Deep generative modeling

Jakob Verbeek
INRIA

January 2019

Plan for this lecture

1. Introduction and motivation
2. Generative adversarial networks
3. Variational autoencoders
4. Autoregressive models

Success stories of deep learning in recent years

- Convolutional neural networks
- For stationary signals such as audio, images, and video
- Applictions: Object detection, semantic segmentation, image retrieval, pose estimation, ...

Success stories of deep learning in recent years

- Convolutional neural networks
- For stationary signals such as audio, images, and video
- Applictions: Object detection, semantic segmentation, image retrieval, pose estimation, ...

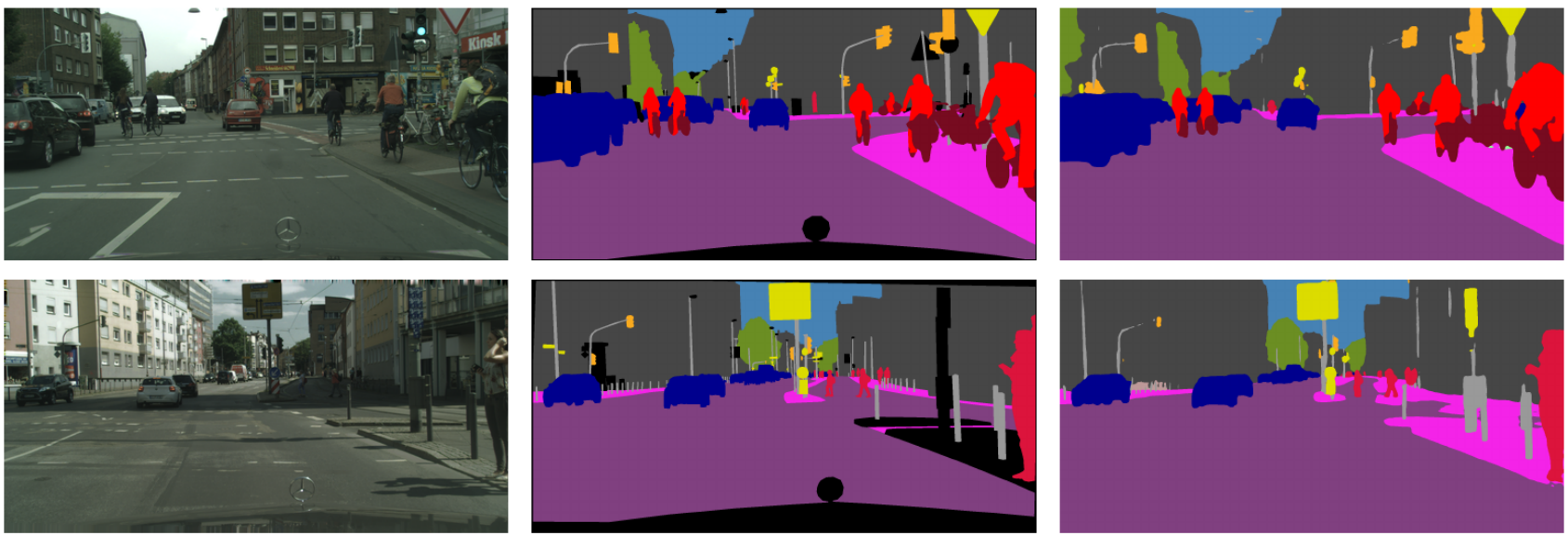
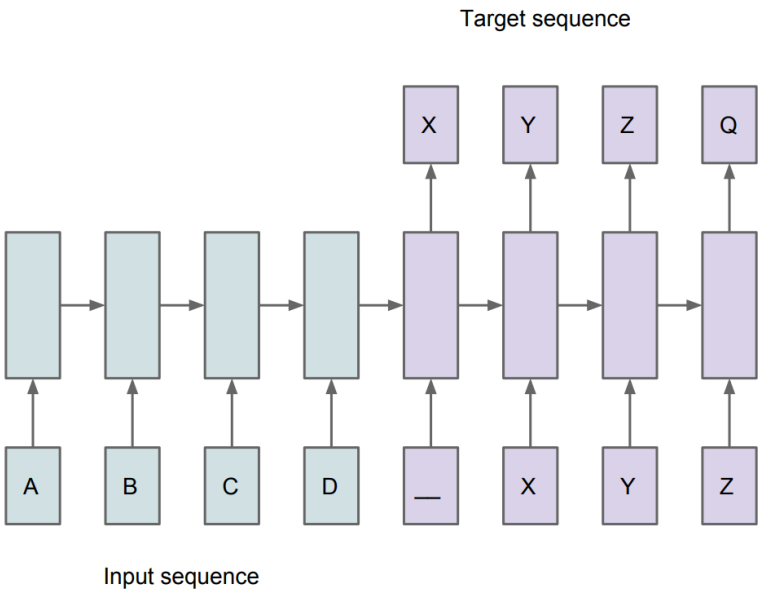


Figure from [Lin et al., 2017]

Success stories of deep learning in recent years

- Recurrent neural networks
- For variable length sequence data, *e.g.* in natural language
- Applications: Machine translation, image captioning, speech recognition, . . .



- **FR**: Les avionneurs se querellent au sujet de la largeur des sièges alors que de grosses commandes sont en jeu
- **GT**: Aircraft manufacturers are quarreling about the seat width as large orders are at stake
- **LSTM**: Aircraft manufacturers are concerned about the width of seats while large orders are at stake

Example from Ilya Sutskever

It's all about the features

It's all about the features

- Conventional vision / audio processing approach
  1. Features (engineered) : SIFT, MFCC, ...
  2. Pooling (unsupervised): bag-of-words, Fisher vectors, ...
  3. Recognition (supervised): linear/kernel classifier, ...

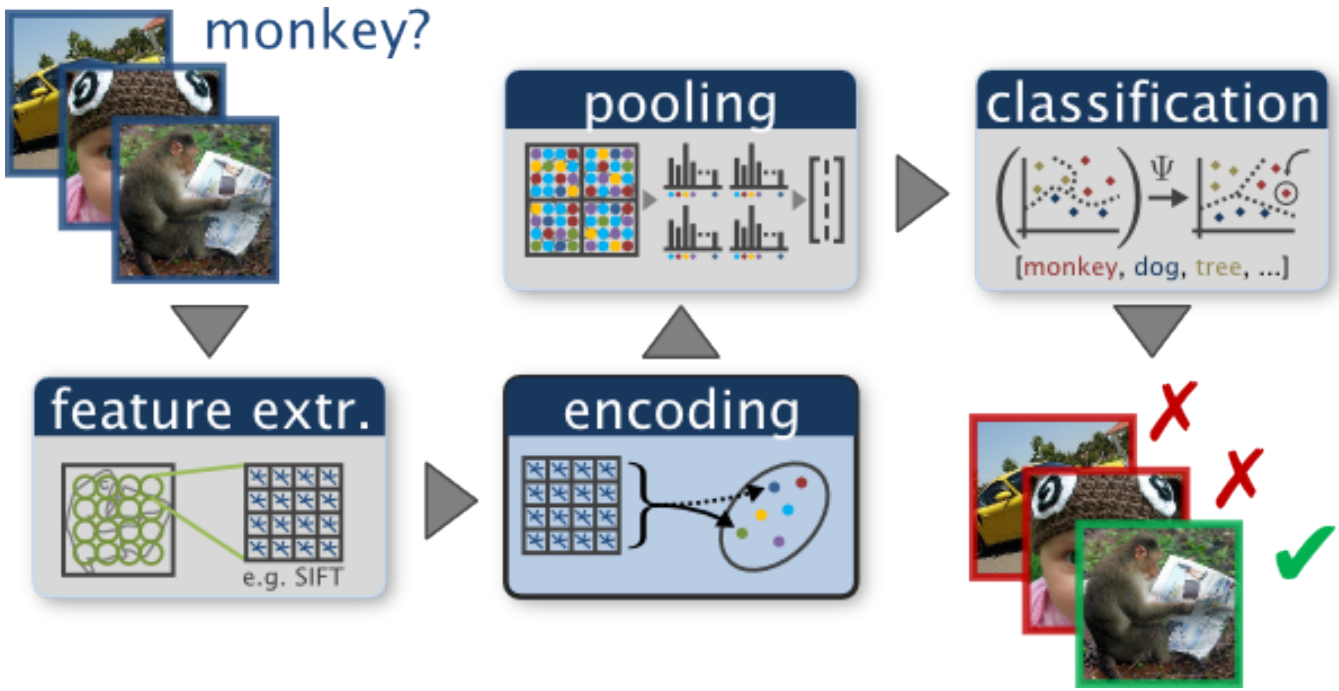
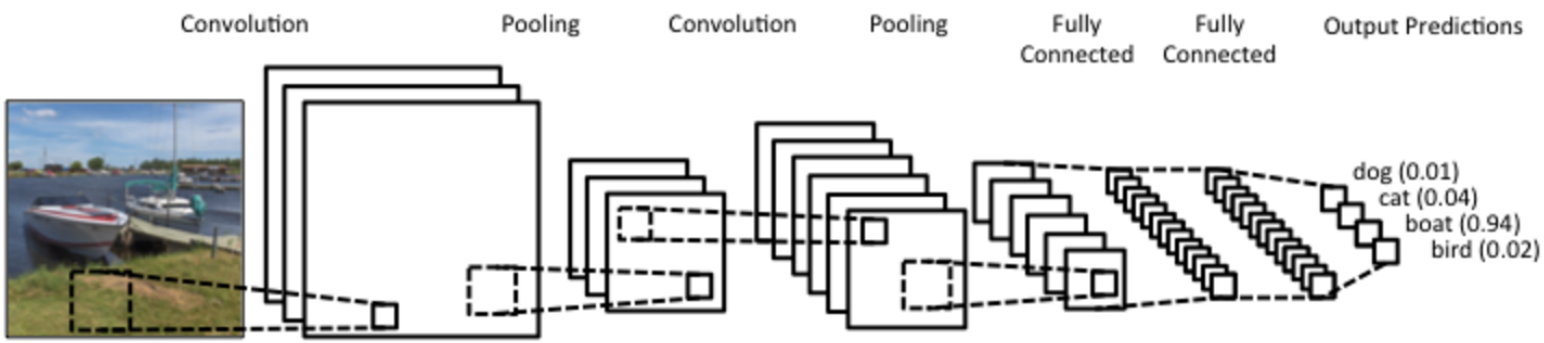


Image from [Chatfield et al., 2011]

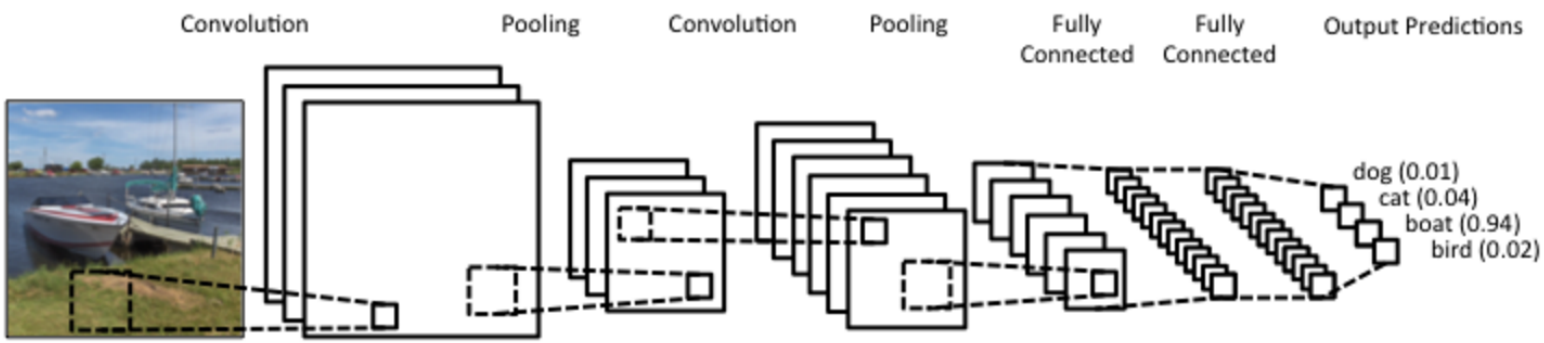
It's all about the features

- Deep learning blurs boundary feature / classifier
  - Stacks simple non-linear transformations
  - Learns progressively more abstract representation
  - Starts from raw input signal, *e.g.* image pixels



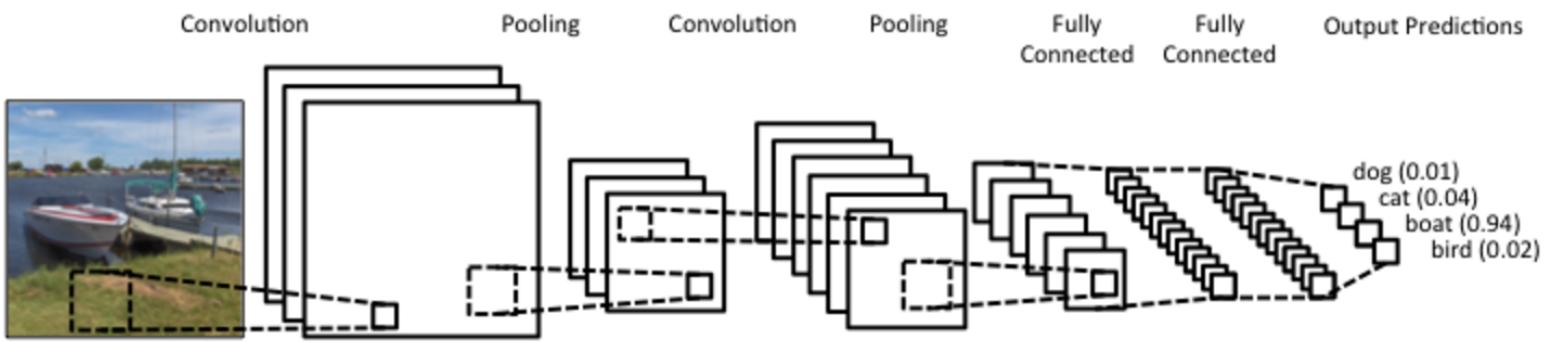
It's all about the features

- Deep learning blurs boundary feature / classifier
  - Stacks simple non-linear transformations
  - Learns progressively more abstract representation
  - Starts from raw input signal, *e.g.* image pixels
- End-to-end training to minimize a task-specific loss



It's all about the features

- Deep learning blurs boundary feature / classifier
  - Stacks simple non-linear transformations
  - Learns progressively more abstract representation
  - Starts from raw input signal, *e.g.* image pixels
- End-to-end training to minimize a task-specific loss
- Supervised learning from lots of labeled data



Motivations for unsupervised (deep) learning

Motivations for unsupervised (deep) learning

1. **Improve supervised learning from few samples**
   - Unlabeled data often abundantly available
   - Learn representations/features from unlabeled data

Motivations for unsupervised (deep) learning

1. **Improve supervised learning from few samples**
   - Unlabeled data often abundantly available
   - Learn representations/features from unlabeled data

2. **Generative models for image and other complex data**

Motivations for unsupervised (deep) learning

1. **Improve supervised learning from few samples**
   - Unlabeled data often abundantly available
   - Learn representations/features from unlabeled data

2. **Generative models for image and other complex data**
   - Unconditional: sandbox research problem (?)

Motivations for unsupervised (deep) learning

1. **Improve supervised learning from few samples**
   - Unlabeled data often abundantly available
   - Learn representations/features from unlabeled data

2. **Generative models for image and other complex data**
   - Unconditional: sandbox research problem (?)
   - Conditional structured prediction: in-painting, colorization, text-to-image, video forecasting, *etc.*

Image colorization [Royer et al., 2017]

(Un)supervised learning and un(conditional) models

(Un)supervised learning and un(conditional) models

- Supervised learning: model conditional distribution $p_\theta(y|\mathbf{x})$
  - For example: $\mathbf{x}$ an image, y a class label

$$\max_\theta \sum_{(\mathbf{x},y)\sim\mathcal{D}} \ln p_\theta(y|\mathbf{x}) \tag{1}$$

  - $\mathcal{D}$: data generating distribution
  - θ: model parameters

(Un)supervised learning and un(conditional) models

- Supervised learning: model conditional distribution $p_\theta(y|\mathbf{x})$
  - For example: $\mathbf{x}$ an image, y a class label

$$\max_\theta \sum_{(\mathbf{x},y)\sim\mathcal{D}} \ln p_\theta(y|\mathbf{x}) \tag{1}$$

  - $\mathcal{D}$: data generating distribution
  - θ: model parameters

- Unsupervised learning: model unconditional distribution $p(\mathbf{x})$
  - For example: $\mathbf{x}$ an image

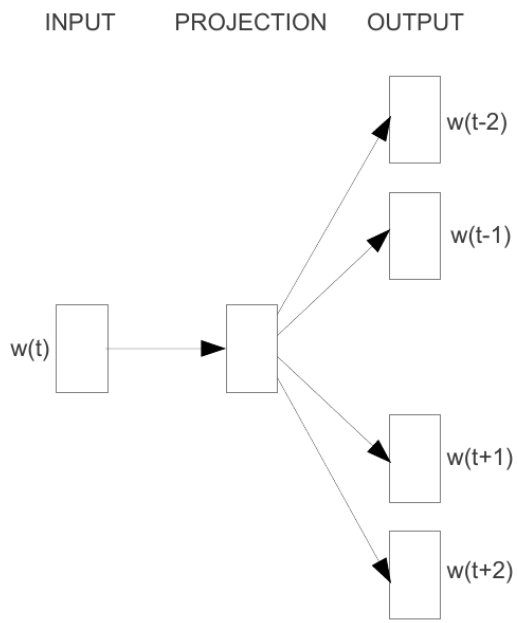
$$\max_\theta \sum_{\mathbf{x}\sim\mathcal{D}} \ln p_\theta(\mathbf{x}) \tag{2}$$

Self-supervised learning

- Learning conditional models $p(y|\mathbf{x})$ from unlabeled data

Self-supervised learning

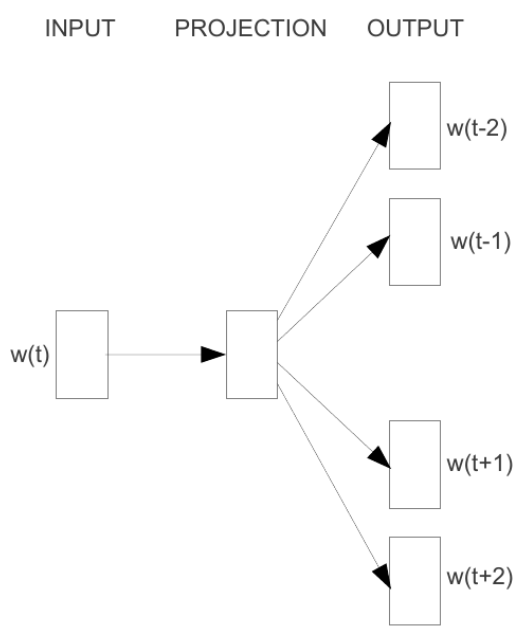
- Learning conditional models $p(y|\mathbf{x})$ from unlabeled data
- Prediction of structural data properties
  - Skip-gram language models (word2vec) [Mikolov et al., 2013]



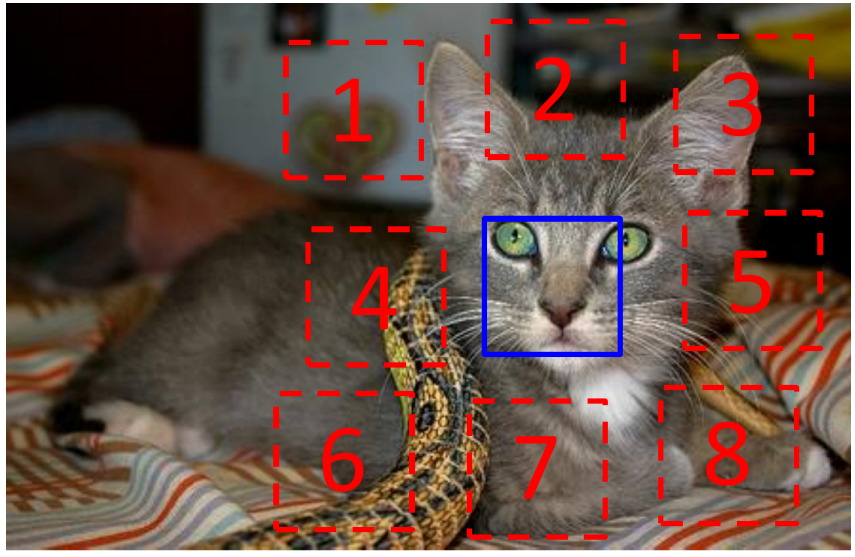
Skip-gram

Self-supervised learning

- Learning conditional models $p(y|\mathbf{x})$ from unlabeled data
- Prediction of structural data properties
  - Skip-gram language models (word2vec) [Mikolov et al., 2013]
  - Relative position of image patches [Doersch et al., 2015]



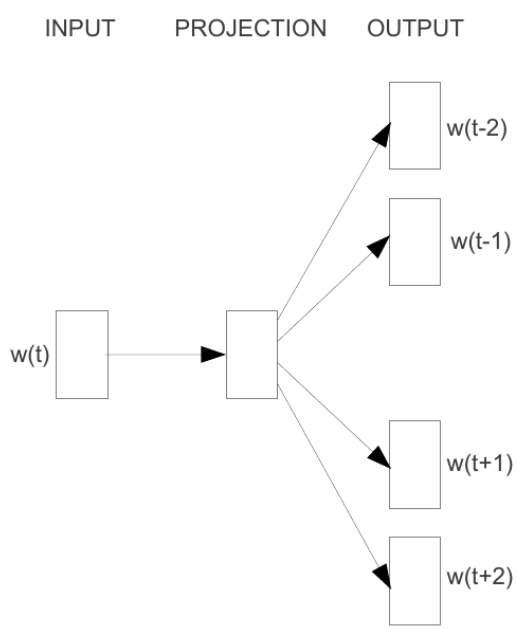
Skip-gram



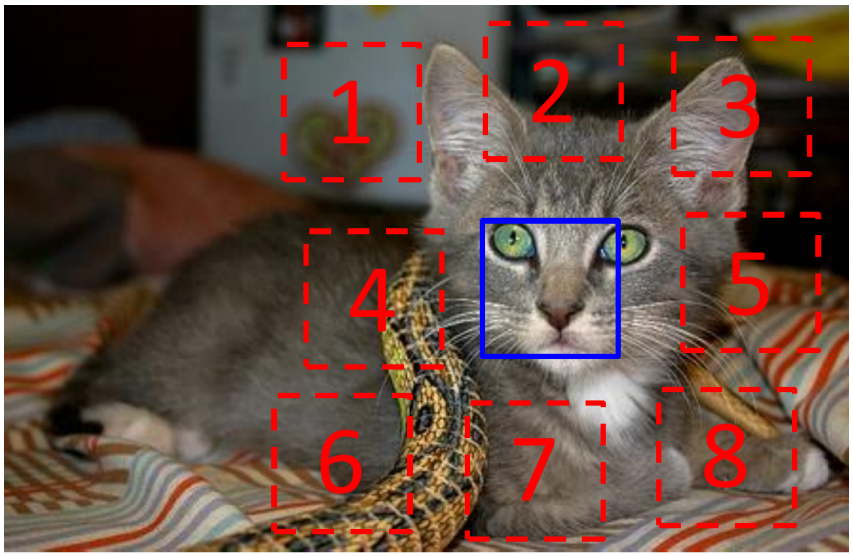
X = (,); Y = 3

Self-supervised learning

- Learning conditional models $p(y|\mathbf{x})$ from unlabeled data
- Prediction of structural data properties
  - Skip-gram language models (word2vec) [Mikolov et al., 2013]
  - Relative position of image patches [Doersch et al., 2015]
  - Relative ordering of video frames [Fernando et al., 2017]



Skip-gram

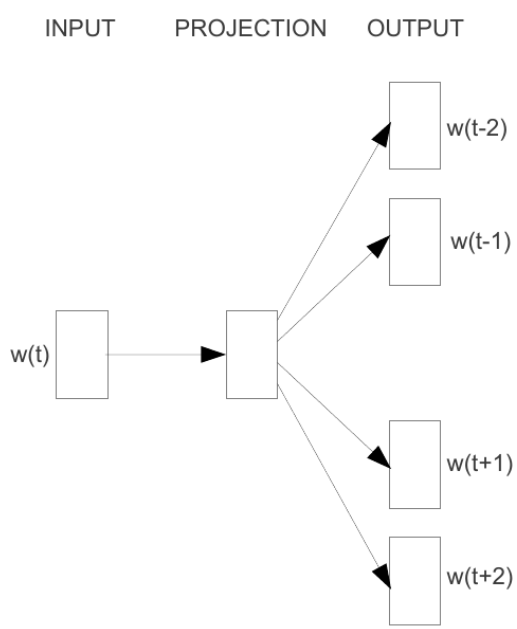


X = (,); Y = 3

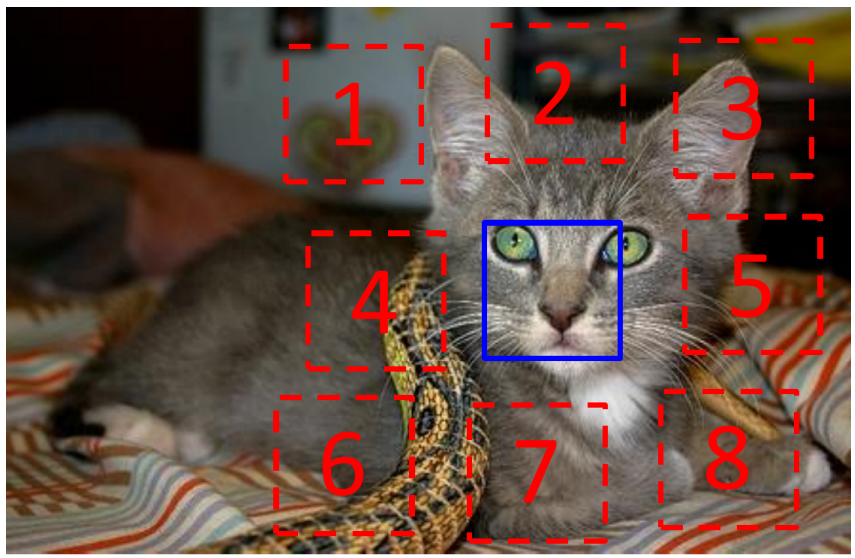
Self-supervised learning

- Learning conditional models $p(y|\mathbf{x})$ from unlabeled data

- Prediction of structural data properties
  - Skip-gram language models (word2vec) [Mikolov et al., 2013]
  - Relative position of image patches [Doersch et al., 2015]
  - Relative ordering of video frames [Fernando et al., 2017]
  - Image inpainting [Pathak et al., 2016]
  - ...



