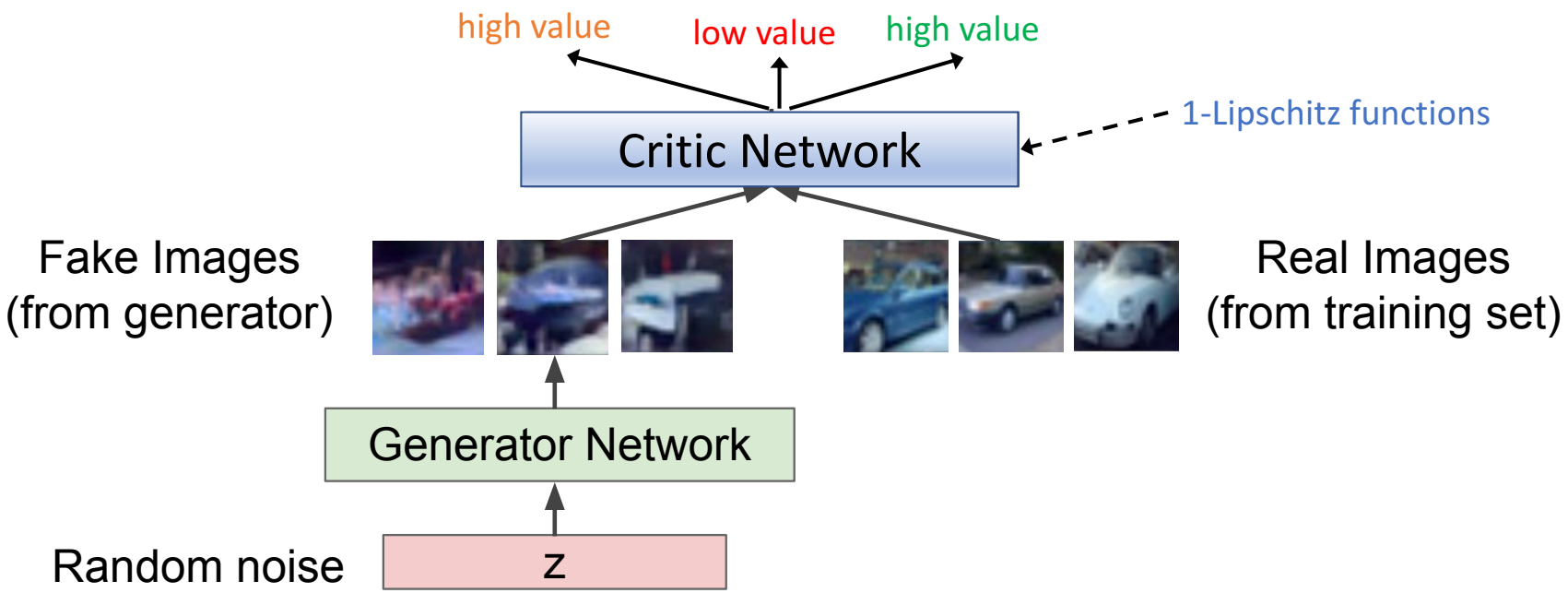
There exists a solution to the above maximization problem, and we have

$$\nabla_{\theta_g} W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{z\sim p(z)} \left[\nabla_{\theta_g} f(G(z))\right] \ .$$

GAN and WGAN


Real or Fake
Discriminator Network
Fake Images
(from generator)
Real Images
(from training set)
Generator Network
Random noise
z

GAN and WGAN


high value
low value
high value
Critic Network
1-Lipschitz functions
Fake Images
(from generator)
Real Images
(from training set)
Generator Network
Random noise
z