Skip-gram



X = (,); Y = 3

Self-supervised learning to prime supervised learning

- Supervised pre-training of network on proxy-task
- Fine-tune on final task with limited training data

Self-supervised learning to prime supervised learning

- Supervised pre-training of network on proxy-task
- Fine-tune on final task with limited training data
- Example: features for action recognition [Fernando et al., 2017]

Method	UCF101-split1	HMDB51-split1
DrLim [17]	45.7	16.3
TempCoh [32]	45.4	15.9
Obj. Patch [44]	40.7	15.6
Seq. Ver. [31]	50.9	19.8
Our - Stack-of-Diff.	**60.3**	**32.5**
Rand weights - Stack-of-Diff.	51.3	28.3
ImageNet weights - Stack-of-Diff.	70.1	40.8

Self-supervised learning to prime supervised learning

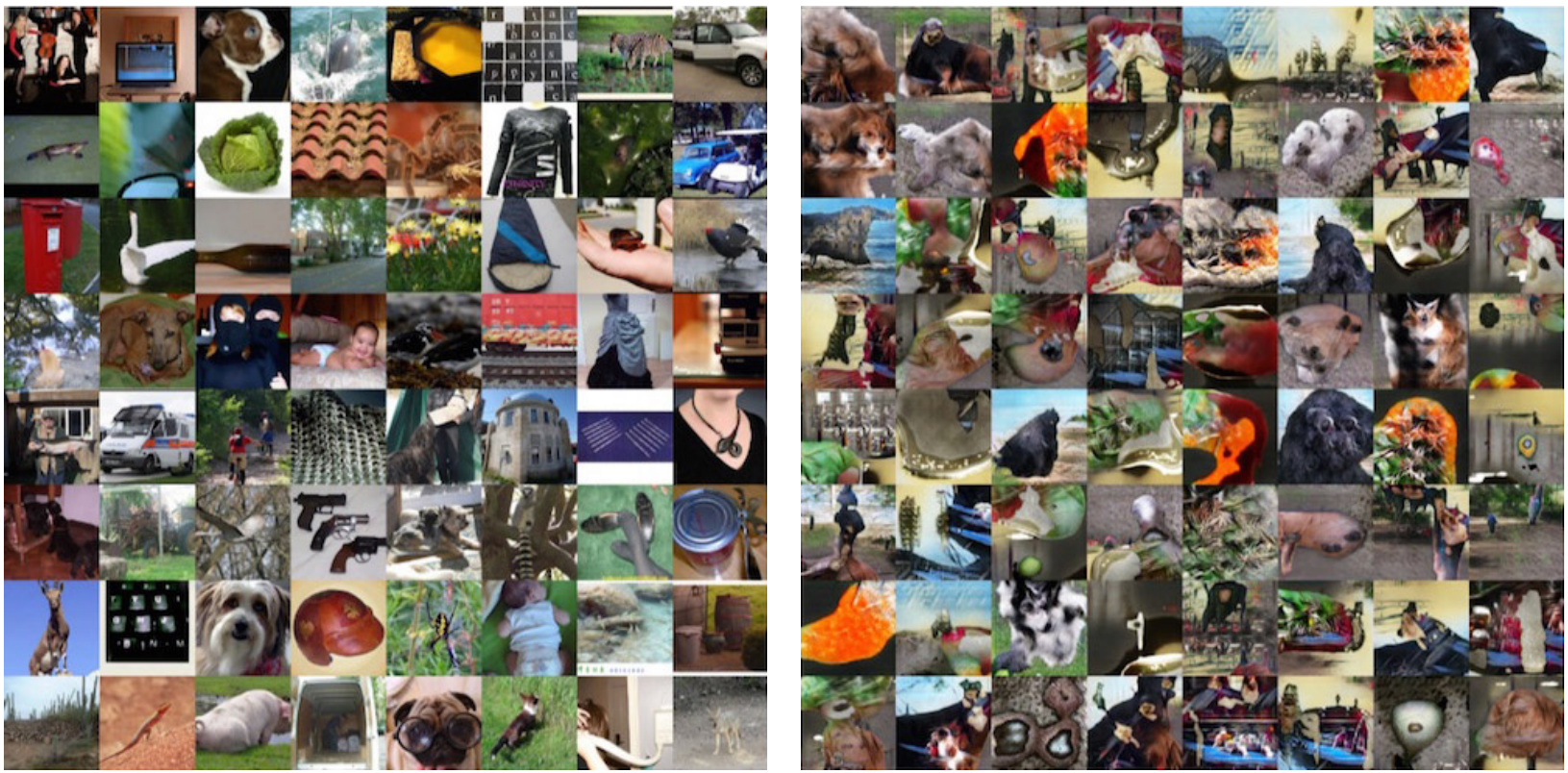
- Supervised pre-training of network on proxy-task
- Fine-tune on final task with limited training data
- Example: features for action recognition [Fernando et al., 2017]

Method	UCF101-split1	HMDB51-split1
DrLim [17]	45.7	16.3
TempCoh [32]	45.4	15.9
Obj. Patch [44]	40.7	15.6
Seq. Ver. [31]	50.9	19.8
Our - Stack-of-Diff.	**60.3**	**32.5**
Rand weights - Stack-of-Diff.	51.3	28.3
ImageNet weights - Stack-of-Diff.	70.1	40.8

- Unsupervised representation learning
- Does not allow to sample data from model

Generative models

- Unconditional density model $p_\theta(\mathbf{x})$
- Parameters estimated from unlabeled data
- Possible to draw samples from model



Samples from ImageNet dataset (left) and GAN model (right), figure from OpenAI

My first generative model

- Gaussian mixture model

$$p(z = k) = \pi_k \tag{3}$$

$$p(\mathbf{x}|z = k) = \mathcal{N}(x; \mu_k, \sigma I_D) \tag{4}$$

$$p(\mathbf{x}) = \sum_z p(z)p(\mathbf{x}|z) \tag{5}$$

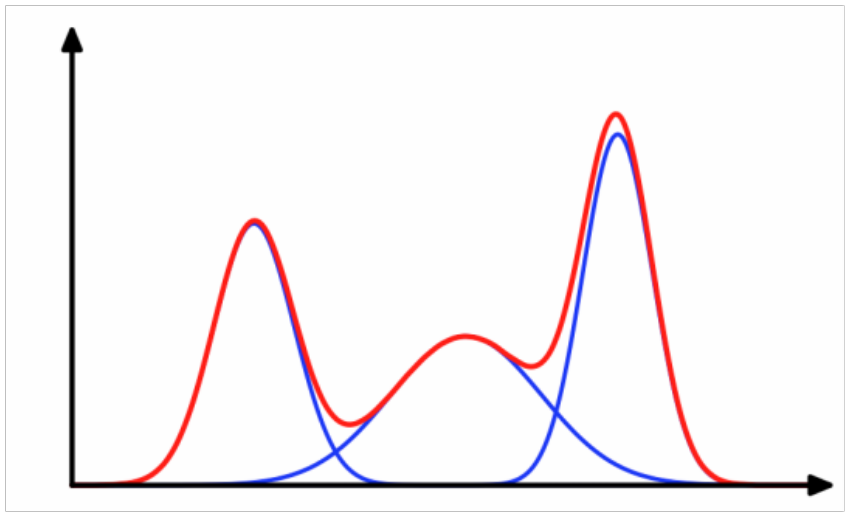


Figure from [Bishop, 2006]

My first generative model

- Gaussian mixture model

$$p(z = k) = \pi_k \tag{3}$$

$$p(\mathbf{x}|z = k) = \mathcal{N}(x; \mu_k, \sigma I_D) \tag{4}$$

$$p(\mathbf{x}) = \sum_z p(z)p(\mathbf{x}|z) \tag{5}$$

- Estimation: Expectation-Maximization (EM) algorithm

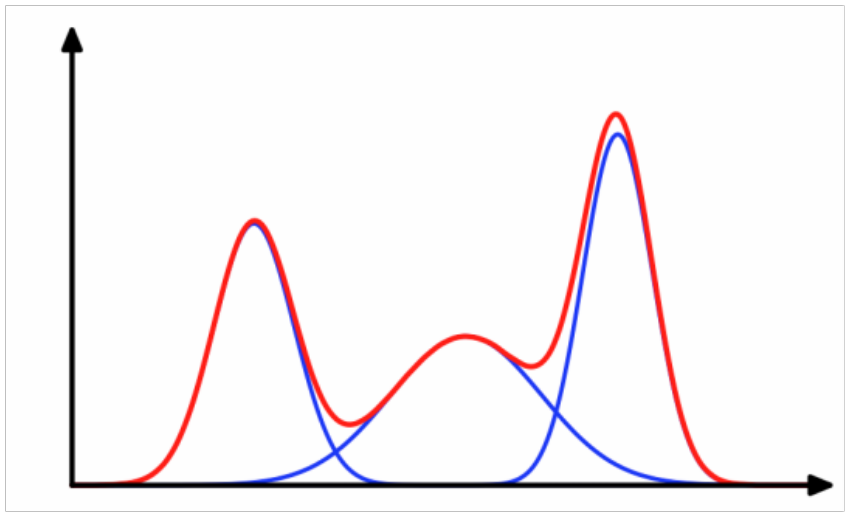


Figure from [Bishop, 2006]

My first generative model

- Gaussian mixture model

$$p(z = k) = \pi_k \tag{3}$$
$$p(\mathbf{x}|z = k) = \mathcal{N}(x; \mu_k, \sigma I_D) \tag{4}$$
$$p(\mathbf{x}) = \sum_z p(z)p(\mathbf{x}|z) \tag{5}$$

- Estimation: Expectation-Maximization (EM) algorithm
- Sampling: pick component from prior distribution $p(z)$, then draw sample from conditional distribution $p(\mathbf{x}|z)$

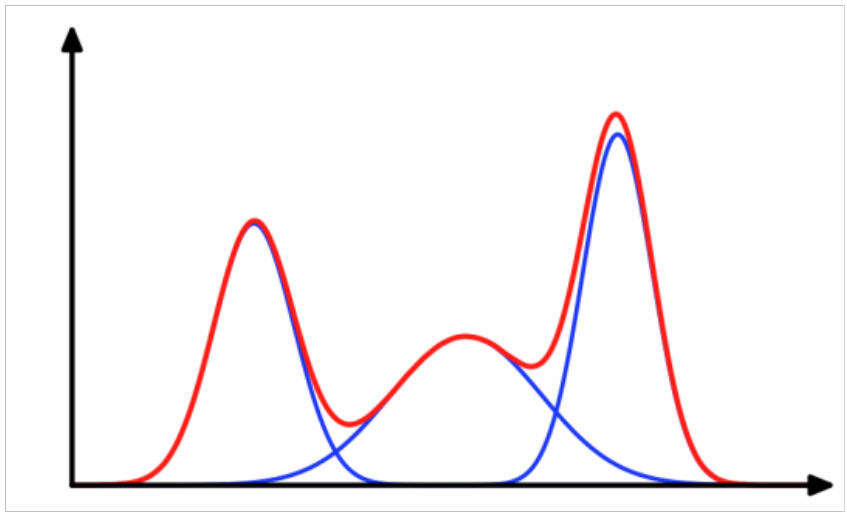


Figure from [Bishop, 2006]

My second generative model

- Probabilistic Principal Component Analysis
  [Roweis, 1997, Tipping and Bishop, 1999]

$$p(z) = \mathcal{N}(z; 0, I_d) \tag{6}$$

$$p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}; \mu + Wz, \sigma I_D) \tag{7}$$

$$p(\mathbf{x}) = \int_z p(z)p(\mathbf{x}|z) \tag{8}$$

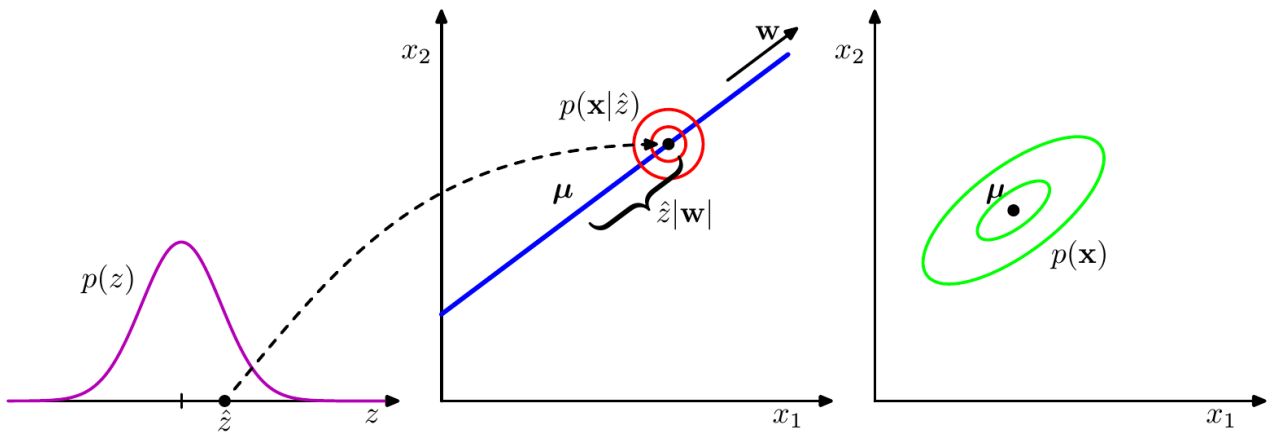


Figure from [Bishop, 2006]

My second generative model

- Probabilistic Principal Component Analysis
  [Roweis, 1997, Tipping and Bishop, 1999]

$$p(z) = \mathcal{N}(z; 0, I_d) \tag{6}$$

$$p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}; \mu + Wz, \sigma I_D) \tag{7}$$

$$p(\mathbf{x}) = \int_z p(z)p(\mathbf{x}|z) \tag{8}$$

- Estimation: SVD or EM algorithm

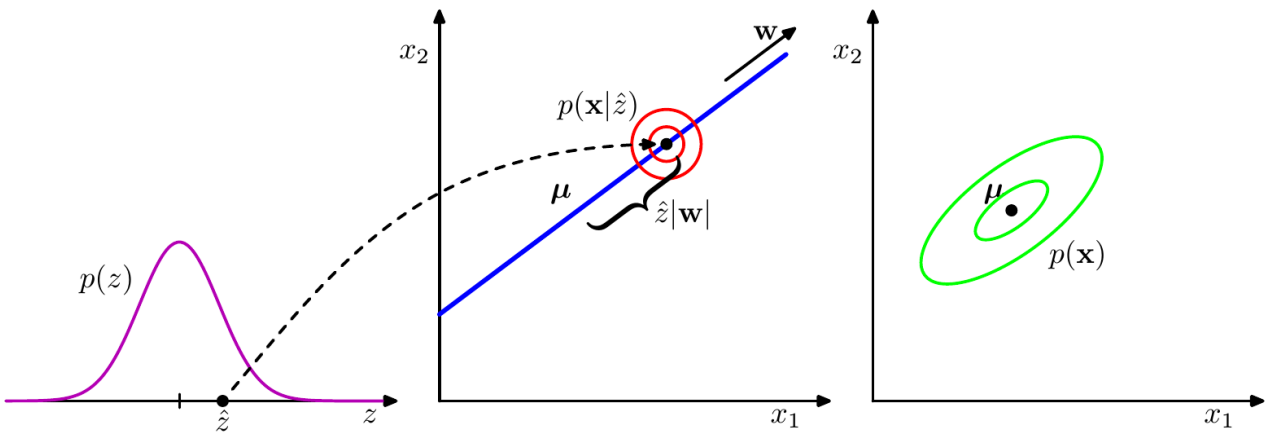


Figure from [Bishop, 2006]

My second generative model

- Probabilistic Principal Component Analysis
  [Roweis, 1997, Tipping and Bishop, 1999]

$$p(z) = \mathcal{N}(z; 0, I_d) \tag{6}$$

$$p(\mathbf{x}|z) = \mathcal{N}(\mathbf{x}; \mu + Wz, \sigma I_D) \tag{7}$$

$$p(\mathbf{x}) = \int_z p(z)p(\mathbf{x}|z) \tag{8}$$

- Estimation: SVD or EM algorithm

- Sampling: pick point in subspace from prior $p(z)$,
  then draw sample from conditional distribution $p(\mathbf{x}|z)$

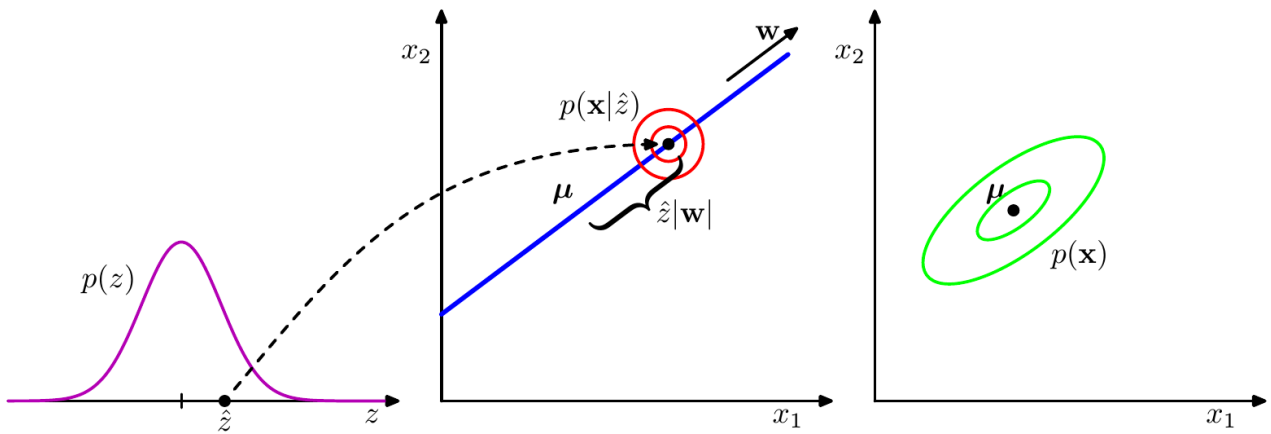
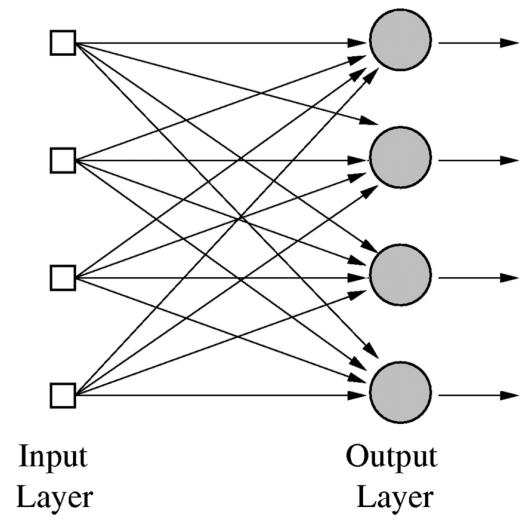


Figure from [Bishop, 2006]

Linear latent variable models

- **Linear transformation** of latent variable
  - PCA: z from unit Gaussian
  - GMM: z random 1-hot vector

$$\hat{\mathbf{x}} = Wz + \mu \tag{9}$$



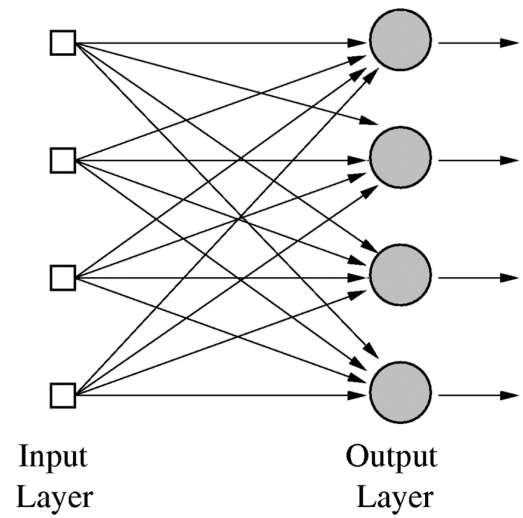
Linear latent variable models

- **Linear transformation** of latent variable
  - PCA: z from unit Gaussian
  - GMM: z random 1-hot vector

$$\hat{\mathbf{x}} = Wz + \mu \tag{9}$$

- Gaussian noise makes support non-degenerate in data space

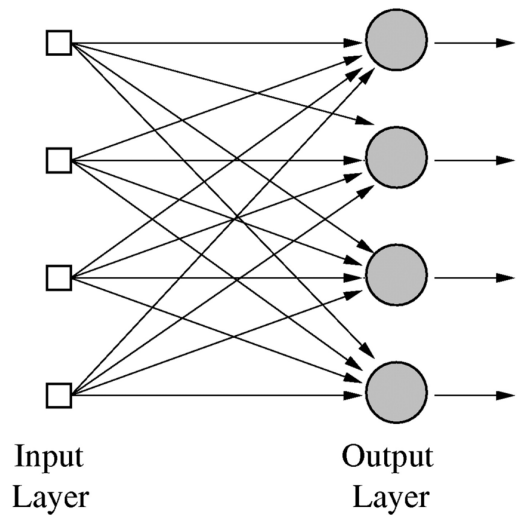
$$p(\mathbf{x}|\hat{\mathbf{x}}) = \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \sigma I_D) \tag{10}$$



Linear latent variable models

- **Linear transformation** of latent variable
  - PCA: z from unit Gaussian
  - GMM: z random 1-hot vector

$$\hat{\mathbf{x}} = Wz + \mu \tag{9}$$



- Gaussian noise makes support non-degenerate in data space

$$p(\mathbf{x}|\hat{\mathbf{x}}) = \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \sigma I_D) \tag{10}$$

- Negative log-likelihood gives ℓ_2 "reconstruction" loss of PCA and k-means

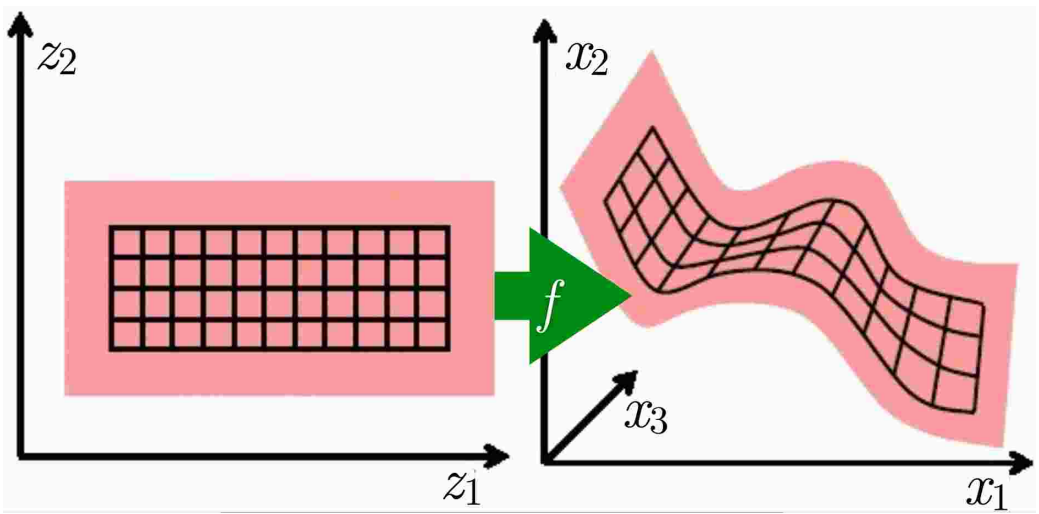
$$-\ln p(\mathbf{x}|\hat{\mathbf{x}}) = ||\mathbf{x} - \hat{\mathbf{x}}||_2^2 \tag{11}$$

Non-linear latent variable models

- Simple distribution $p(\mathbf{z})$ on latent variable $\mathbf{z}$, *e.g.* standard Gaussian

Non-linear latent variable models

- Simple distribution $p(\mathbf{z})$ on latent variable $\mathbf{z}$, *e.g.* standard Gaussian
- Non-linear function $\mathbf{x} = f_\theta(\mathbf{z})$ maps latent variable to data space, for example deep neural net



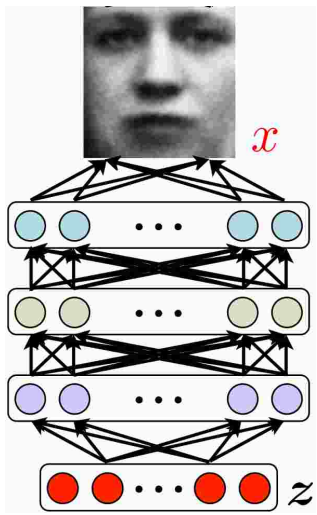
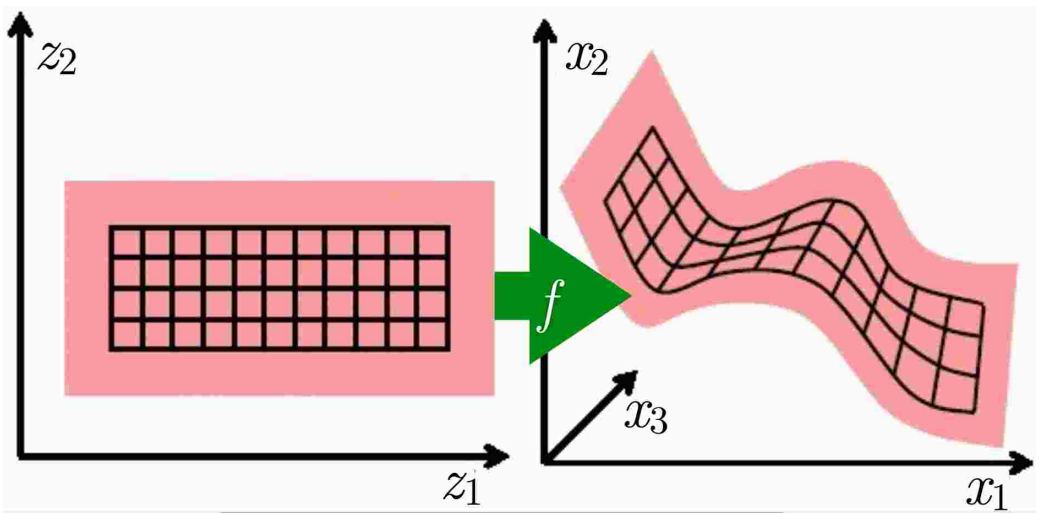


Figure from Aaron Courville

Non-linear latent variable models

- Simple distribution $p(\mathbf{z})$ on latent variable $\mathbf{z}$, *e.g.* standard Gaussian
- Non-linear function $\mathbf{x} = f_\theta(\mathbf{z})$ maps latent variable to data space, for example deep neural net
- Induces complex marginal distribution $p_\theta(\mathbf{x})$



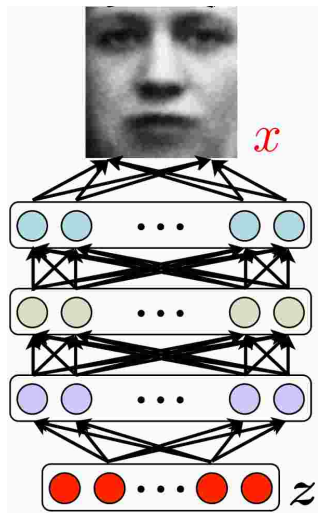


Figure from Aaron Courville

Learning deep latent variable models

- Marginal distribution on $\mathbf{x}$ obtained by integrating out $\mathbf{z}$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{12}$$

$$p_\theta(\mathbf{x}) = \int_z p(\mathbf{z}) p(\mathbf{x}|f_\theta(\mathbf{z})). \tag{13}$$

Learning deep latent variable models

- Marginal distribution on $\mathbf{x}$ obtained by integrating out $\mathbf{z}$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{12}$$

$$p_\theta(\mathbf{x}) = \int_z p(\mathbf{z}) p(\mathbf{x}|f_\theta(\mathbf{z})). \tag{13}$$

- Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$

Learning deep latent variable models

- Marginal distribution on $\mathbf{x}$ obtained by integrating out $\mathbf{z}$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{12}$$

$$p_\theta(\mathbf{x}) = \int_z p(\mathbf{z}) p(\mathbf{x}|f_\theta(\mathbf{z})). \tag{13}$$

- Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$

- Two approaches to learn deep latent variable models

Learning deep latent variable models

- Marginal distribution on $\mathbf{x}$ obtained by integrating out $\mathbf{z}$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{12}$$

$$p_\theta(\mathbf{x}) = \int_z p(\mathbf{z}) p(\mathbf{x}|f_\theta(\mathbf{z})). \tag{13}$$

- Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$

- Two approaches to learn deep latent variable models

1. Approximate integral: Variational autoencoders (VAE)

Learning deep latent variable models

- Marginal distribution on $\mathbf{x}$ obtained by integrating out $\mathbf{z}$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{12}$$

$$p_\theta(\mathbf{x}) = \int_z p(\mathbf{z}) p(\mathbf{x}|f_\theta(\mathbf{z})). \tag{13}$$

- Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$

- Two approaches to learn deep latent variable models

1. Approximate integral: Variational autoencoders (VAE)
2. Avoid integral: Generative adversarial networks (GAN)

Part I

Generative adversarial networks

Generative adversarial networks [Goodfellow et al., 2014]

- Sample $p(\mathbf{z})$, map it using deep net to $\mathbf{x} = G_\theta(\mathbf{z})$

Generative adversarial networks [Goodfellow et al., 2014]

- Sample $p(\mathbf{z})$, map it using deep net to $\mathbf{x} = G_\theta(\mathbf{z})$
- Instead of evaluating $p_\theta(\mathbf{x})$, use classifier D_ϕ
  - $D_\phi(\mathbf{x}) \in [0, 1]$ probability $\mathbf{x}$ is real *vs.* synth. image

Generative adversarial networks [Goodfellow et al., 2014]

- Sample $p(\mathbf{z})$, map it using deep net to $\mathbf{x} = G_\theta(\mathbf{z})$
- Instead of evaluating $p_\theta(\mathbf{x})$, use classifier D_ϕ
  - $D_\phi(\mathbf{x}) \in [0, 1]$ probability $\mathbf{x}$ is real *vs.* synth. image

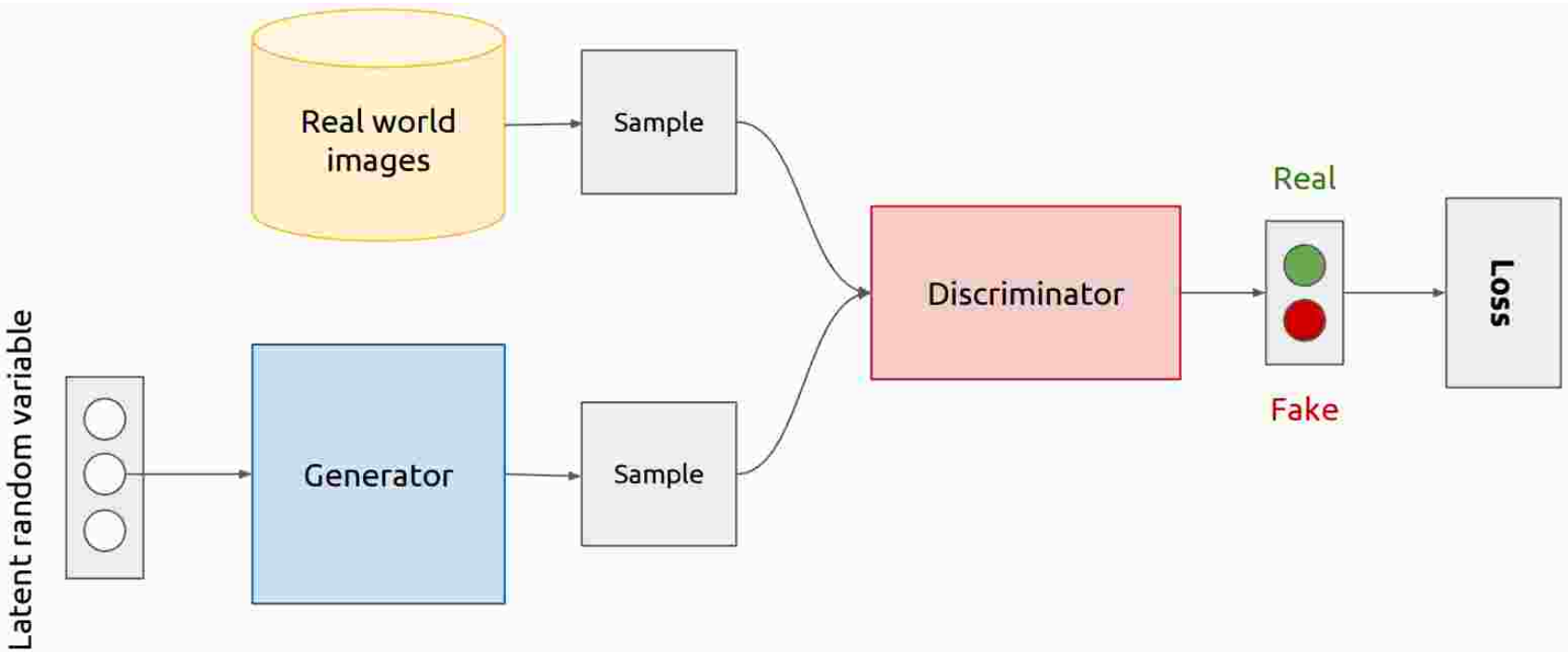


Figure from Kevin McGuinness

Discriminator architecture for images

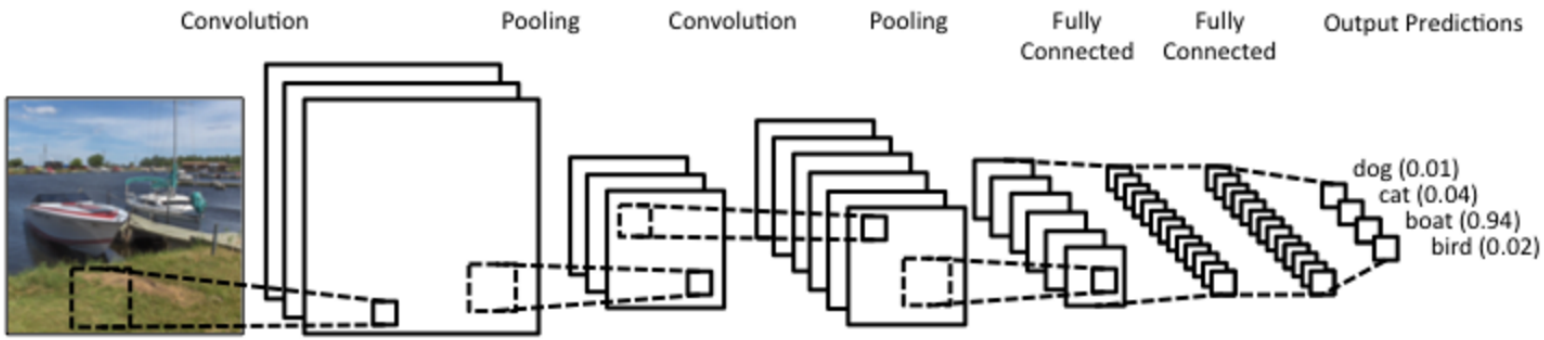


Figure from Kevin McGuinness

- Recognition CNN model, with sigmoid output layer
- Binary classification output: real / synthetic

Generator architecture for images

- Unit Gaussian prior on $\mathbf{z}$, typically 10^2 to 10^3 dimensions

Generator architecture for images

- Unit Gaussian prior on $\mathbf{z}$, typically 10^2 to 10^3 dimensions
- Up-convolutional deep network (reverse recognition CNN)
  - Replace pooling layers that reduce resolution with upsampling layers (nearest neighbor, bi-linear, or learned)

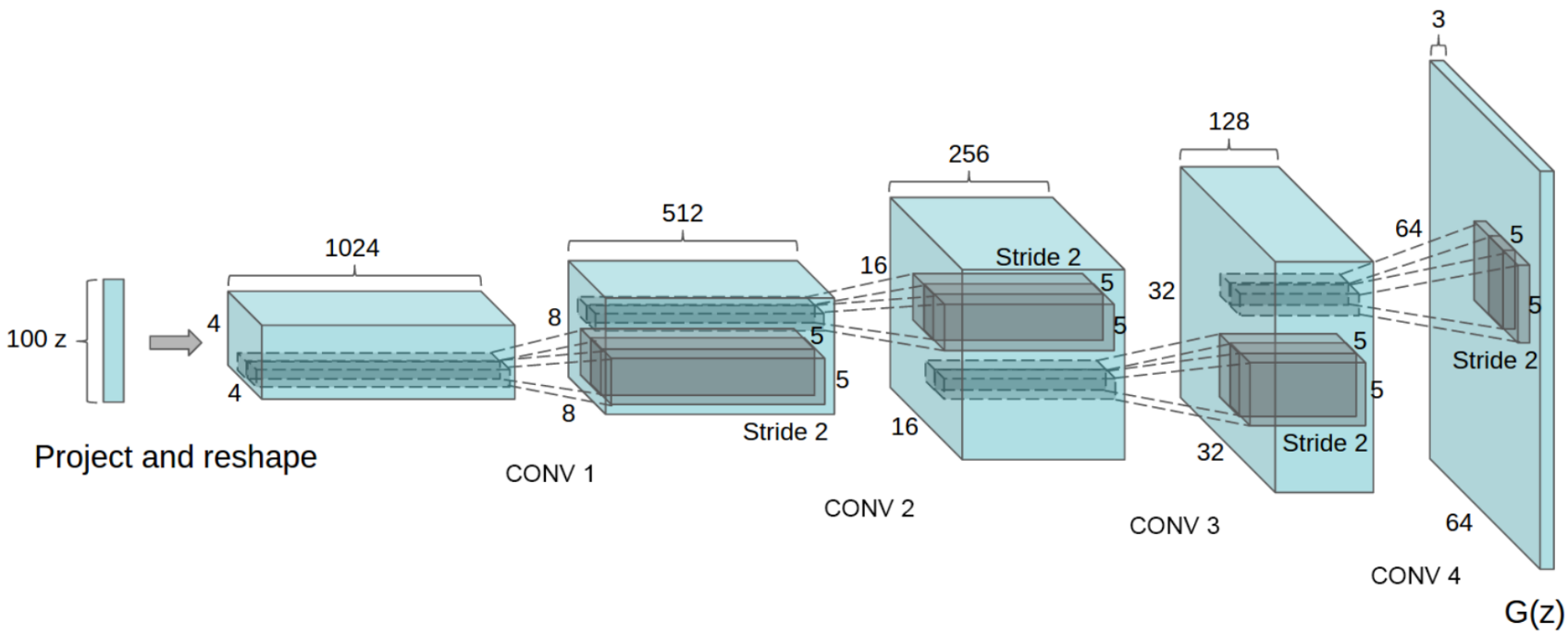


Figure from OpenAI

Generator architecture for images

- Unit Gaussian prior on $\mathbf{z}$, typically 10^2 to 10^3 dimensions
- Up-convolutional deep network (reverse recognition CNN)
  - Replace pooling layers that reduce resolution with upsampling layers (nearest neighbor, bi-linear, or learned)
  - Low-resolution layers induce long-range correlations
  - High-resolution layers induce short-range correlations

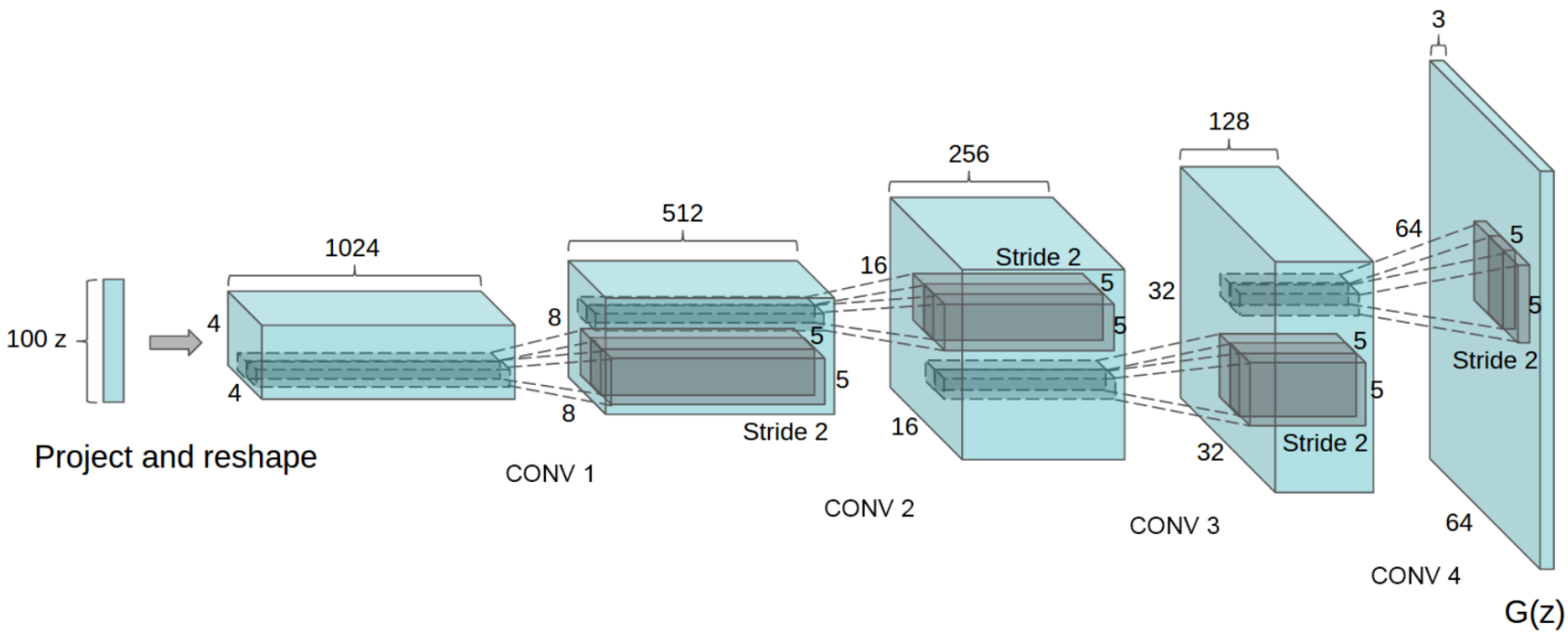
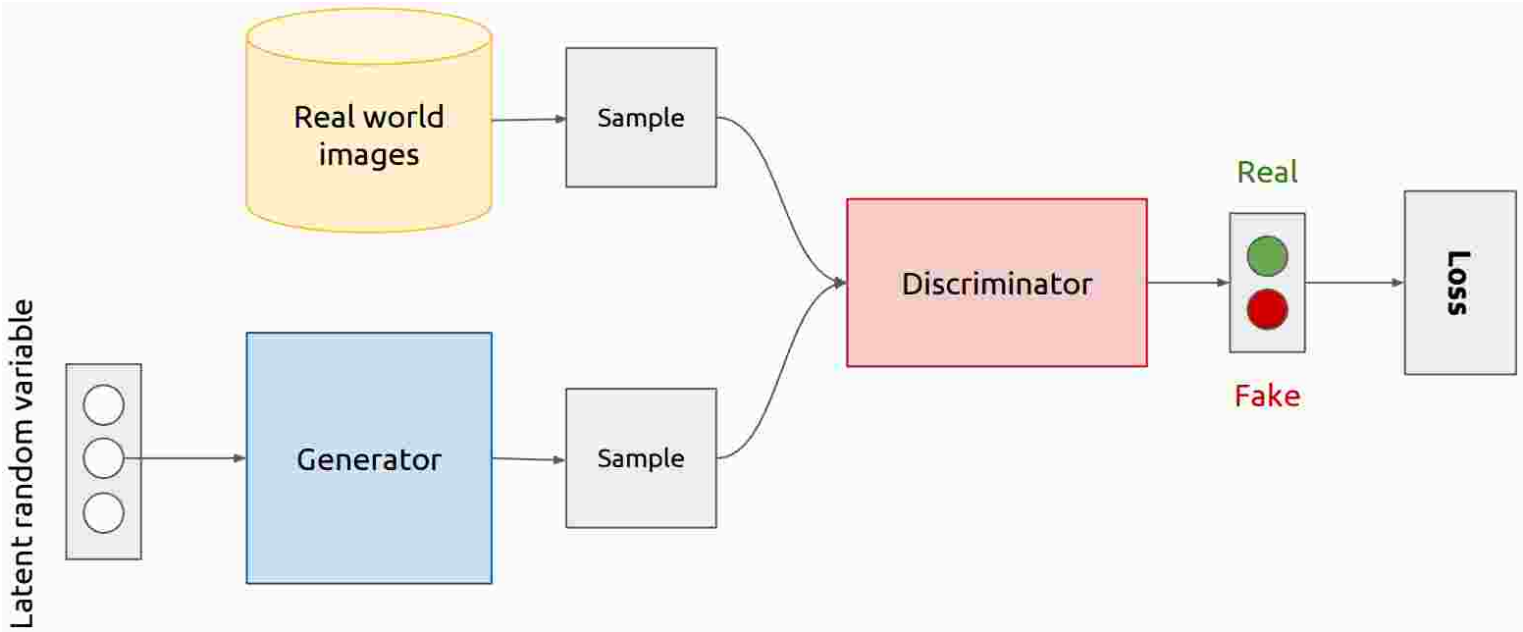


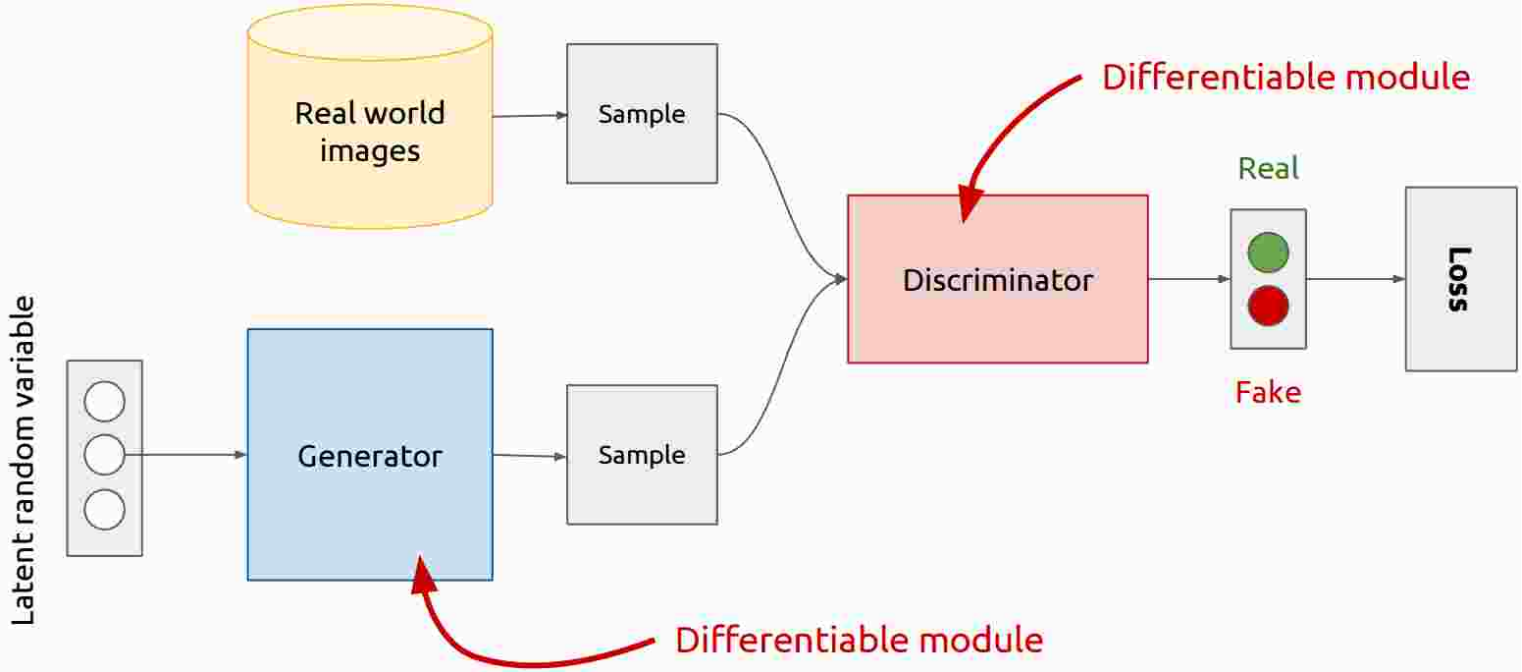
Figure from OpenAI

Training GANs



- Discriminator: maximize classification for a given generator
- Generator: degrade classification of a given discriminator

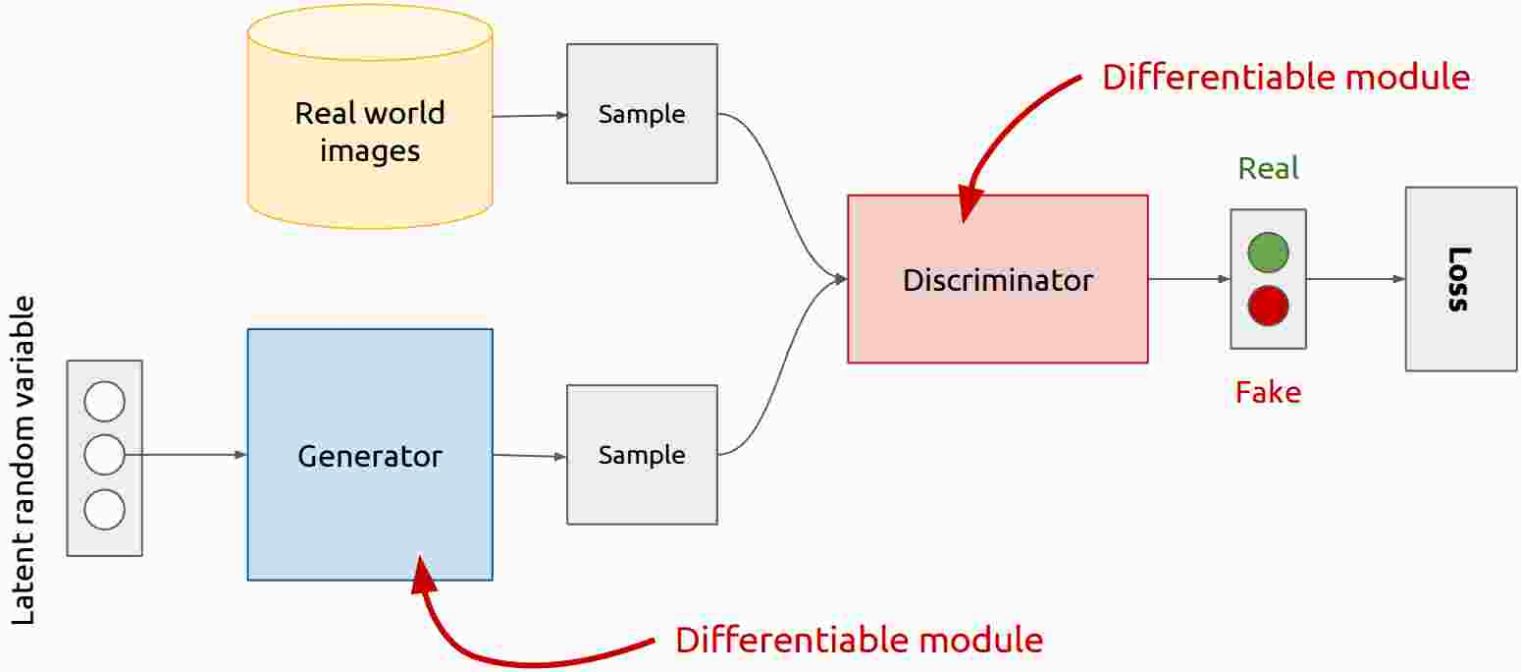
Training GANs



- Discriminator: maximize classification for a given generator
- Generator: degrade classification of a given discriminator