WGAN Algorithm

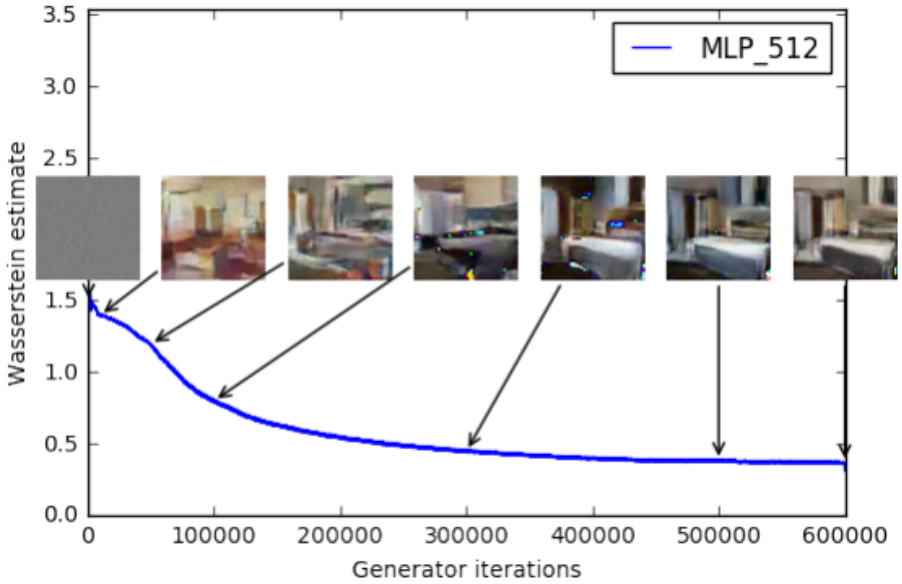
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

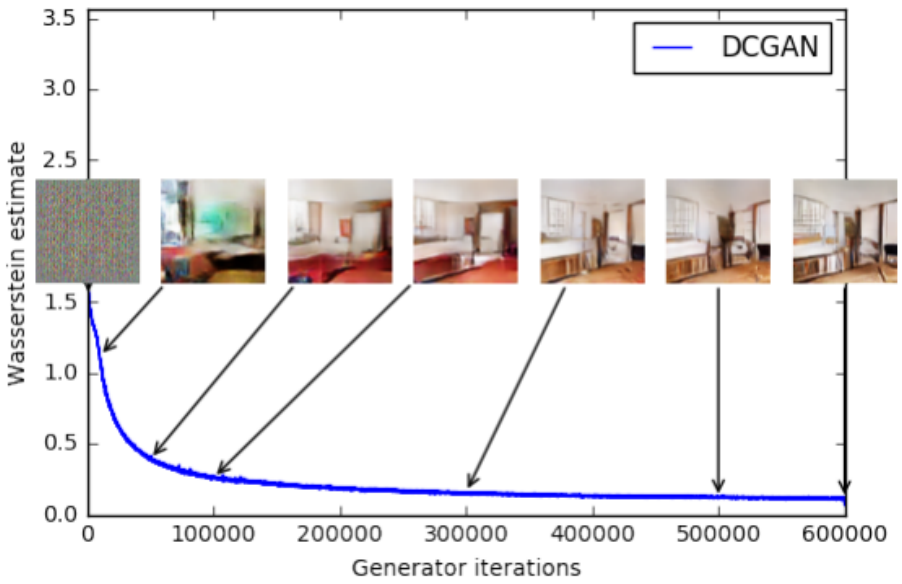
Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

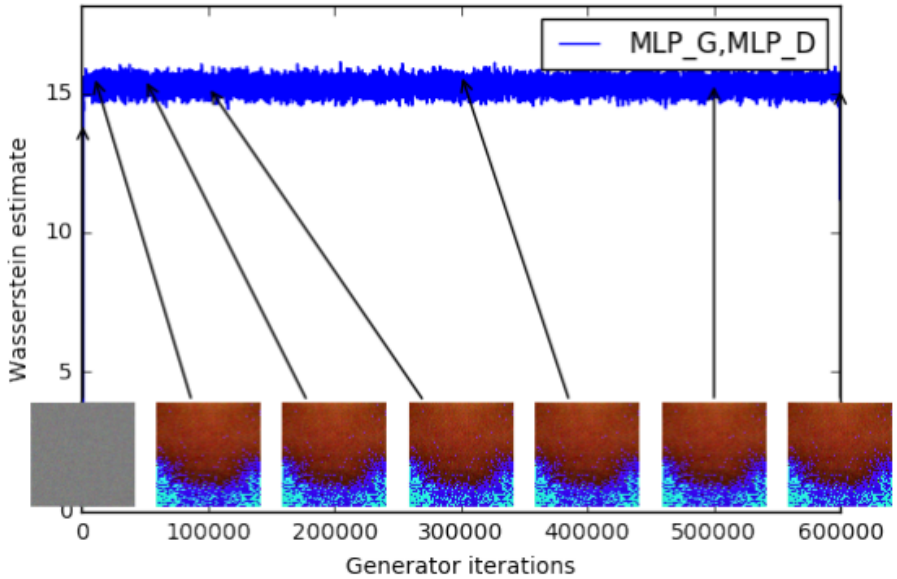
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