- Samples $\mathbf{z}$ pass through two differentiable modules

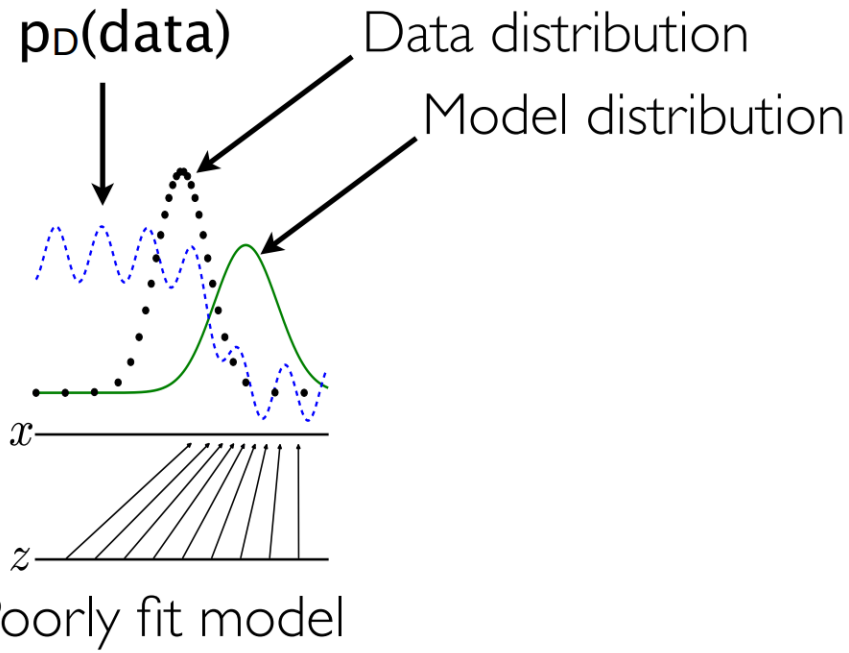
Training GANs



- Discriminator: maximize classification for a given generator
- Generator: degrade classification of a given discriminator

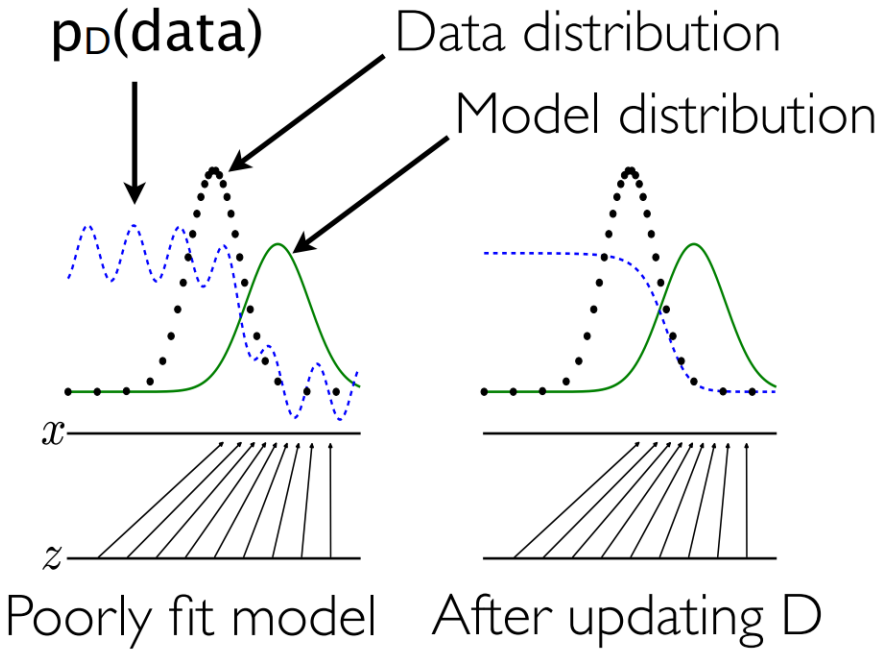
- Samples $\mathbf{z}$ pass through two differentiable modules
- Discriminator acts as **trainable loss function**

GAN learning process

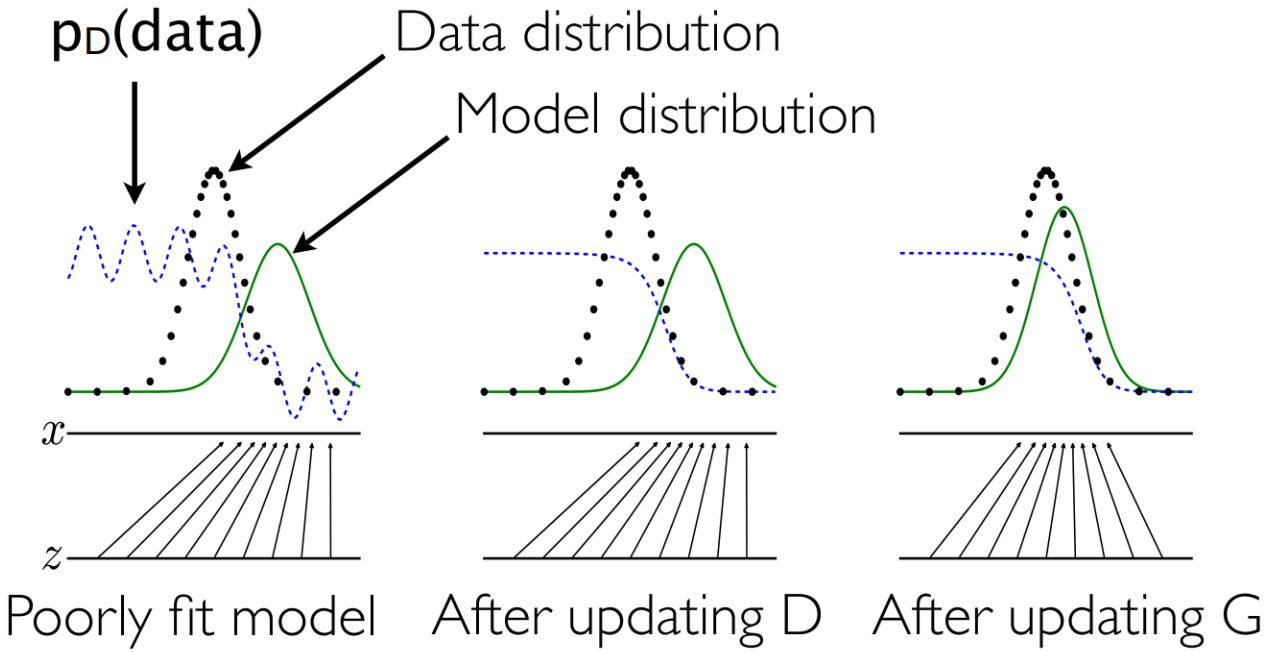


Poorly fit model

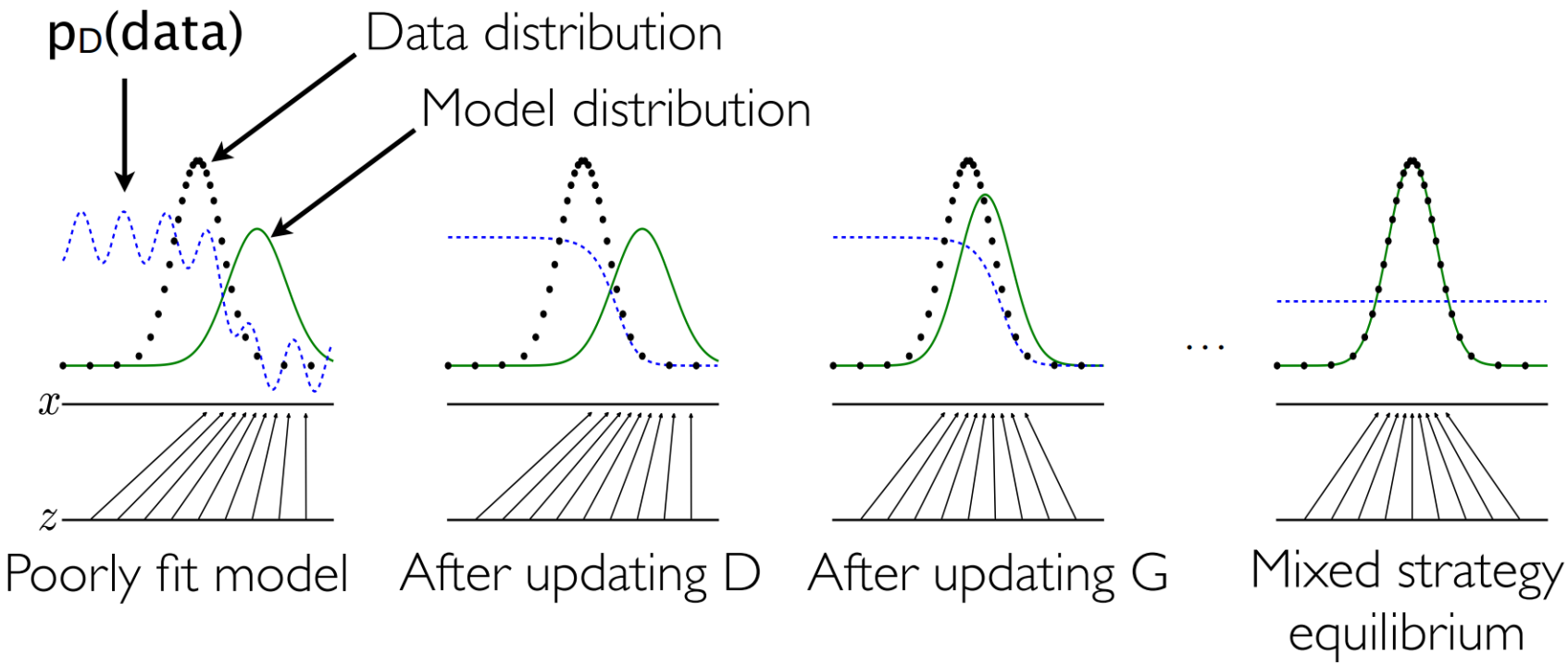
GAN learning process



GAN learning process



GAN learning process



GAN Optimization problem

- Objective function $V(\boldsymbol{\phi}, \boldsymbol{\theta})$: performance of discriminator

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_{\boldsymbol{\phi}}(x)] + \mathbb{E}_{x \sim p(\mathbf{z})}[\ln (1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})))]$$

GAN Optimization problem

- Objective function $V(\boldsymbol{\phi}, \boldsymbol{\theta})$: performance of discriminator

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_{\boldsymbol{\phi}}(x)] + \mathbb{E}_{x \sim p(\mathbf{z})}[\ln (1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})))]$$

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} V(\boldsymbol{\phi}, \boldsymbol{\theta})$$

GAN Optimization problem

- Objective function $V(\boldsymbol{\phi}, \boldsymbol{\theta})$: performance of discriminator

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_{\boldsymbol{\phi}}(x)] + \mathbb{E}_{x \sim p(\mathbf{z})}[\ln (1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})))]$$

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} V(\boldsymbol{\phi}, \boldsymbol{\theta})$$

- Assuming infinite data and model capacity,
and reaching optimal discriminator at each iteration

GAN Optimization problem

- Objective function $V(\boldsymbol{\phi}, \boldsymbol{\theta})$: performance of discriminator

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_{\boldsymbol{\phi}}(x)] + \mathbb{E}_{x \sim p(\mathbf{z})}[\ln (1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})))]$$

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} V(\boldsymbol{\phi}, \boldsymbol{\theta})$$

- Assuming infinite data and model capacity,
  and reaching optimal discriminator at each iteration
  1. Unique global optimum for G at data distribution

GAN Optimization problem

- Objective function $V(\boldsymbol{\phi}, \boldsymbol{\theta})$: performance of discriminator

$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_{\boldsymbol{\phi}}(x)] + \mathbb{E}_{x \sim p(\mathbf{z})}[\ln (1 - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\mathbf{z})))]$$

$$\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} V(\boldsymbol{\phi}, \boldsymbol{\theta})$$

- Assuming infinite data and model capacity, and reaching optimal discriminator at each iteration
  1. Unique global optimum for G at data distribution
  2. Convergence to optimum guaranteed

Optimal discriminator

- For fixed generator G, the optimal discriminator D is the Bayes classifier

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \tag{14}$$

- Proof: Given generator f, the optimal discriminator maximizes

$$\begin{aligned} V(D,G) &= \mathbb{E}_{x\sim p_{\text{data}}(x)}[\ln D(x)] + \mathbb{E}_{z\sim p(z)}[\ln(1-D(G(z)))] \\ &= \int_x p_{\text{data}}(x)\ln D(x) + p_G(x)\ln(1-D(x))\,dx \end{aligned}$$

For any $(a,b) \in \mathbb{R}^2 \setminus \{0,0\}$ the function $a\ln(y) + b\ln(1-y)$ achieves its maximum in $[0,1]$ at $y = a/(a+b)$.
Discriminator only needs to be defined in support of training data and $p_G(x)$.

Link with Jensen-Shannon divergence

- Plugging in the optimal discriminator we obtain

$$\max_D V(D,G) = -\ln 4 + 2D_{JS}(p_{\text{data}}||p_G)$$

Link with Jensen-Shannon divergence

- Plugging in the optimal discriminator we obtain

$$\max_D V(D,G) = -\ln 4 + 2D_{JS}(p_{\text{data}}||p_G)$$

with Jensen-Shannon divergence

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\middle|\middle|\frac{p+q}{2}\right)$$

Link with Jensen-Shannon divergence

- Plugging in the optimal discriminator we obtain

$$\max_D V(D,G) = -\ln 4 + 2D_{JS}(p_{\text{data}}||p_G)$$

with Jensen-Shannon divergence

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\middle|\middle|\frac{p+q}{2}\right)$$

- Unique global minimum obtained for $p_{\text{data}} = p_G$

Link with Jensen-Shannon divergence

- Plugging in the optimal discriminator we obtain

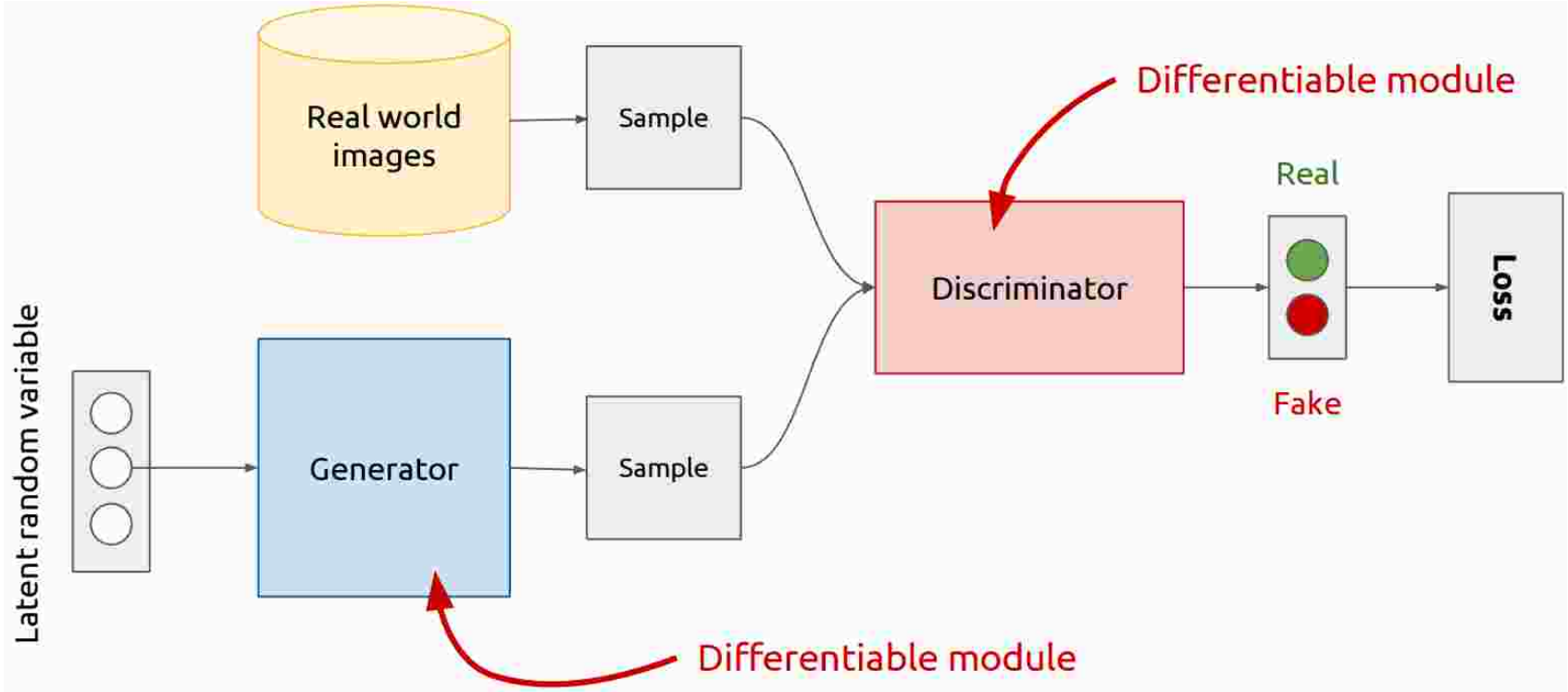
$$\max_D V(D,G) = -\ln 4 + 2D_{JS}(p_{\text{data}}||p_G)$$

with Jensen-Shannon divergence

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\middle|\middle|\frac{p+q}{2}\right)$$

- Unique global minimum obtained for $p_{\text{data}} = p_G$
- If D is set to optimum at each iteration, then convexity shows that gradient descent on p_G recovers the global optimum

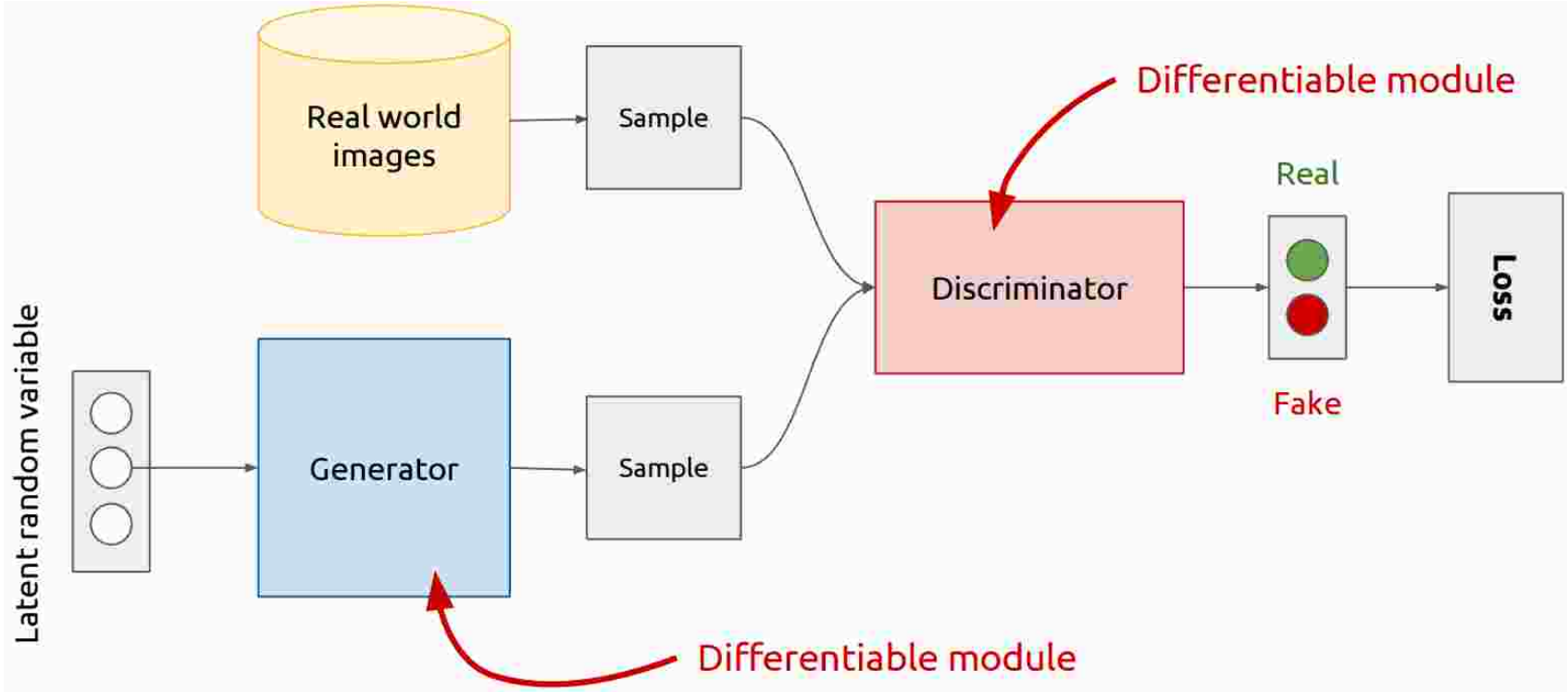
Training GANs in practice



$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

- Replace expectations with sample average in mini-batch

Training GANs in practice



$$V(\phi, \theta) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\ln D_\phi(x)] + \mathbb{E}_{z \sim p(z)}[\ln(1 - D_\phi(f_\theta(z)))]$$

- Replace expectations with sample average in mini-batch
- Parallel stochastic gradient descent on ϕ and θ

Samples model learned on face images [Radford et al., 2016]

GAN generalizes beyond training data

- Sample along linear trajectory in latent space $z_1 \rightarrow z_2$
- Smooth transitions suggest generalization,
  sharp transitions would suggest literal memorization

GAN generalizes beyond training data

- Sample along linear trajectory in latent space $z_1 \rightarrow z_2$
- Smooth transitions suggest generalization,
  sharp transitions would suggest literal memorization

Examples taken from [Radford et al., 2016], trained on LSUN bedroom dataset

Vector arithmetic on latent variables

- Word2vec word embedding shows semantic regularities [Mikolov et al., 2013]

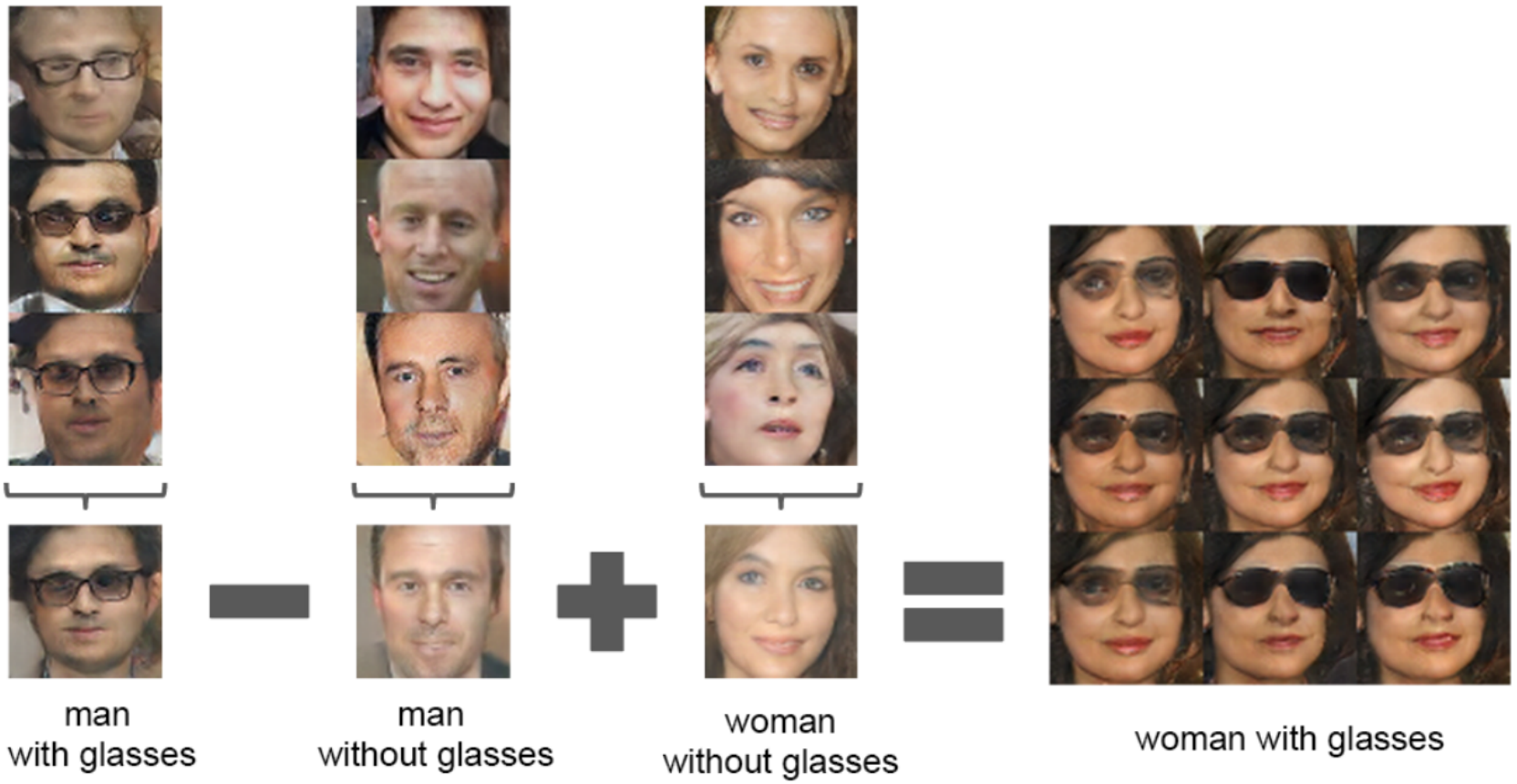
$$(\mathbf{z}_{\text{king}} - \mathbf{z}_{\text{man}}) + \mathbf{z}_{\text{woman}} \approx \mathbf{z}_{\text{queen}} \tag{15}$$

Vector arithmetic on latent variables

- Word2vec word embedding shows semantic regularities [Mikolov et al., 2013]

$$(\mathbf{z}_{\text{king}} - \mathbf{z}_{\text{man}}) + \mathbf{z}_{\text{woman}} \approx \mathbf{z}_{\text{queen}} \tag{15}$$

- Consider GAN trained on human faces, average $\mathbf{z}$ vectors over three samples for stability



Why is GAN training is difficult in practice?

Why is GAN training is difficult in practice?

- Recall divergence measures between distributions

Why is GAN training is difficult in practice?

- Recall divergence measures between distributions
- Kullback-Leibler divergence: maximum likelihood training
  - Infinite if q (model) has a zero in the support of p (data)

$$D_{KL}(p||q) = \int_x p(x)\big[\ln q(x) - \ln p(x)\big] \tag{16}$$

Why is GAN training is difficult in practice?

- Recall divergence measures between distributions
- Kullback-Leibler divergence: maximum likelihood training
  - Infinite if q (model) has a zero in the support of p (data)

$$D_{KL}(p||q) = \int_x p(x)\big[\ln q(x) - \ln p(x)\big] \tag{16}$$

- Jensen-Shannon divergence: idealized GAN training
  - Symmetric KL to mixture of p and q

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p\middle|\middle|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\middle|\middle|\frac{p+q}{2}\right) \tag{17}$$

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
   - Generator no longer minimize JS divergence

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
   - Generator no longer minimize JS divergence

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
   - Generator no longer minimize JS divergence

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
   - Optimizes $KL(p_G||p_{\text{data}}) - 2JS(p_G||p_{\text{data}})$

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
   - Generator no longer minimize JS divergence

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
   - Optimizes $KL(p_G||p_{\text{data}}) - 2JS(p_G||p_{\text{data}})$
   - Wrong sign in the JS divergence

Why is GAN training is difficult in practice?

[Arjovsky et al., 2017]

1. Strong discriminator leads to vanishing gradients of $\mathbb{E}_{p_z}[\ln(1 - D(G(z)))]$ w.r.t. generator
   - Happens early in training with poor generator
   - Tuning of capacity and training regime of discriminator
   - Generator no longer minimize JS divergence

2. Minimizing $-\mathbb{E}_{p_z}[\ln(D(G(z)))]$ instead to boost gradient
   - Optimizes $KL(p_G||p_{\text{data}}) - 2JS(p_G||p_{\text{data}})$
   - Wrong sign in the JS divergence
   - Direction of KL term leads to mode dropping

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$
- Conditional $\gamma(y|x)$ "moves mass" to transform $p(\cdot)$ into $q(\cdot)$

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$
- Conditional $\gamma(y|x)$ "moves mass" to transform $p(\cdot)$ into $q(\cdot)$
- Cost associated with a given transformation

$$T(\gamma) = \int_{x,y} \gamma(x, y) \, ||x - y|| = \int_x p(x) \int_y \gamma(y|x) \, ||x - y||$$

Wasserstein or "earth-mover" distance

- Consider joint distribution $\gamma(x, y)$
  with marginals $p(x) = \gamma(x)$ and $q(y) = \gamma(y)$
- Conditional $\gamma(y|x)$ "moves mass" to transform $p(\cdot)$ into $q(\cdot)$
- Cost associated with a given transformation

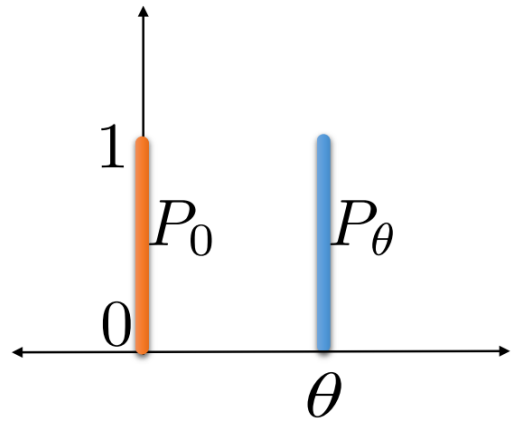
$$T(\gamma) = \int_{x,y} \gamma(x, y) \, ||x - y|| = \int_x p(x) \int_y \gamma(y|x) \, ||x - y||$$

- Wasserstein distance is the cost of optimal transformation

$$D_{WS}(p||q) = \inf_{\gamma \in \Gamma(p,q)} T(\gamma) \tag{18}$$

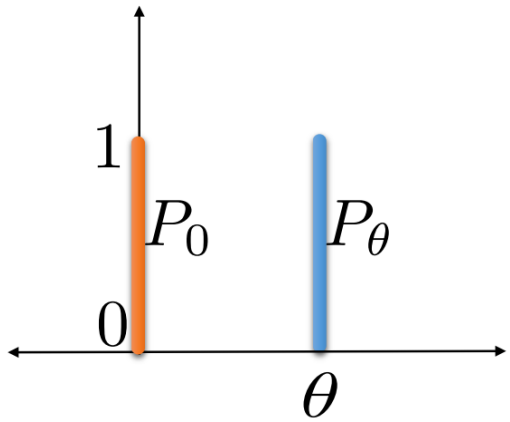
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0,1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0,1]$ for $x_1 = \theta$



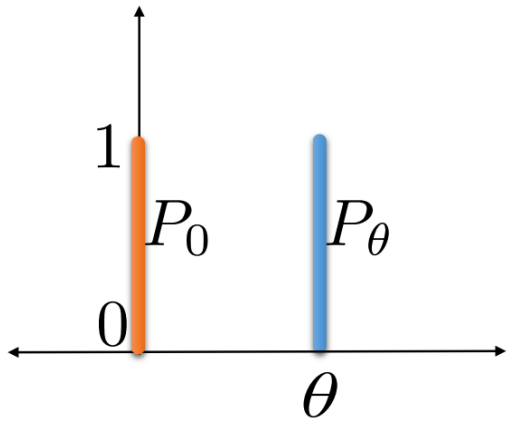
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0,1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0,1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$



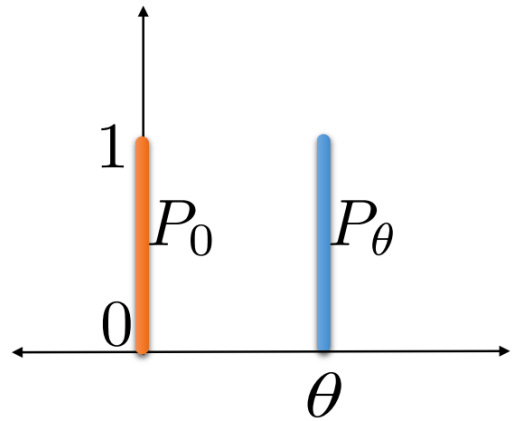
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$



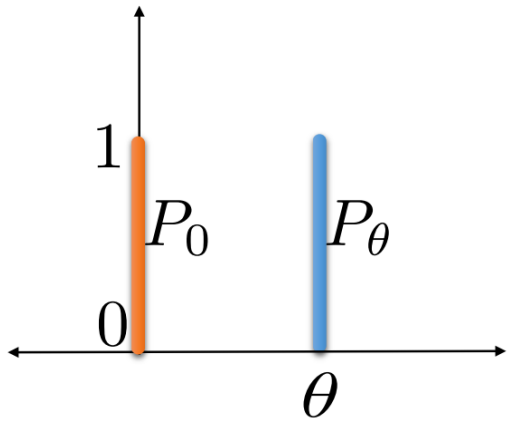
Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$



Distributions with low dimensional support

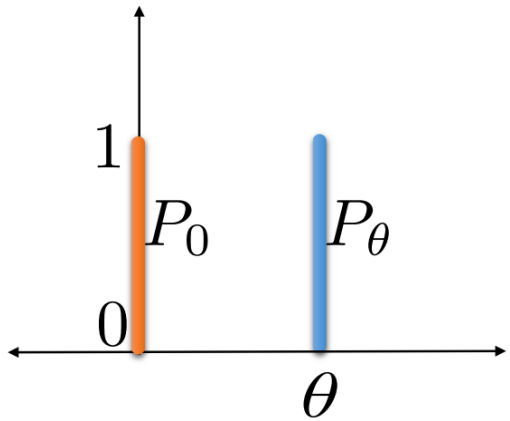
- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0,1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0,1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$



- Wasserstein based on proximity of support

Distributions with low dimensional support

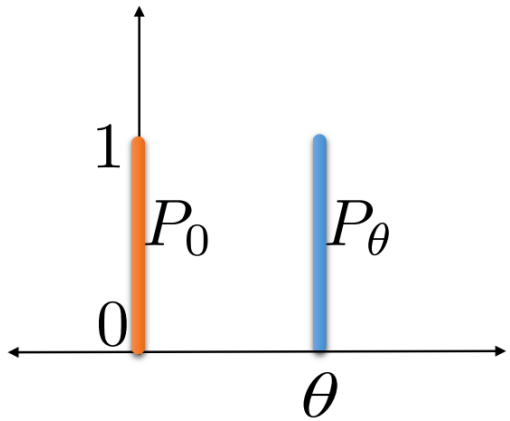
- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0,1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0,1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$



- Wasserstein based on proximity of support
- JS and KL based on overlap of support

Distributions with low dimensional support

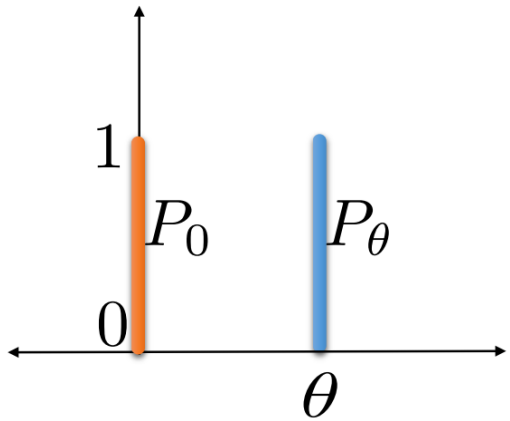
- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$



- Wasserstein based on proximity of support
- JS and KL based on overlap of support
  - In general measure zero overlap with low dim. supports

Distributions with low dimensional support

- Simple example: support on lines in $\mathbb{R}^2$
  - p_0 uniform on $x_2 \in [0, 1]$ for $x_1 = 0$
  - p_θ uniform on $x_2 \in [0, 1]$ for $x_1 = \theta$
- All measures zero for $\theta = 0$, but for $\theta \neq 0$
  - $D_{KL}(p_0||p_\theta) = \infty$
  - $D_{JS}(p_0||p_\theta) = \ln 2$
  - $D_{WS}(p_0||p_\theta) = |\theta|$



- Wasserstein based on proximity of support
- JS and KL based on overlap of support
  - In general measure zero overlap with low dim. supports
  - GAN has support with dimension of latent variable $\mathbf{z}$

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_{\text{data}}||p_G) = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$$

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_{\text{data}}||p_G) = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$$

1. Restrict D to some deep net architecture

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_{\text{data}}||p_G) = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$$

1. Restrict D to some deep net architecture
2. Enforce Lipschitz constraint by clipping discriminator weights or penalty on gradient magnitude [Gulrajani et al., 2017a]

Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_{\text{data}}||p_G) = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$$

  1. Restrict D to some deep net architecture
  2. Enforce Lipschitz constraint by clipping discriminator weights or penalty on gradient magnitude [Gulrajani et al., 2017a]

- Removes log-sigmoid transformation w.r.t. normal GAN

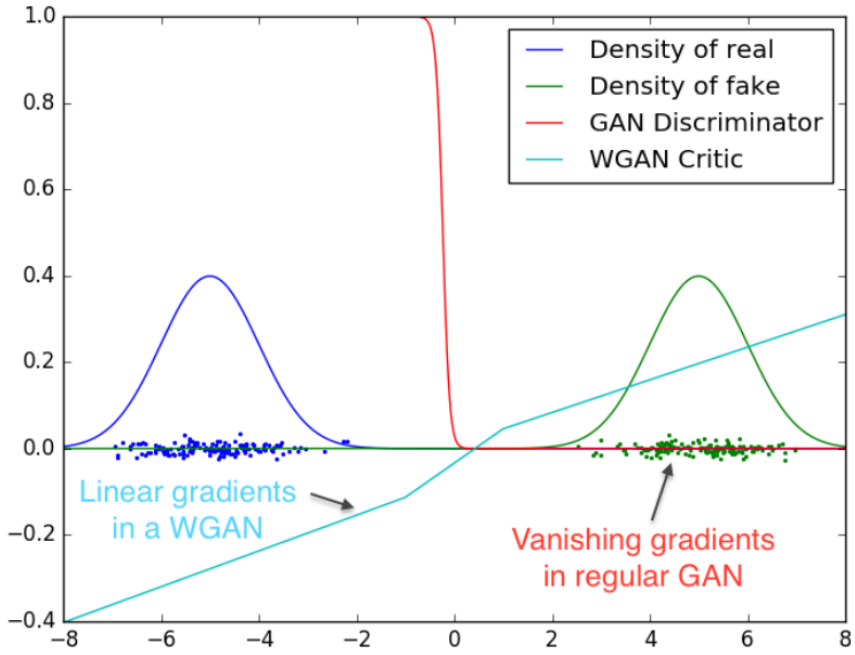
Wasserstein GAN

- Dual formulation of Wasserstein distance

$$D_{WS}(p_{\text{data}}||p_G) = \frac{1}{k} \max_{||D||_L \leq k} \mathbb{E}_{p_{\text{data}}}[D(\mathbf{x})] - \mathbb{E}_{p_z}[D(G(\mathbf{z}))]$$

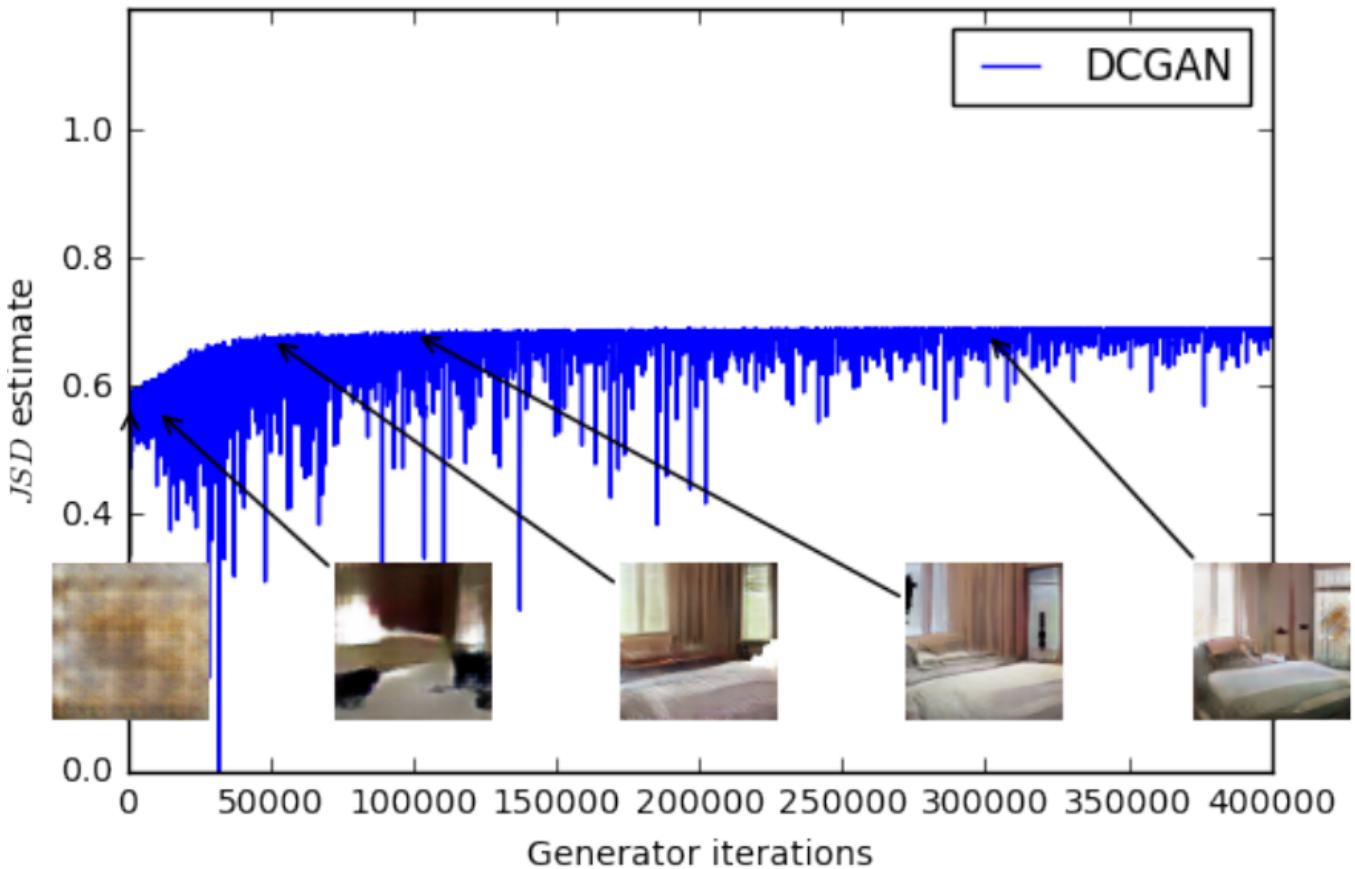
  1. Restrict D to some deep net architecture
  2. Enforce Lipschitz constraint by clipping discriminator weights or penalty on gradient magnitude [Gulrajani et al., 2017a]

- Removes log-sigmoid transformation w.r.t. normal GAN



Experimental comparison GAN and WGAN

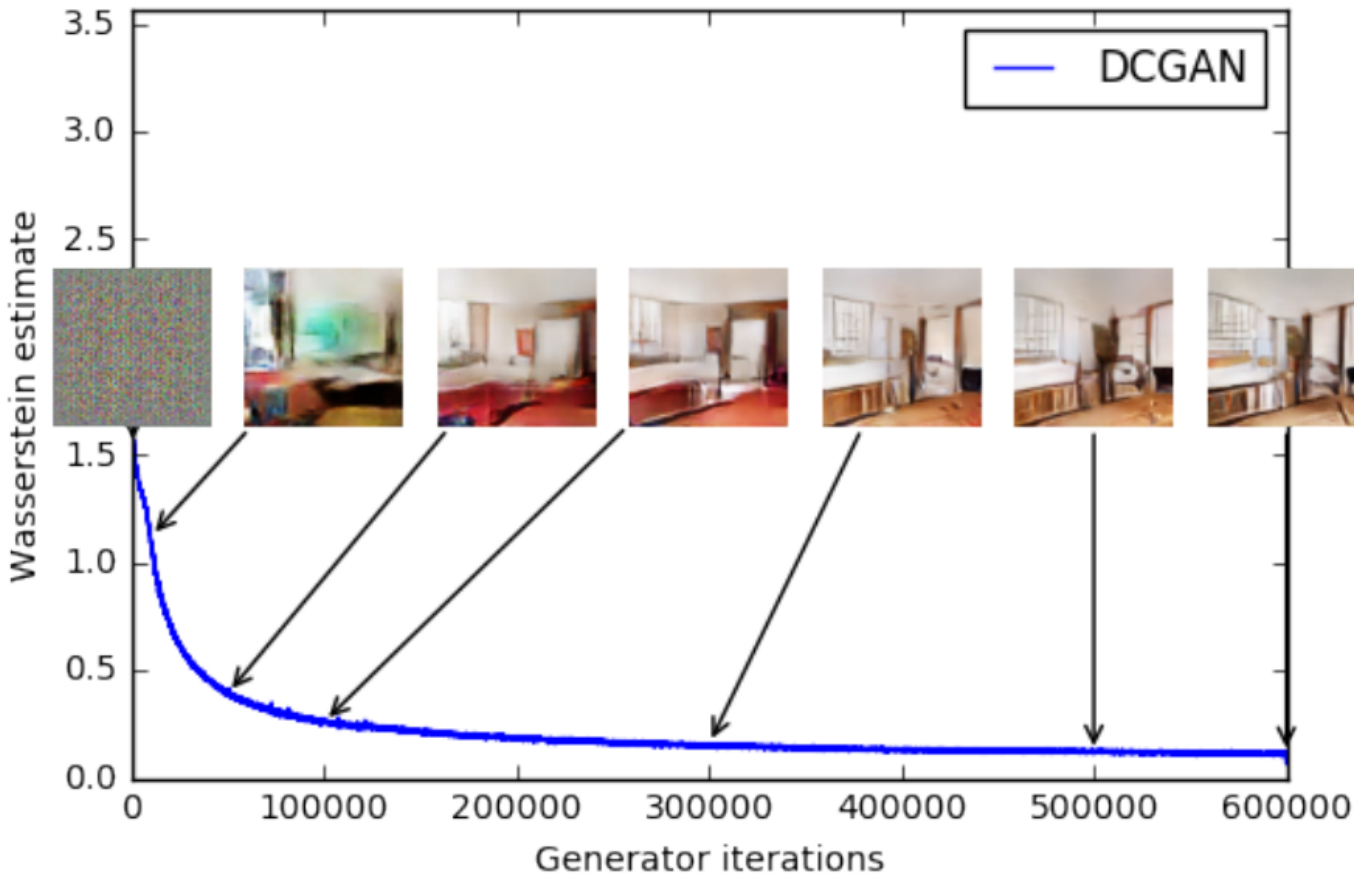
- GAN loss unstable, and actually increases over iterations!
- WGAN loss deceases in stable manner
- WGAN gives better correlation loss and sample quality



GAN

Experimental comparison GAN and WGAN

- GAN loss unstable, and actually increases over iterations!
- WGAN loss deceases in stable manner
- WGAN gives better correlation loss and sample quality



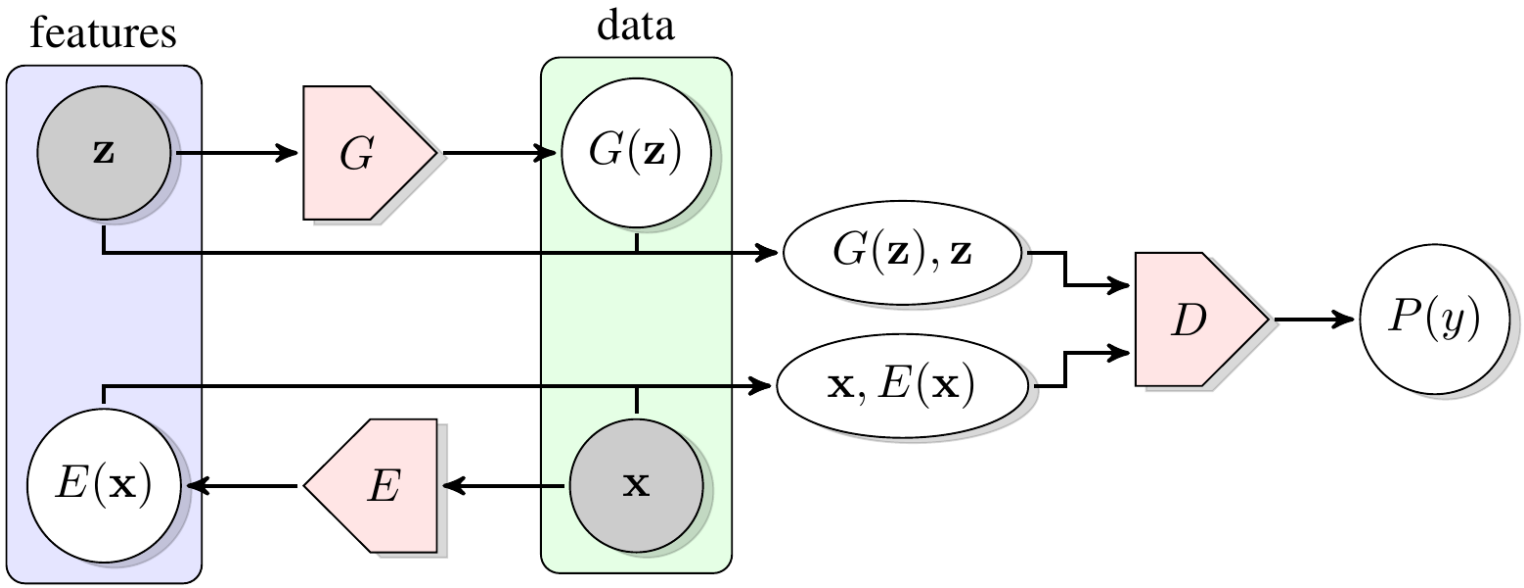
WGAN

Latent variable inference in GANs [Donahue et al., 2017]

- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$

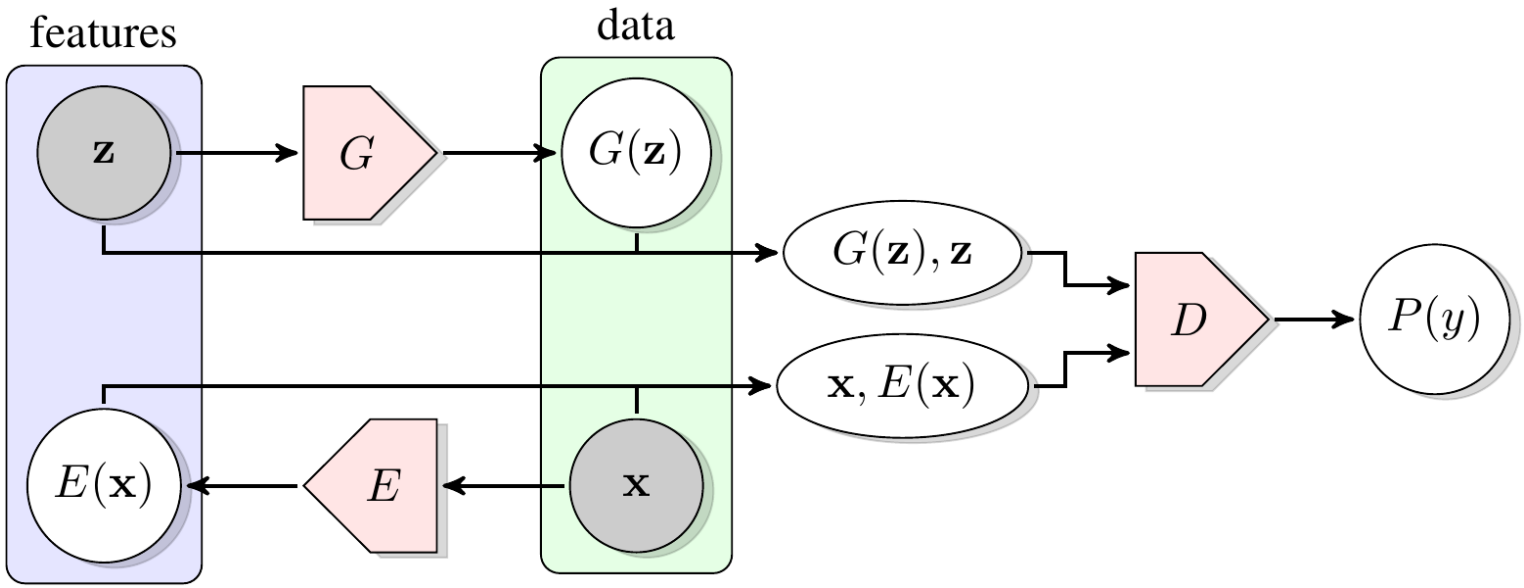
Latent variable inference in GANs [Donahue et al., 2017]

- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$



Latent variable inference in GANs [Donahue et al., 2017]

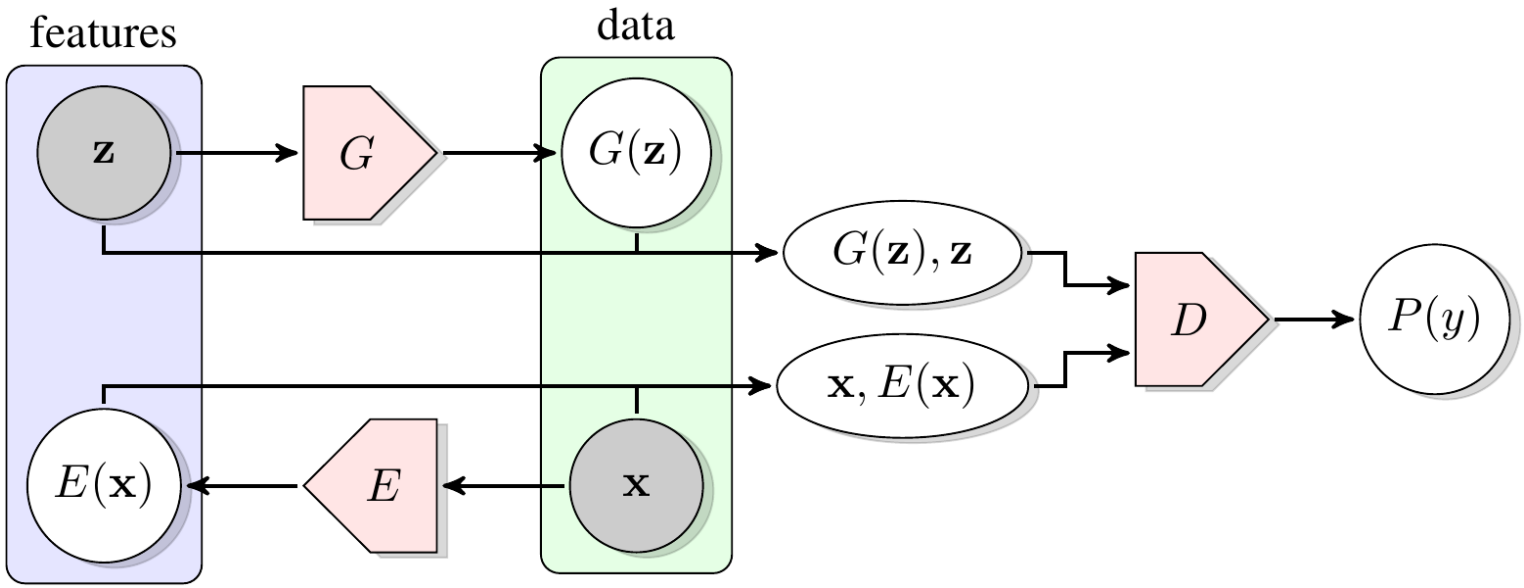
- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$



- Generator: maps latent variable $\mathbf{z}$ to data point $\mathbf{x}$

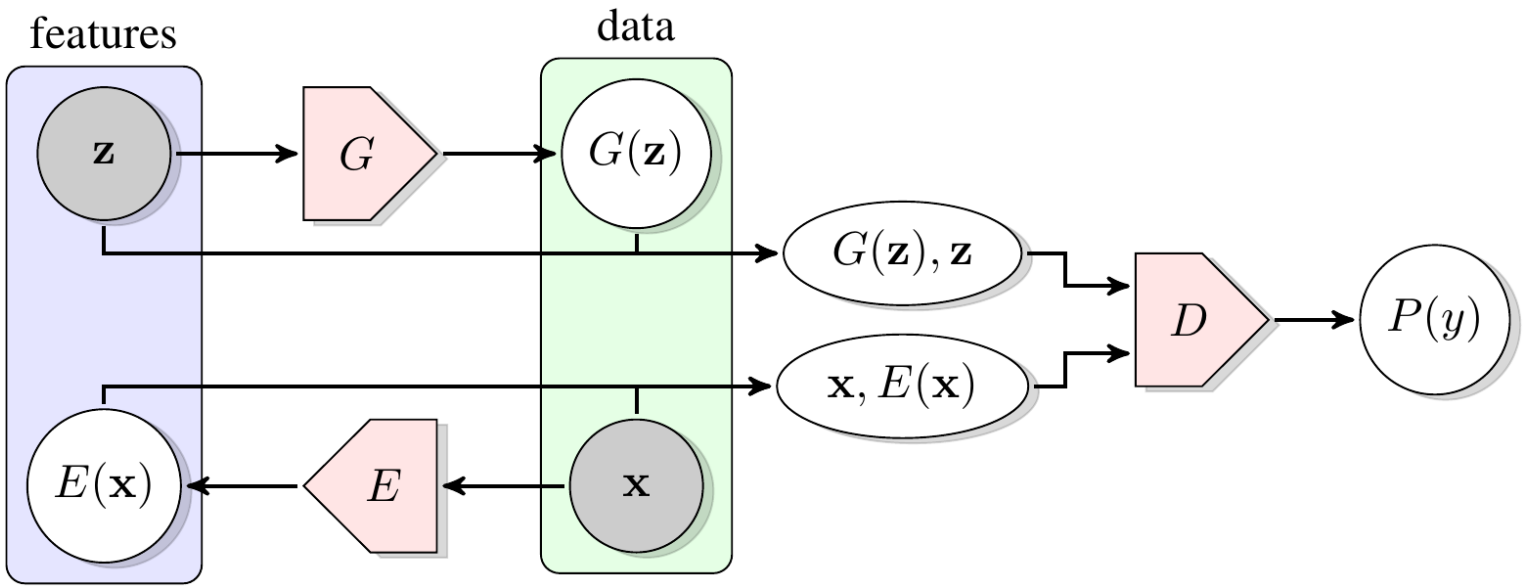
Latent variable inference in GANs [Donahue et al., 2017]

- Vanilla GAN lacks a mechanism to infer $\mathbf{z}$ from $\mathbf{x}$

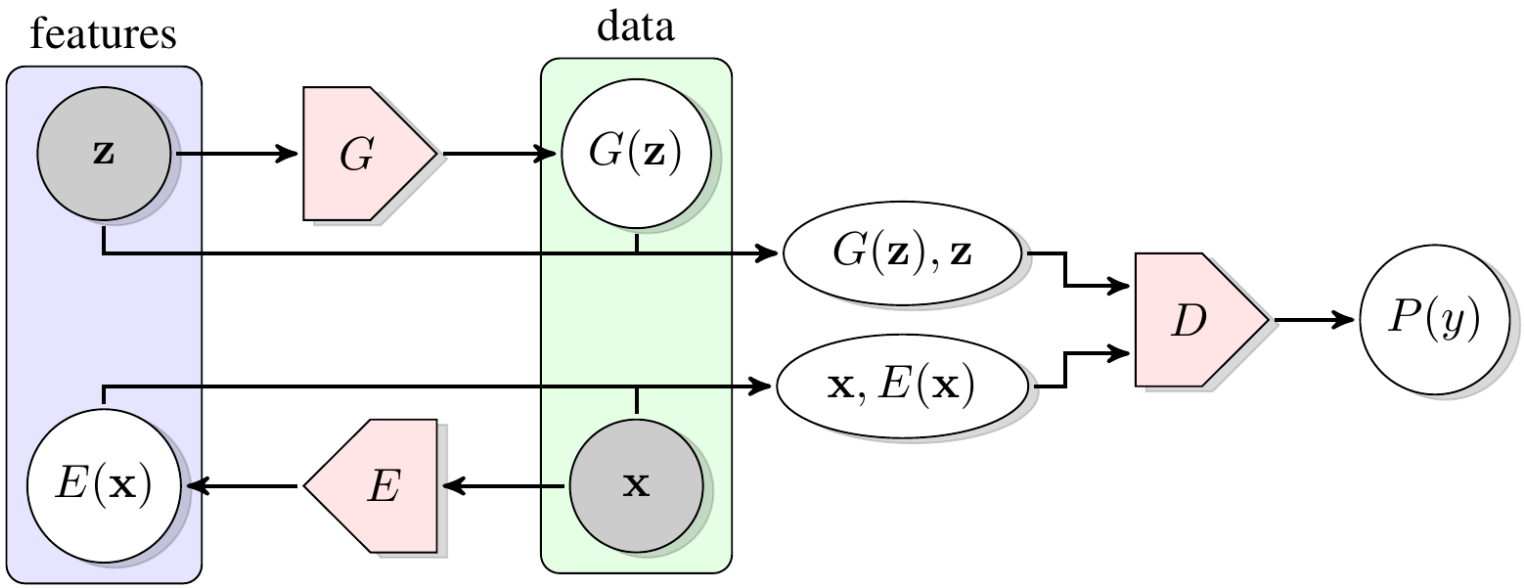


- Generator: maps latent variable $\mathbf{z}$ to data point $\mathbf{x}$
- Encoder: infers latent representation $\mathbf{z}$ from data point $\mathbf{x}$

Induced joint distributions over $(\mathbf{x}, \mathbf{z})$

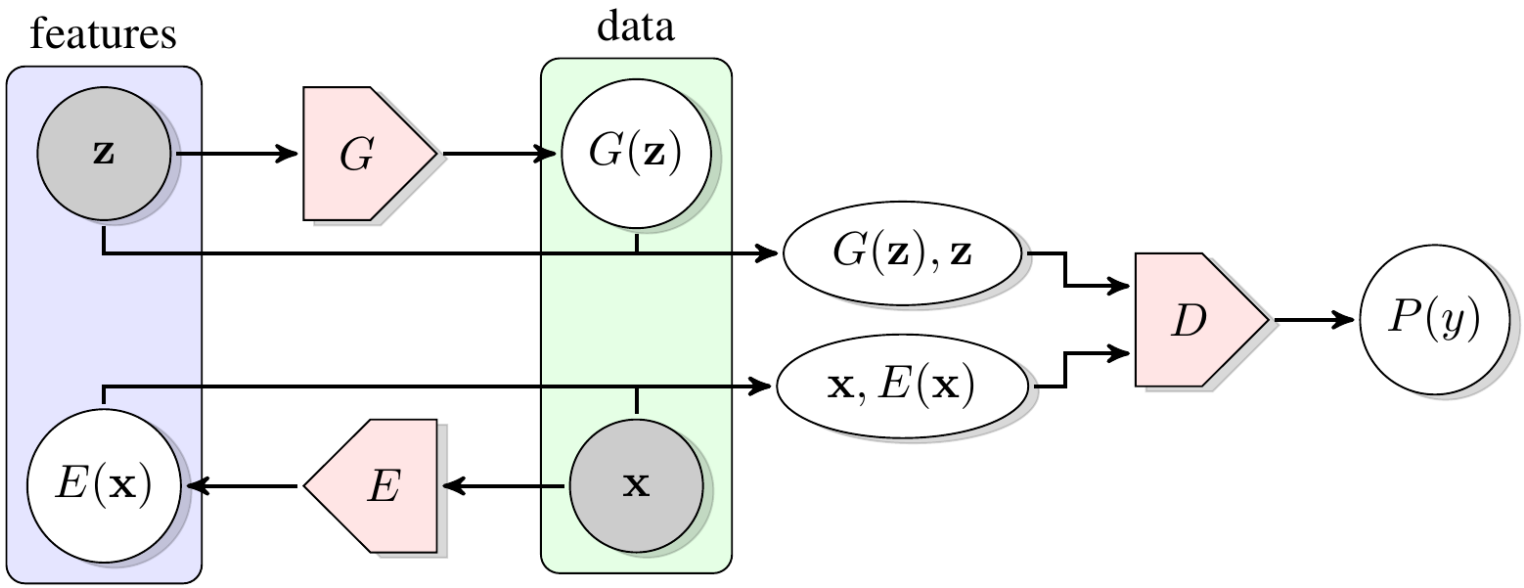


Induced joint distributions over $(\mathbf{x}, \mathbf{z})$



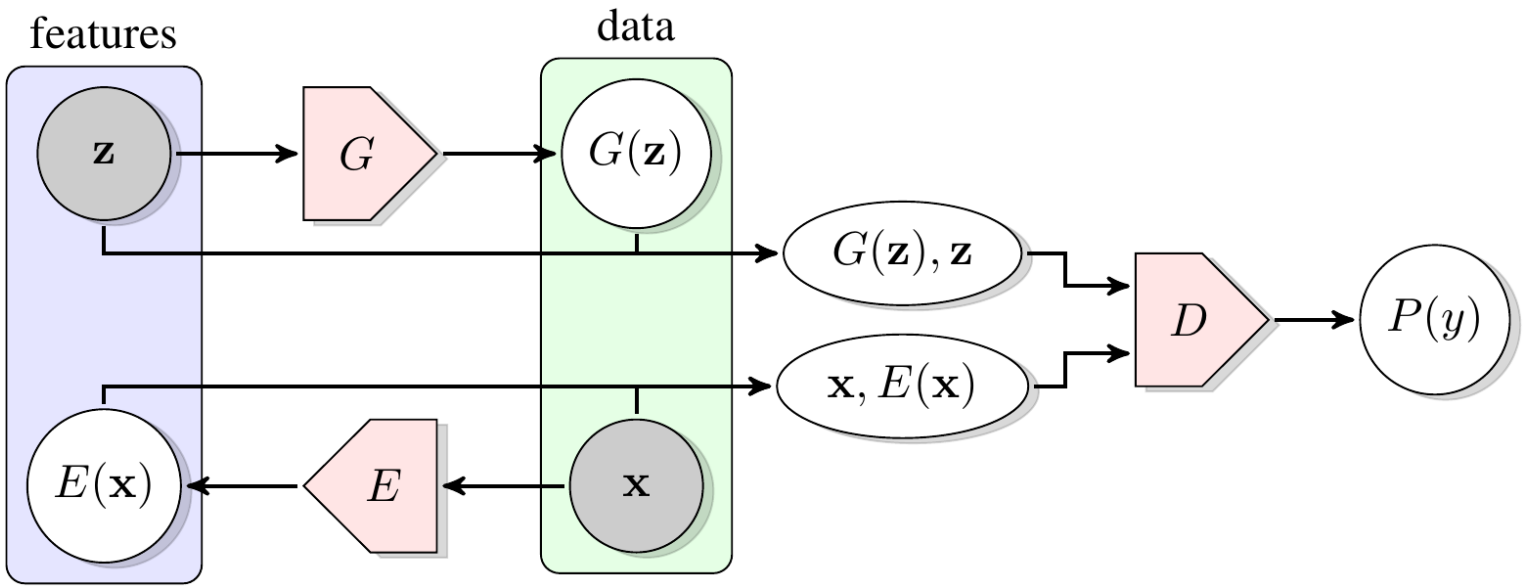
- Generator: $p_G(\mathbf{x}, \mathbf{z}) = p_{\mathbf{z}}(\mathbf{z})\ \delta\left(\mathbf{x} - G(\mathbf{z})\right)$

Induced joint distributions over $(\mathbf{x}, \mathbf{z})$



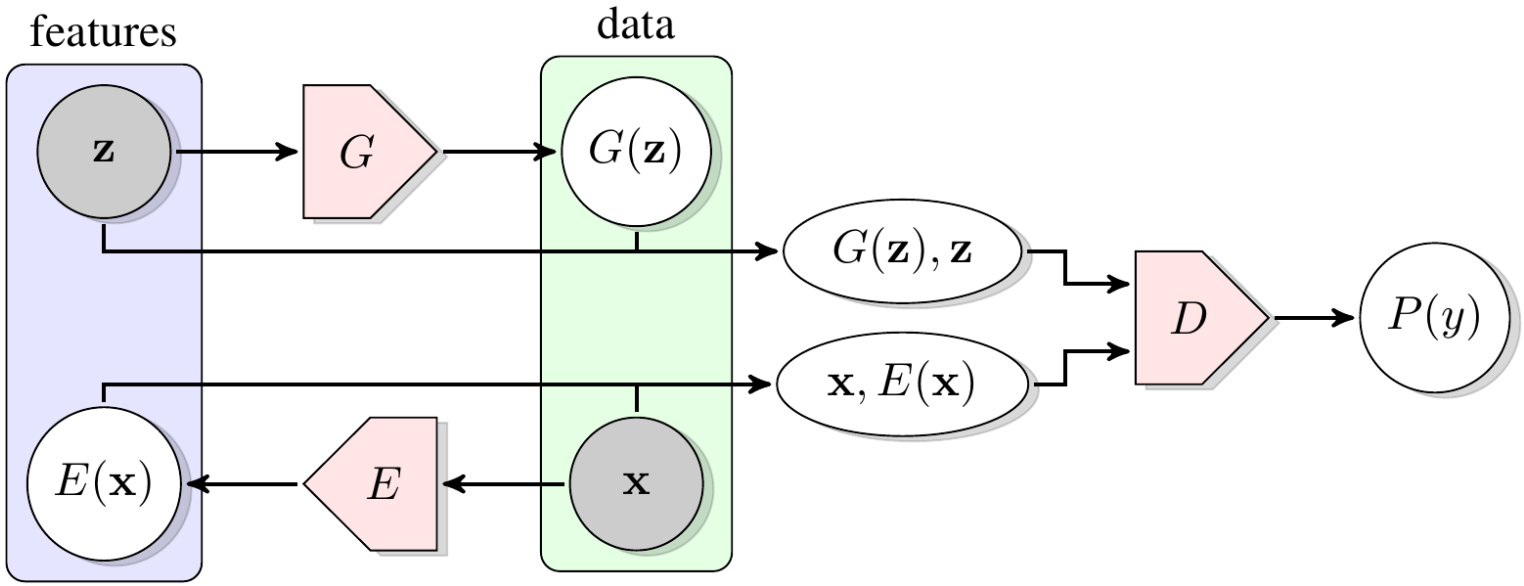
- Generator: $p_G(\mathbf{x}, \mathbf{z}) = p_{\mathbf{z}}(\mathbf{z})\ \delta\left(\mathbf{x} - G(\mathbf{z})\right)$
- Encoder: $p_E(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x})\ \delta\left(\mathbf{z} - E(\mathbf{x})\right)$

Induced joint distributions over $(\mathbf{x}, \mathbf{z})$



- Generator: $p_G(\mathbf{x}, \mathbf{z}) = p_{\mathbf{z}}(\mathbf{z})\ \delta\left(\mathbf{x} - G(\mathbf{z})\right)$
- Encoder: $p_E(\mathbf{x}, \mathbf{z}) = p_{\text{data}}(\mathbf{x})\ \delta\left(\mathbf{z} - E(\mathbf{x})\right)$
- Discriminator: pair $(\mathbf{x}, \mathbf{z})$ completed by generator or encoder?

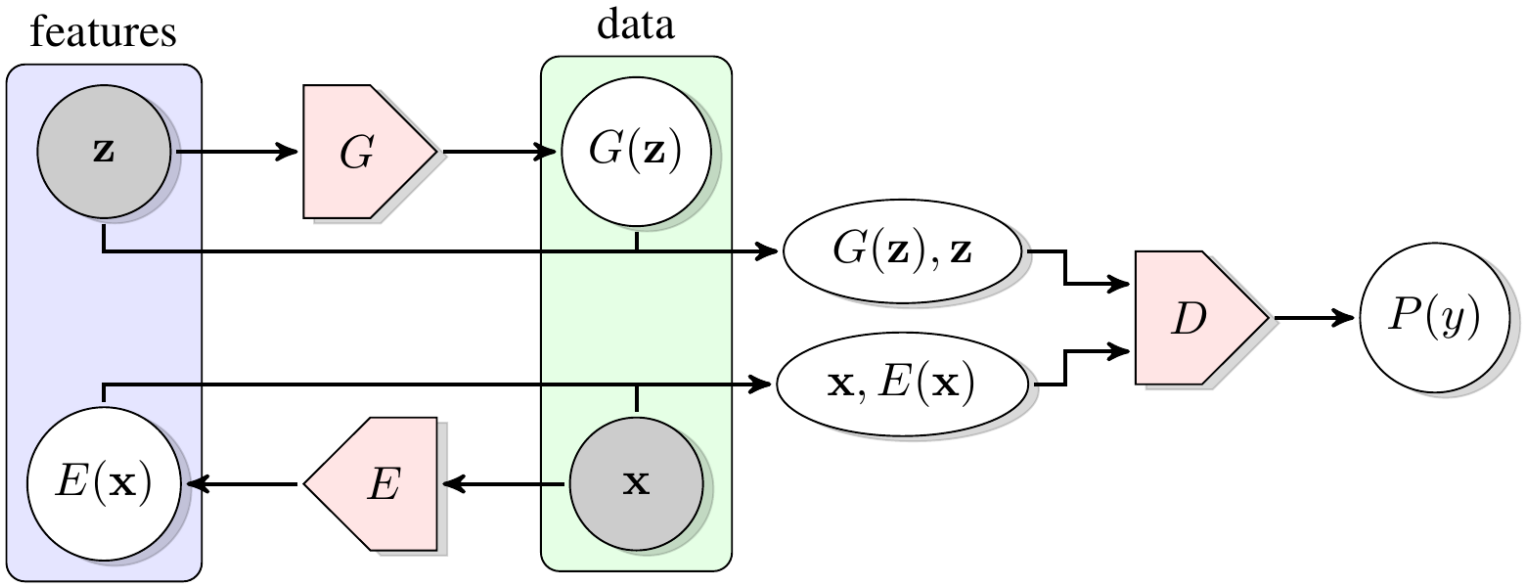
Bidirectional GANs [Donahue et al., 2017]



$$V(D, E, G) = \mathbb{E}_{p_{\text{data}}}[\ln D(\mathbf{x}, E(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}), \mathbf{z}))]$$

$$\min_{G,E} \max_{D} V(D, E, G)$$

Bidirectional GANs [Donahue et al., 2017]



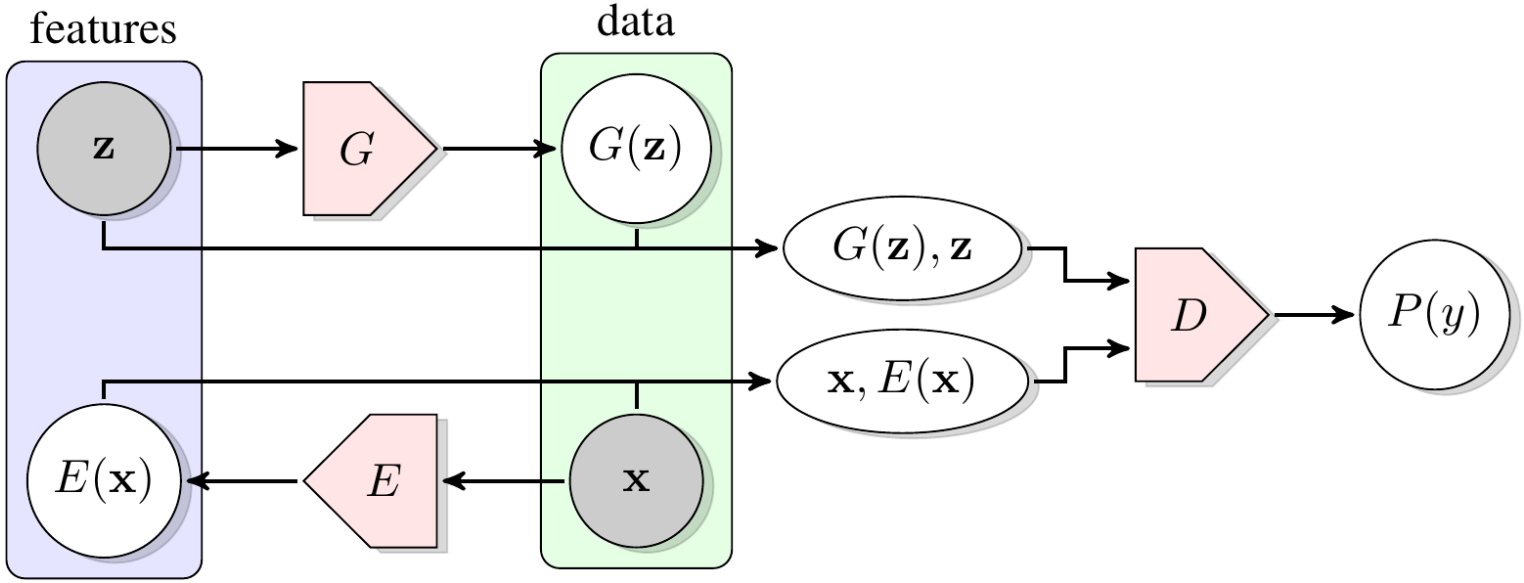
$$V(D, E, G) = \mathbb{E}_{p_{\text{data}}}[\ln D(\mathbf{x}, E(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}), \mathbf{z}))]$$

$$\min_{G,E} \max_{D} V(D, E, G)$$

- For optimal discriminator objective equals JS divergence

$$\max_{D} V(D, E, G) = 2D_{JS}\left(p_E(\mathbf{x}, \mathbf{z}) || p_G(\mathbf{x}, \mathbf{z})\right) - \ln 4$$