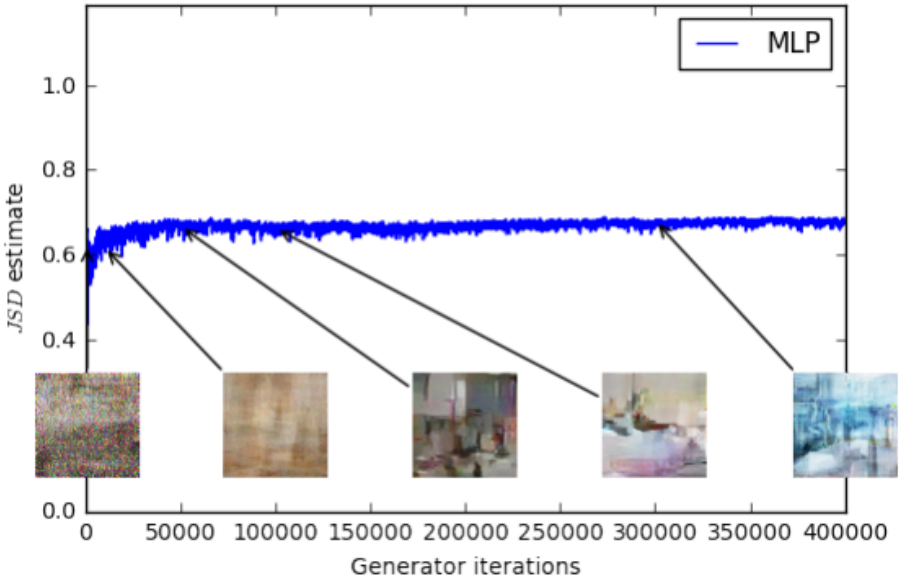
WGAN Experiments: Wasserstein Distance

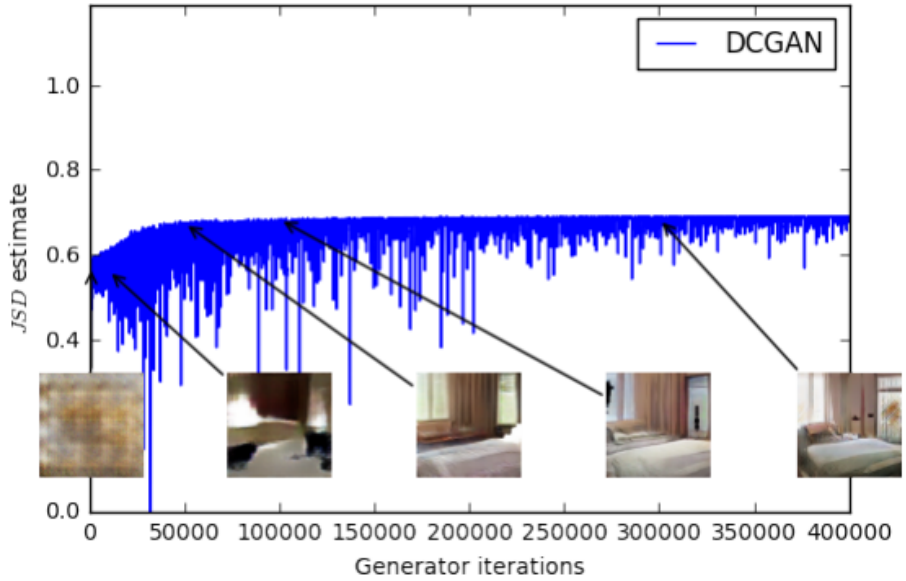


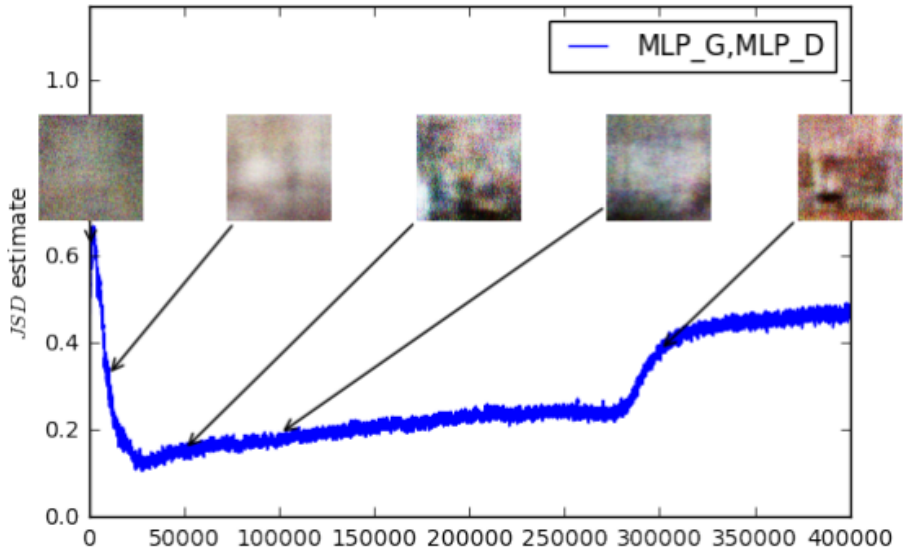




WGAN Experiments: Jason-Shanon Divergence

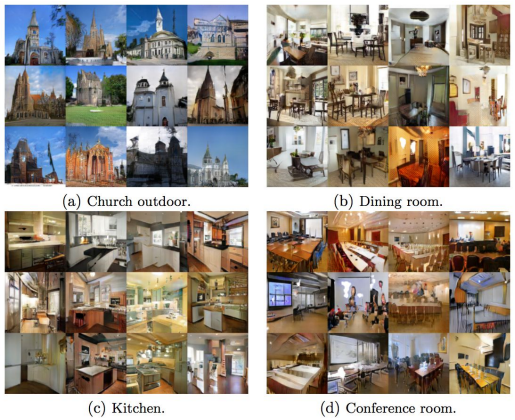






2017: Year of the GAN

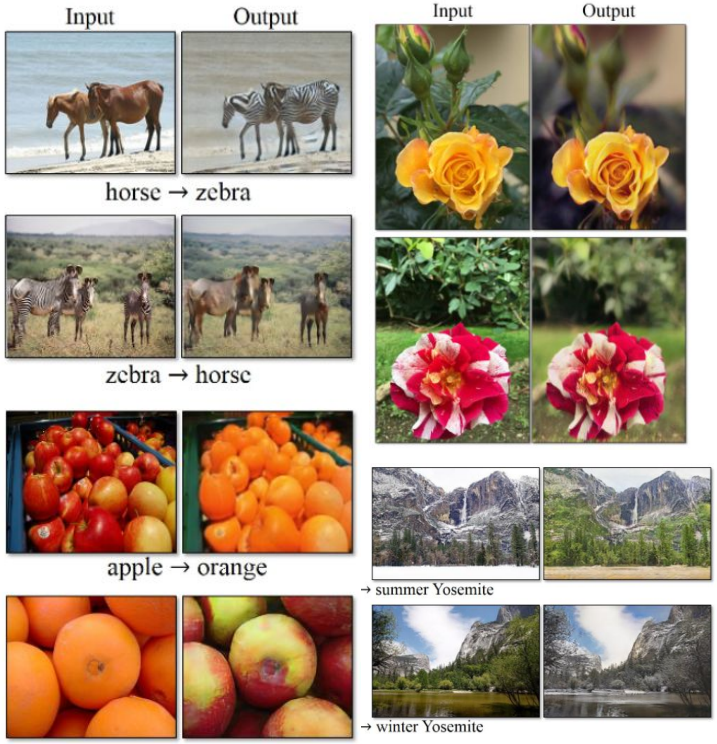
Better training and generation



LSGAN. Mao et al. 2017.

BEGAN. Bertholet et al. 2017.

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

Akata et al. 2017.

Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

The GAN Zoo

https://github.com/hindupuravinash/the-gan-zoo
https://github.com/soumith/ganhacks

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

The GAN Zoo

https://github.com/hindupuravinash/the-gan-zoo
https://github.com/soumith/ganhacks

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

Summary

GANs do not learn explicit density functions, they learn transformations from simple distributions to training-data distributions by taking a game-theoretic approach.

1. Pros:
   - ⋆ Beautiful, state-of-the-art samples.
2. Cons:
   - ⋆ Trickier / more unstable to train.
3. Active areas of research:
   - ⋆ Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others).
   - ⋆ Conditional GANs, GANs for all kinds of applications.
   - ⋆ Adversarial idea for training.