Bidirectional GANs [Donahue et al., 2017]



$$V(D, E, G) = \mathbb{E}_{p_{\text{data}}}[\ln D(\mathbf{x}, E(\mathbf{x}))] + \mathbb{E}_{p(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}), \mathbf{z}))]$$

$$\min_{G,E} \max_{D} V(D, E, G)$$

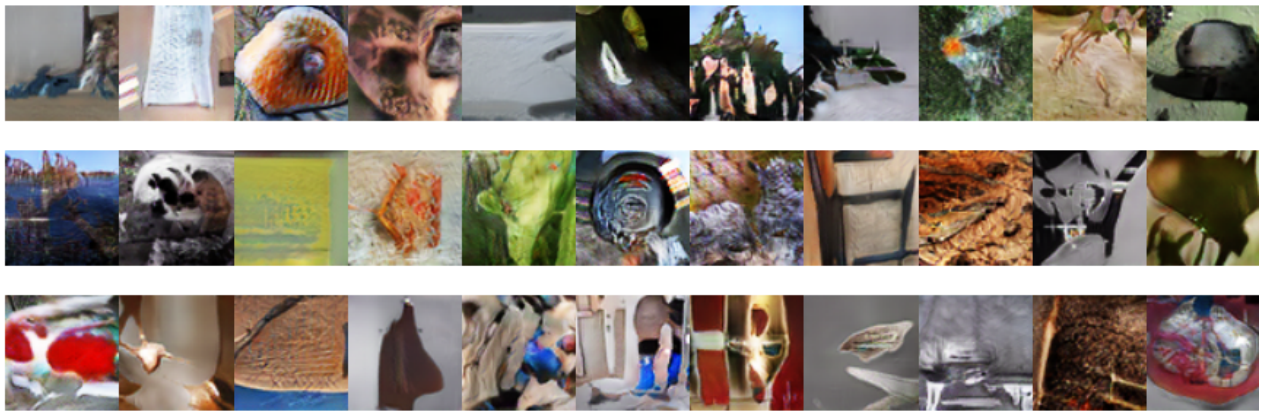
- For optimal discriminator objective equals JS divergence

$$\max_{D} V(D, E, G) = 2D_{JS}\left(p_E(\mathbf{x}, \mathbf{z}) || p_G(\mathbf{x}, \mathbf{z})\right) - \ln 4$$

- At optimum G and E are each others inverse

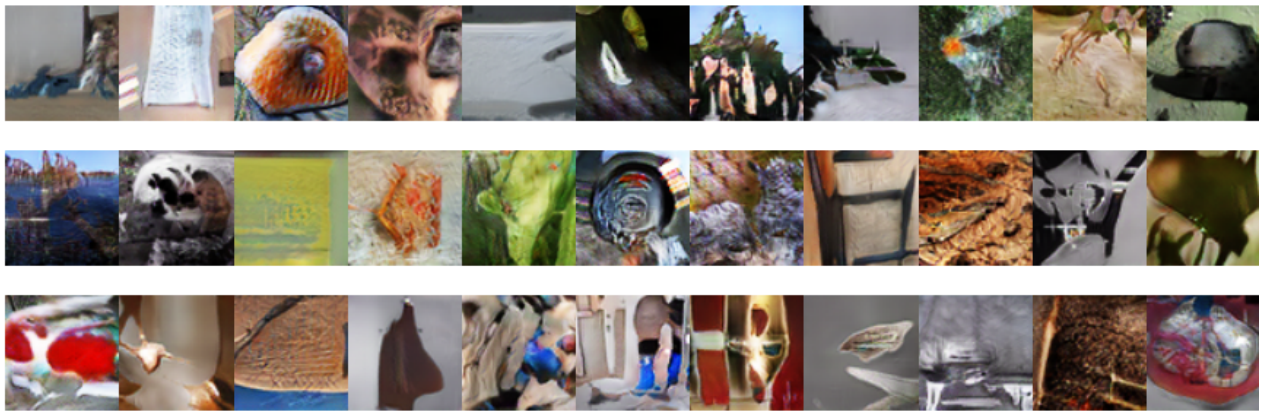
BiGAN samples, ImageNet 64×64

$G(\mathbf{z})$



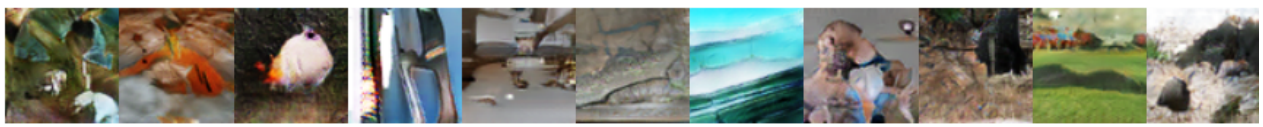
BiGAN samples, ImageNet 64×64

$G(\mathbf{z})$

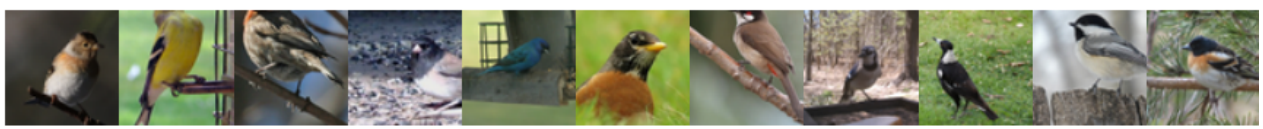


$\mathbf{x}$

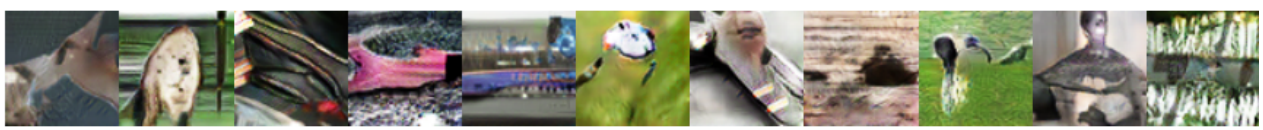
$G(E(\mathbf{x}))$



$\mathbf{x}$



$G(E(\mathbf{x}))$

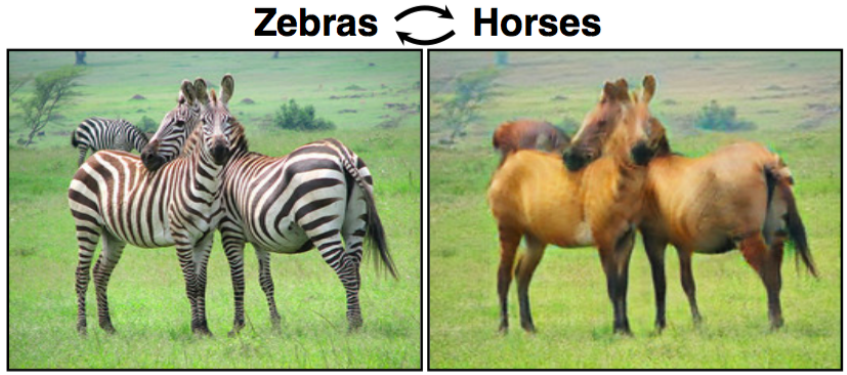


BiGAN: feature transfer to PASCAL VOC'07

	trained layers	Classification (% mAP) fc8	fc6-8	all	*FRCN* Detection (% mAP) all	*FCN* Segmentation (% mIU) all
sup.	ImageNet (Krizhevsky et al. 2012)	**77.0**	**78.8**	**78.3**	**56.8**	**48.0**
self-sup.	Agrawal et al. (2015)	31.2	31.0	54.2	43.9	-
	Pathak et al. (2016)	30.5	34.6	56.5	44.5	**30.0**
	Wang & Gupta (2015)	28.4	**55.6**	63.1	47.4	-
	Doersch et al. (2015)	**44.7**	55.1	**65.3**	**51.1**	-
unsup.	k-means (Krähenbühl et al. 2016)	32.0	39.2	56.6	45.6	32.6
	Discriminator (D)	30.7	40.5	56.4	-	-
	Latent Regressor (LR)	36.9	47.9	57.1	-	-
	Joint LR	37.1	47.9	56.5	-	-
	Autoencoder (ℓ_2)	24.8	16.0	53.8	41.9	-
	BiGAN (ours)	37.5	48.7	58.9	46.2	34.9
	BiGAN, 112×112 E (ours)	**40.7**	**52.3**	**60.1**	**46.9**	**35.2**

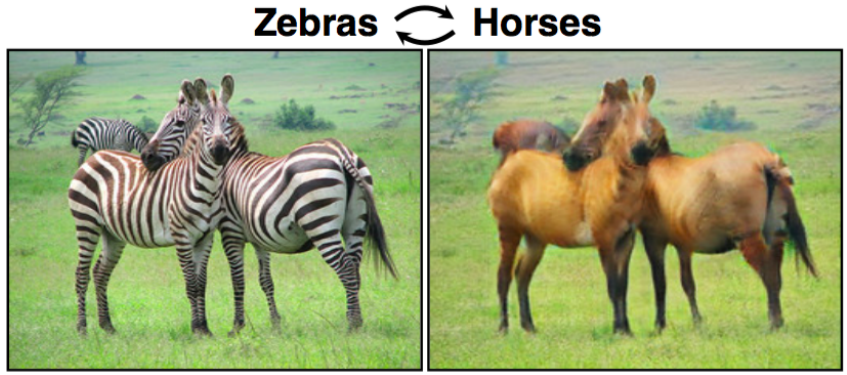
- Encoder network used to pre-train/initialize recognition net
- Similar performance of BiGAN and self-supervised methods
  - Self-sup: CNN trained using image structure as supervision
  - Discriminator: uses GAN discriminator layers as features
  - Latent regressor: trains network to invert GAN (jointly)

Unpaired image-to-image translation [Zhu et al., 2017]



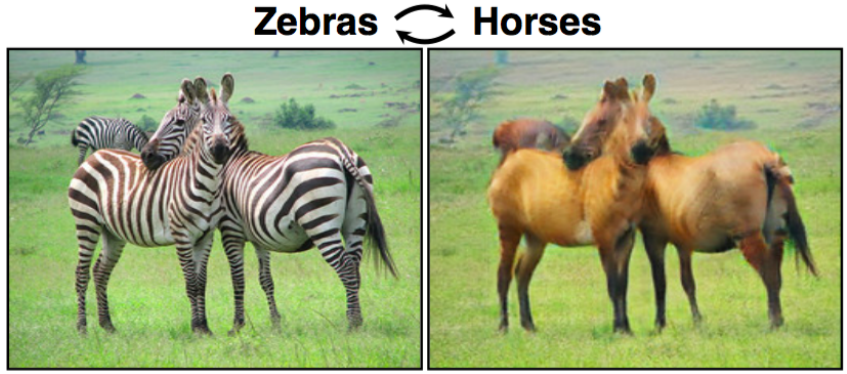
- Learn 2-way mapping between different image domains

Unpaired image-to-image translation [Zhu et al., 2017]



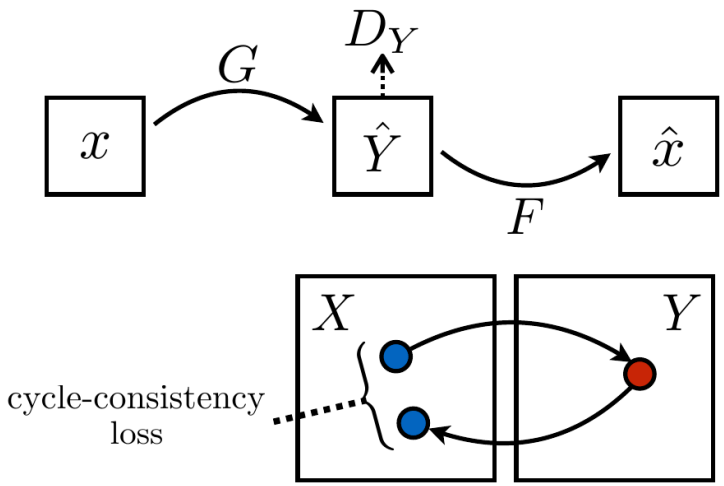
- Learn 2-way mapping between different image domains
- Without using supervised aligned training samples

Unpaired image-to-image translation [Zhu et al., 2017]

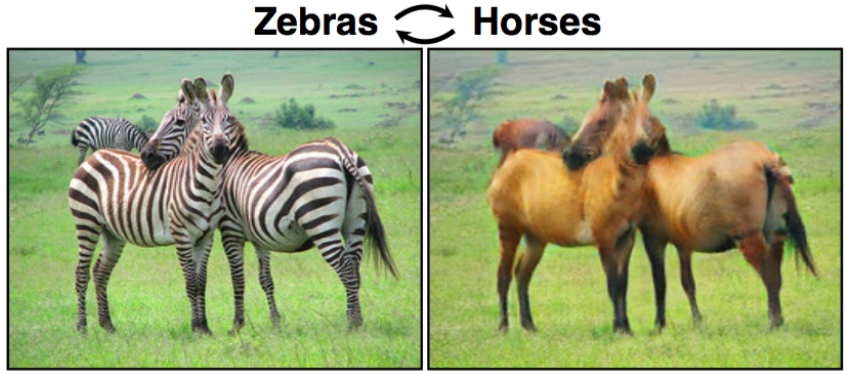


- Learn 2-way mapping between different image domains
- Without using supervised aligned training samples

1. Discriminator ensures realistic samples in each domain

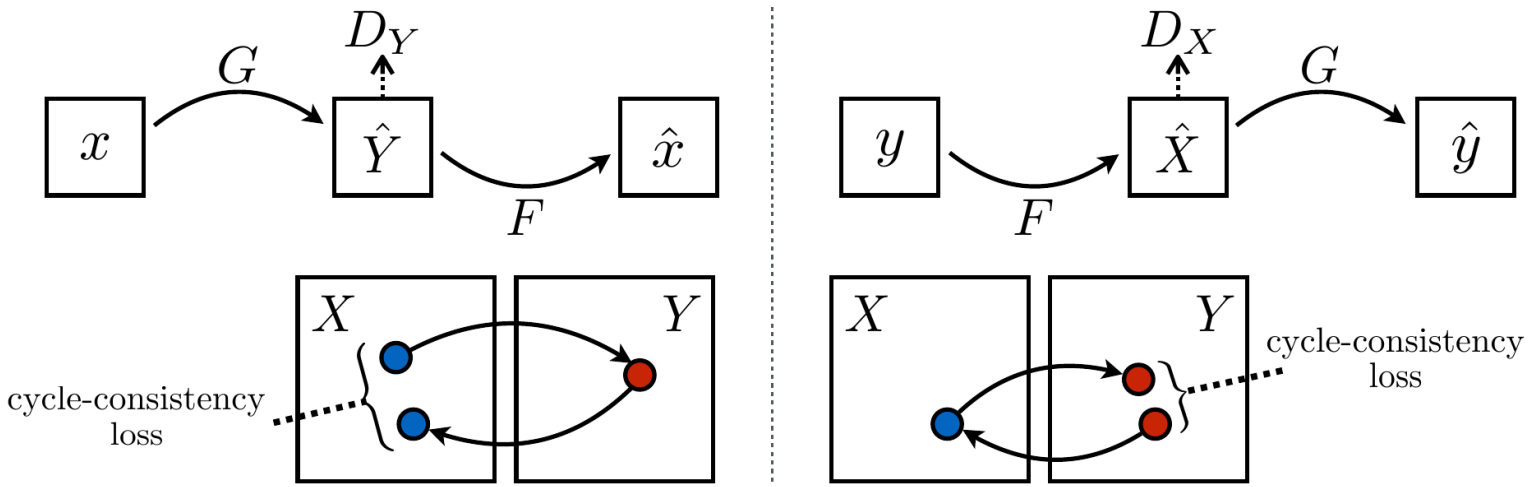


Unpaired image-to-image translation [Zhu et al., 2017]

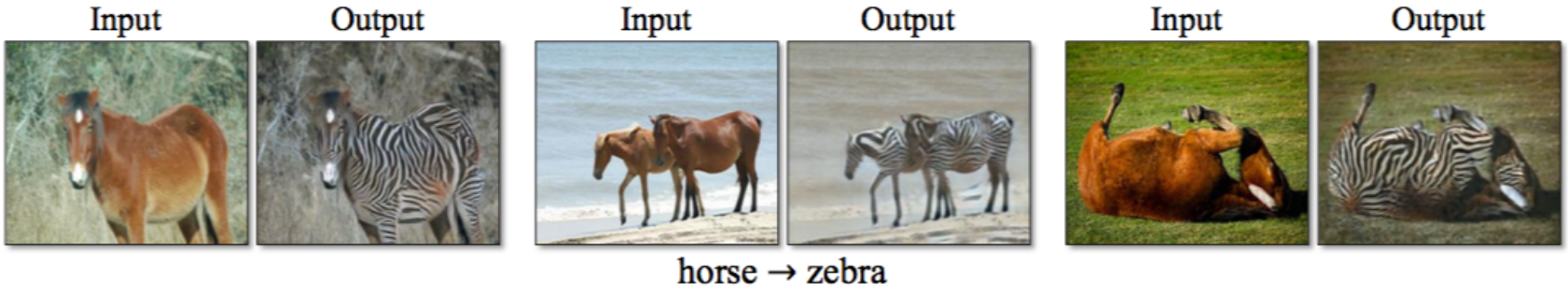


- Learn 2-way mapping between different image domains
- Without using supervised aligned training samples

1. Discriminator ensures realistic samples in each domain
2. Cycle-consistency loss ensures alignment

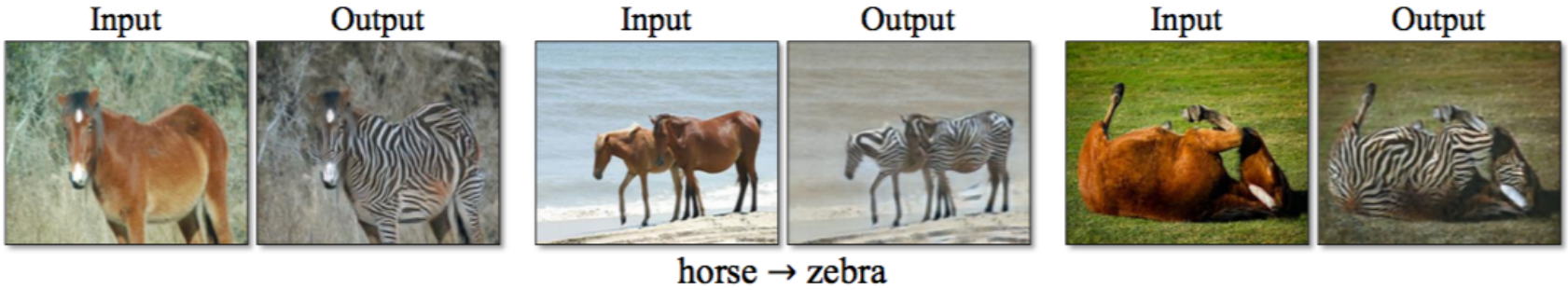


Some successful examples



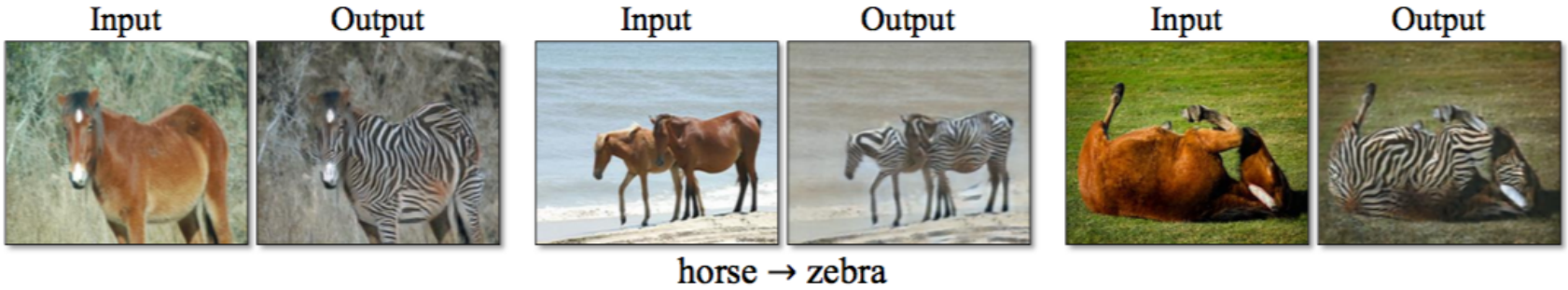
Some successful examples

- Without using any supervised/aligned examples!



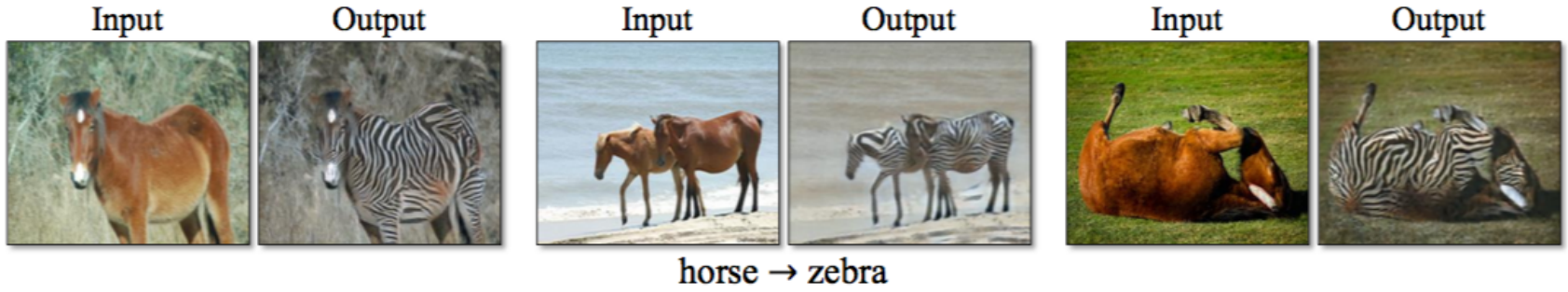
Some successful examples

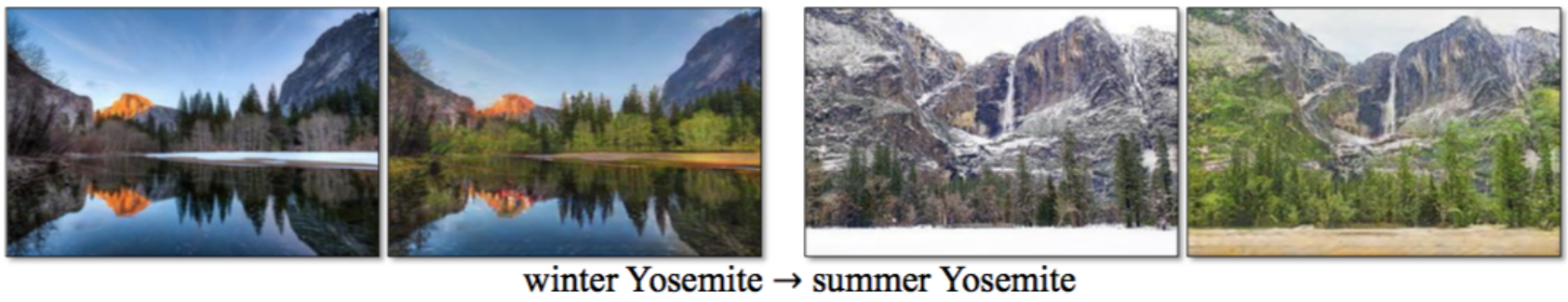
- Without using any supervised/aligned examples!



Some successful examples

- Without using any supervised/aligned examples!





And a failure case

Part II

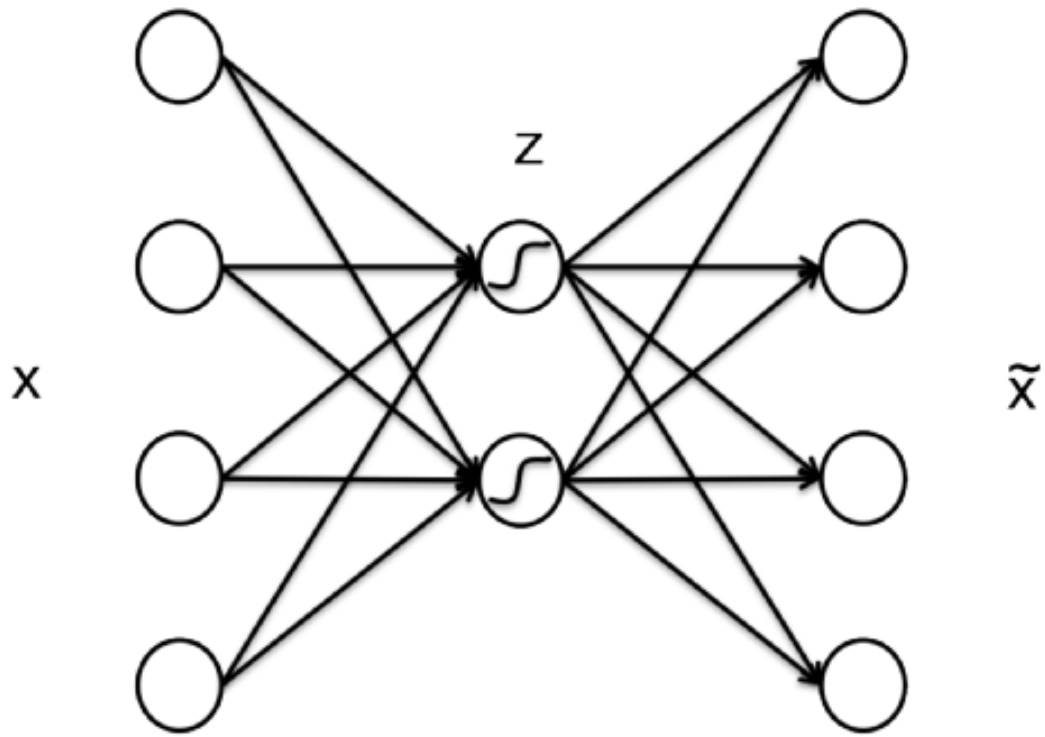
Variational Autoencoders

Autoencoders

- Learn latent representation $\mathbf{z}$ via reconstruction of data $\mathbf{x}$

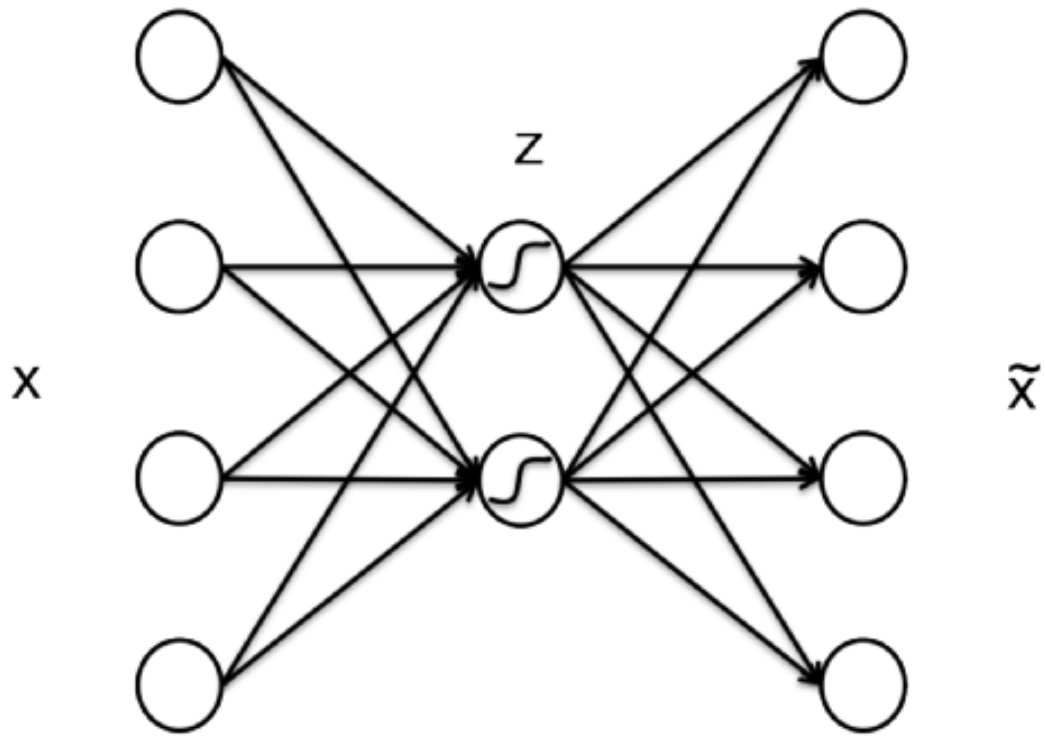
Autoencoders

- Learn latent representation $\mathbf{z}$ via reconstruction of data $\mathbf{x}$
- Neural network where output $\sim$ input
  - Encoder: maps data $\mathbf{x}$ to latent code $\mathbf{z}$
  - Decoder: maps latent code $\mathbf{z}$ to reconstruction $\tilde{\mathbf{x}}$



Autoencoders

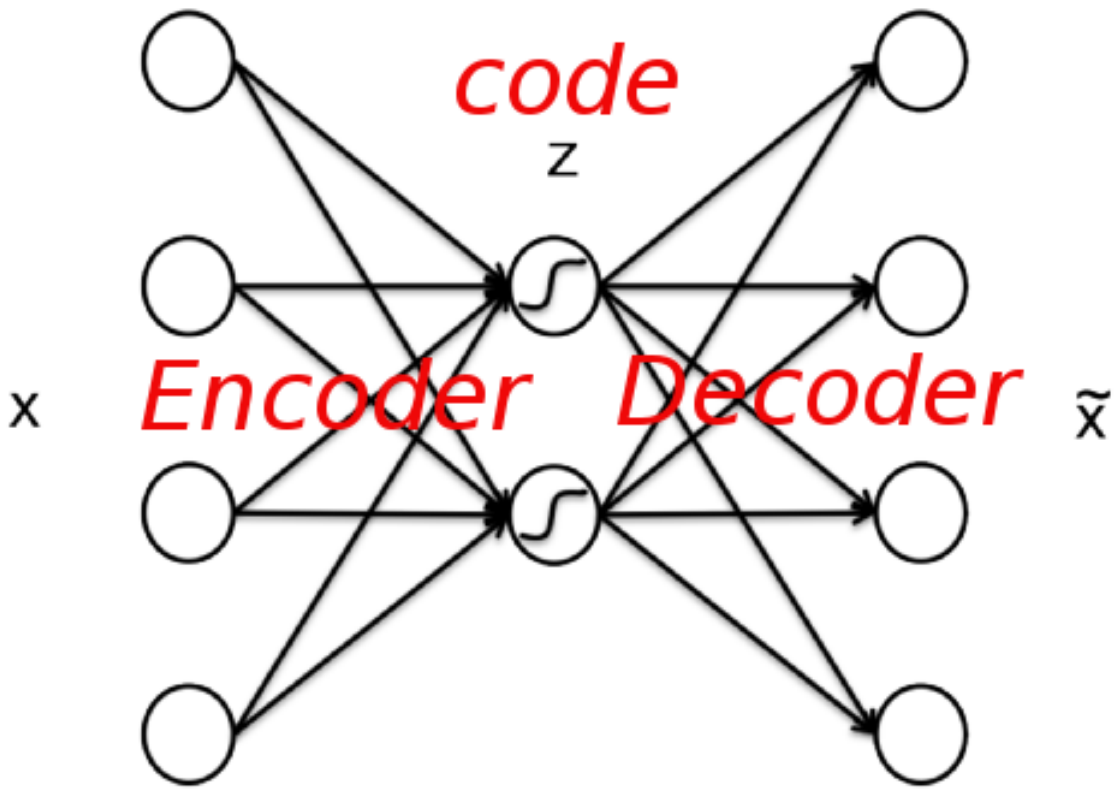
- Learn latent representation $\mathbf{z}$ via reconstruction of data $\mathbf{x}$
- Neural network where output $\sim$ input
  - Encoder: maps data $\mathbf{x}$ to latent code $\mathbf{z}$
  - Decoder: maps latent code $\mathbf{z}$ to reconstruction $\tilde{\mathbf{x}}$
- Loss minimizes discrepancy between $\mathbf{x}$ and $\tilde{\mathbf{x}}$



Relation autoencoders and PCA [Baldi and Hornik, 1989]

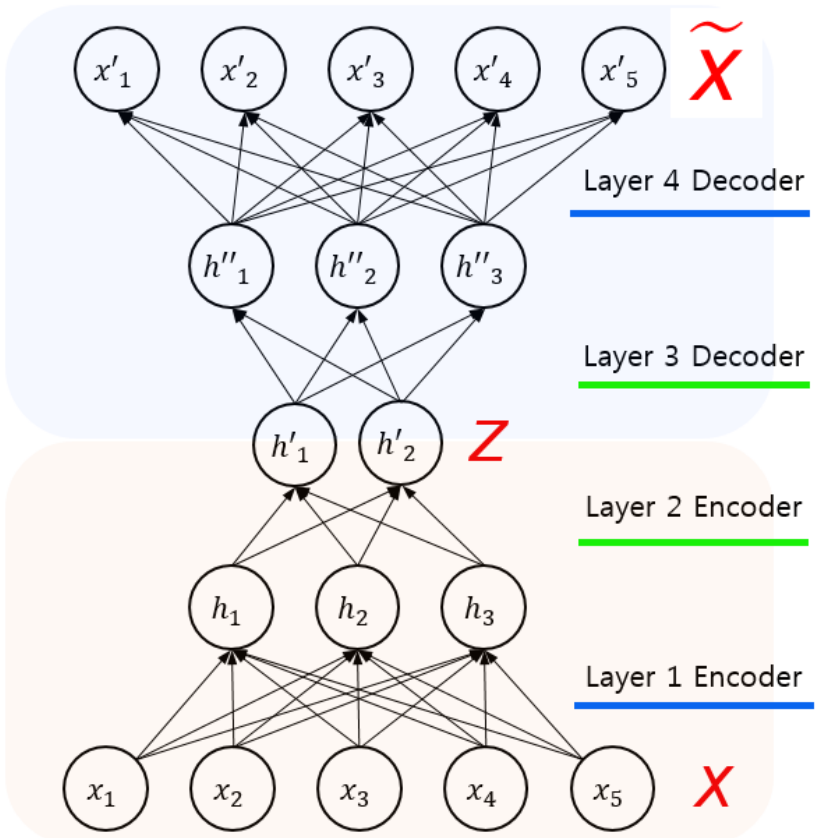
- Autoencoder recovers PCA if
  1. Encoder and decoder are both linear
  2. Optimizing ℓ_2 reconstruction loss

$$\min_{V,W} \frac{1}{2N} \sum_{n=1}^{N} ||x_n - VWx_n||^2 \tag{19}$$



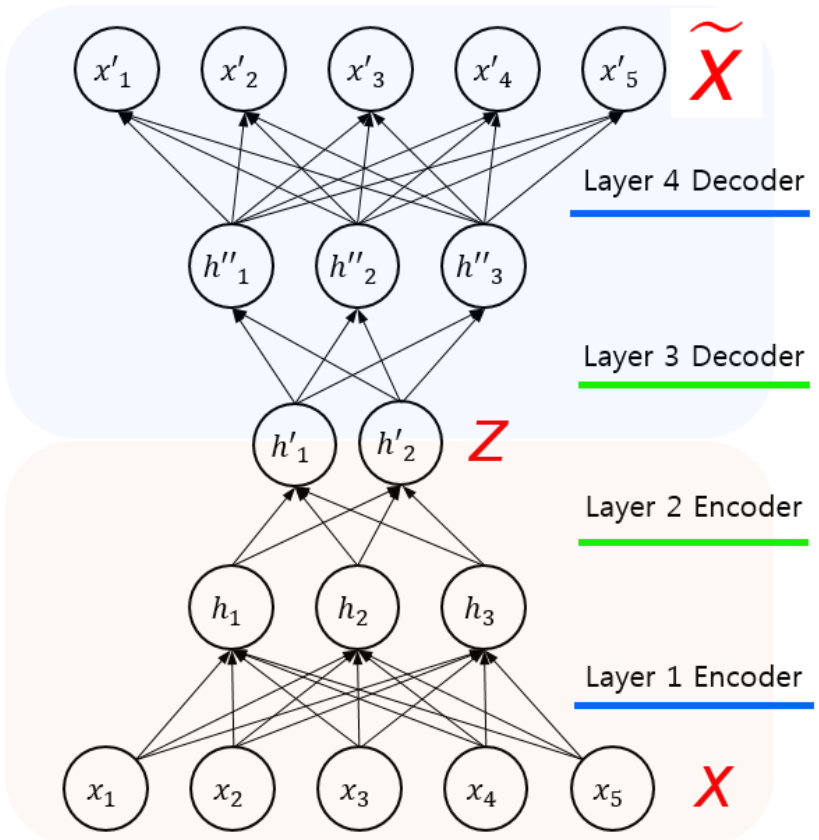
Deep non-linear autoencoders

- Stack many non-linear layers in encoder and decoder



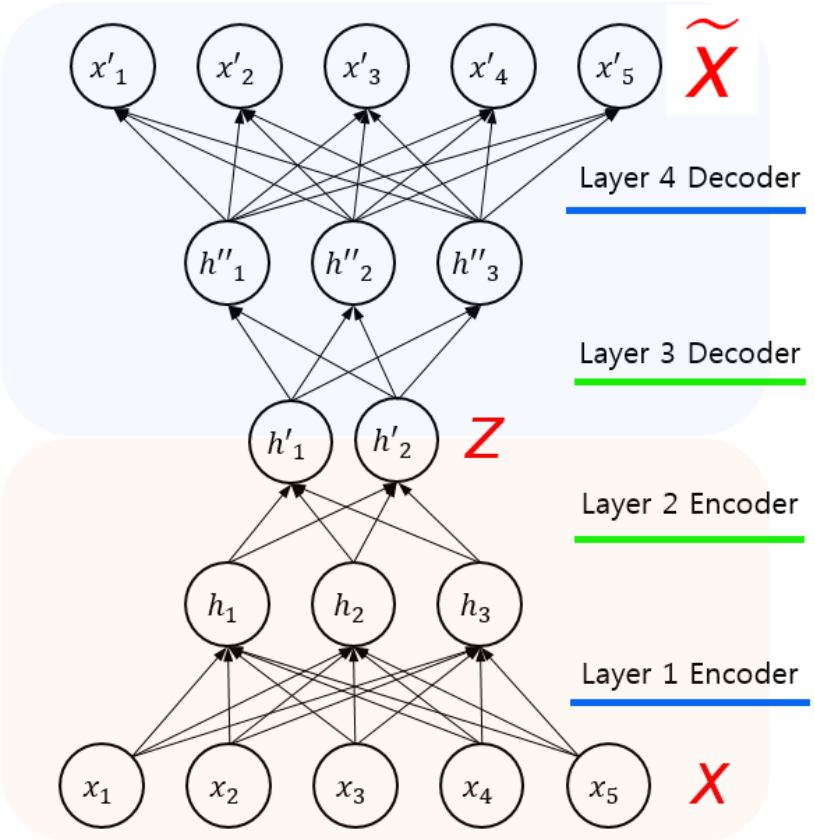
Deep non-linear autoencoders

- Stack many non-linear layers in encoder and decoder
- Non-linear representation learning



Deep non-linear autoencoders

- Stack many non-linear layers in encoder and decoder
- Non-linear representation learning
- Does not provide a generative model that can be sampled



Autoencoding variational Bayes [Kingma and Welling, 2014]

- Decoder f implements generative latent variable model
  - Maps latent code $\mathbf{z}$ to observation $\mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta^\mu(\mathbf{z}), f_\theta^\sigma(\mathbf{z})) \tag{20}$$

Autoencoding variational Bayes [Kingma and Welling, 2014]

- Decoder f implements generative latent variable model
  - Maps latent code $\mathbf{z}$ to observation $\mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta^\mu(\mathbf{z}), f_\theta^\sigma(\mathbf{z})) \tag{20}$$

- Encoder g compute approximate posterior distribution
  - Maps data $\mathbf{x}$ to latent code $\mathbf{z}$

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; g_\phi^\mu(\mathbf{x}), g_\phi^\sigma(\mathbf{x})) \tag{21}$$

Autoencoding variational Bayes [Kingma and Welling, 2014]

- Decoder f implements generative latent variable model
  - Maps latent code $\mathbf{z}$ to observation $\mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta^\mu(\mathbf{z}), f_\theta^\sigma(\mathbf{z})) \tag{20}$$

- Encoder g compute approximate posterior distribution
  - Maps data $\mathbf{x}$ to latent code $\mathbf{z}$

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; g_\phi^\mu(\mathbf{x}), g_\phi^\sigma(\mathbf{x})) \tag{21}$$

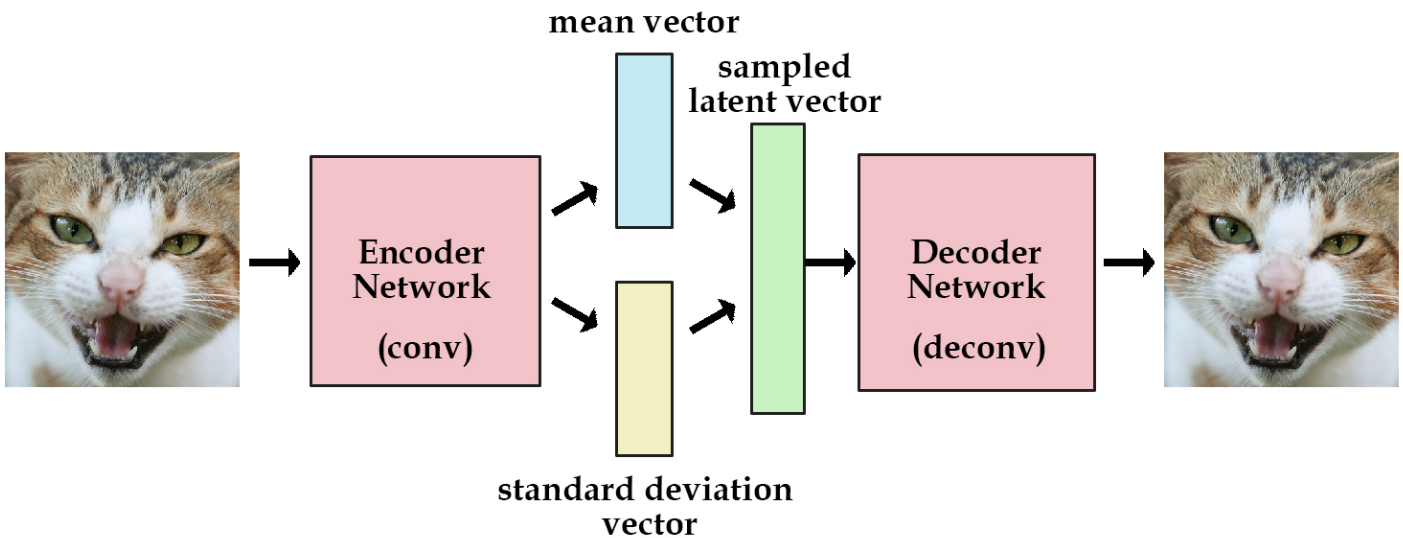


Figure from kvfrans@github

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{22}$$

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{22}$$

- Bound using variational approximation of posterior $p_\theta(\mathbf{z}|\mathbf{x})$

$$F(\theta, q) \equiv \ln p_\theta(\mathbf{x}) - D_{KL}\big(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\big) \leq \ln p_\theta(\mathbf{x}) \tag{23}$$

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{22}$$

- Bound using variational approximation of posterior $p_\theta(\mathbf{z}|\mathbf{x})$

$$F(\theta, q) \equiv \ln p_\theta(\mathbf{x}) - D_{KL}\big(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\big) \leq \ln p_\theta(\mathbf{x}) \tag{23}$$

- KL divergence: non-negative, and zero if and only if $q = p$

$$D_{KL}(q||p) = \int_{\mathbf{z}} q(\mathbf{z}) \ln \frac{q(\mathbf{z})}{p(\mathbf{z})} \tag{24}$$

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{22}$$

- Bound using variational approximation of posterior $p_\theta(\mathbf{z}|\mathbf{x})$

$$F(\theta, q) \equiv \ln p_\theta(\mathbf{x}) - D_{KL}\big(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\big) \leq \ln p_\theta(\mathbf{x}) \tag{23}$$

- KL divergence: non-negative, and zero if and only if $q = p$

$$D_{KL}(q||p) = \int_{\mathbf{z}} q(\mathbf{z}) \ln \frac{q(\mathbf{z})}{p(\mathbf{z})} \tag{24}$$

- Bound tight if variational distribution matches real posterior

EM algorithm performs coordinate ascent over ELBO

- Expectation step: Fix parameters θ, optimize over $q(\mathbf{z})$
  - Bound is tight if we can set $q(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$

$$F(\theta, q) \equiv \underbrace{\ln p_\theta(\mathbf{x})}_{\text{fixed}} - D_{KL}\big(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{25}$$

EM algorithm performs coordinate ascent over ELBO

- Expectation step: Fix parameters θ, optimize over $q(\mathbf{z})$
  - Bound is tight if we can set $q(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$

$$F(\theta, q) \equiv \underbrace{\ln p_\theta(\mathbf{x})}_{\text{fixed}} - D_{KL}\big(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{25}$$

- Maximization step: Fix $q(\mathbf{z})$, optimize model parameters θ
  - Log-marginal decomposed into log-prior and log-conditional

$$F(\theta, q) = \underbrace{H(q)}_{\text{fixed}} + \mathbb{E}_q[\ln p(\mathbf{z}) + \ln p_\theta(\mathbf{x}|\mathbf{z})] \tag{26}$$

EM algorithm performs coordinate ascent over ELBO

- Expectation step: Fix parameters θ, optimize over $q(\mathbf{z})$
  - Bound is tight if we can set $q(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$

$$F(\theta, q) \equiv \underbrace{\ln p_\theta(\mathbf{x})}_{\text{fixed}} - D_{KL}\big(q(\mathbf{z})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{25}$$

- Maximization step: Fix $q(\mathbf{z})$, optimize model parameters θ
  - Log-marginal decomposed into log-prior and log-conditional

$$F(\theta, q) = \underbrace{H(q)}_{\text{fixed}} + \mathbb{E}_q[\ln p(\mathbf{z}) + \ln p_\theta(\mathbf{x}|\mathbf{z})] \tag{26}$$

- Iterating EM steps monotonically increases evidence bound
  - Monotonically increases evidence with exact inference in E-step

Variational EM with inference net (amortized inference)

- Efficient feed-forward computation of approximate posterior

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; g_\phi^\mu(\mathbf{x}), g_\phi^\sigma(\mathbf{x})) \tag{27}$$

  - No iterative optimization of approx. posterior per data point
  - Avoids initialization and tracking posterior across iterations

Variational EM with inference net (amortized inference)

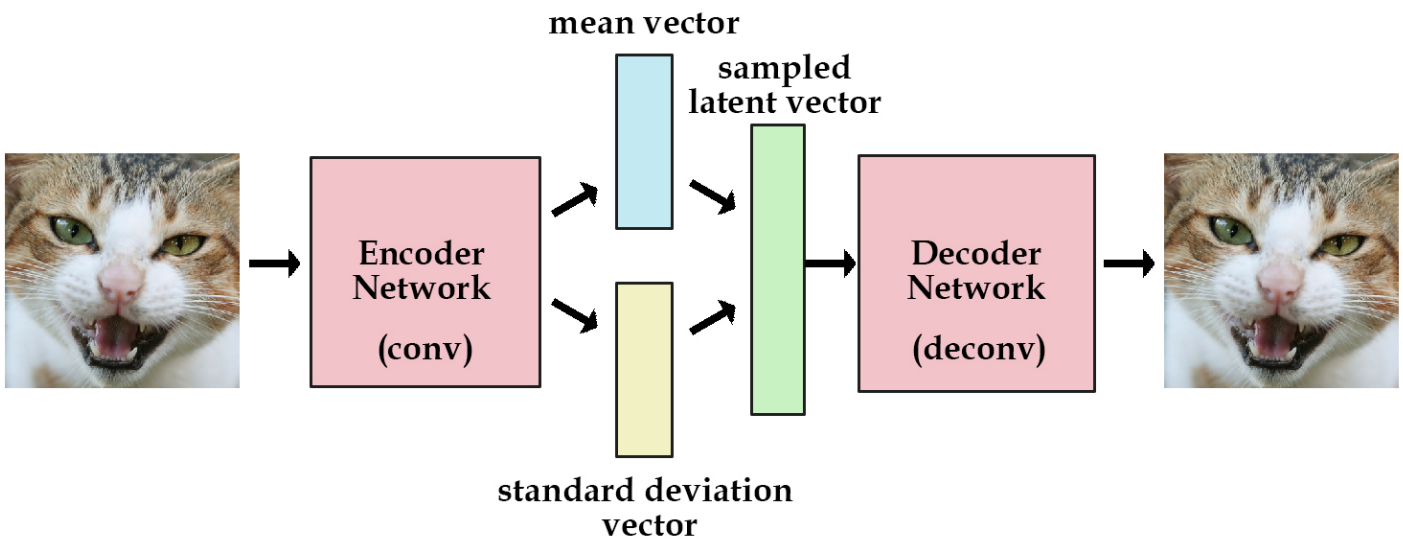
- Efficient feed-forward computation of approximate posterior

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; g_\phi^\mu(\mathbf{x}), g_\phi^\sigma(\mathbf{x})) \tag{27}$$

  - No iterative optimization of approx. posterior per data point
  - Avoids initialization and tracking posterior across iterations

- ELBO becomes function of inference net and generative net

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \tag{28}$$



Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\textbf{Regularization}} \tag{29}$$

- **Regularization term** keeps q from collapsing to single point $\mathbf{z}$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\textbf{Regularization}} \tag{29}$$

- **Regularization term** keeps q from collapsing to single point $\mathbf{z}$
- Closed form if both terms are Gaussian, for $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I)$

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right) = \frac{1}{2}\left[1 + \ln g_\phi^\sigma(\mathbf{x}) - g_\phi^\mu(\mathbf{x}) - g_\phi^\sigma(\mathbf{x})\right] \tag{30}$$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\textbf{Regularization}} \tag{29}$$

- **Regularization term** keeps q from collapsing to single point $\mathbf{z}$
- Closed form if both terms are Gaussian, for $p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I)$

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right) = \frac{1}{2}\left[1 + \ln g_\phi^\sigma(\mathbf{x}) - g_\phi^\mu(\mathbf{x}) - g_\phi^\sigma(\mathbf{x})\right] \tag{30}$$

- Differentiable function of inference net parameters

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\textbf{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{31}$$

- **Reconstruction term**: to what extent can $\mathbf{x}$ be reconstructed from $\mathbf{z}$ following approximate posterior $q(\mathbf{z}|\mathbf{x})$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\textbf{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{31}$$

- **Reconstruction term**: to what extent can $\mathbf{x}$ be reconstructed from $\mathbf{z}$ following approximate posterior $q(\mathbf{z}|\mathbf{x})$
- Use sample approximation of intractable expectation $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x})$

$$\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] \approx \frac{1}{S}\sum_{s=1}^{S} \ln p_\theta(\mathbf{x}|\mathbf{z_s}) \tag{32}$$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\textbf{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{31}$$

- **Reconstruction term**: to what extent can $\mathbf{x}$ be reconstructed from $\mathbf{z}$ following approximate posterior $q(\mathbf{z}|\mathbf{x})$
- Use sample approximation of intractable expectation $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x})$

$$\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] \approx \frac{1}{S}\sum_{s=1}^{S} \ln p_\theta(\mathbf{x}|\mathbf{z_s}) \tag{32}$$

- Estimator is non-differentiable due to sampling operator

Re-parametrization trick

- Side-step non-differentiable sampling operator by re-parametrizing samples $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; g^\mu_\phi(\mathbf{x}), g^\sigma_\phi(\mathbf{x})\right)$

Re-parametrization trick

- Side-step non-differentiable sampling operator by re-parametrizing samples $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; g^\mu_\phi(\mathbf{x}), g^\sigma_\phi(\mathbf{x})\right)$
- Use inference net to modulate samples from a unit Gaussian

$$\mathbf{z}_s = g^\mu_\phi(\mathbf{x}) + g^\sigma_\phi(\mathbf{x}) \odot \epsilon_s, \qquad \epsilon_s \sim \mathcal{N}(\epsilon_s; 0, I) \tag{33}$$

Re-parametrization trick

- Side-step non-differentiable sampling operator by re-parametrizing samples $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; g^\mu_\phi(\mathbf{x}), g^\sigma_\phi(\mathbf{x})\right)$
- Use inference net to modulate samples from a unit Gaussian

$$\mathbf{z}_s = g^\mu_\phi(\mathbf{x}) + g^\sigma_\phi(\mathbf{x}) \odot \epsilon_s, \qquad \epsilon_s \sim \mathcal{N}(\epsilon_s; 0, I) \tag{33}$$

- Samples $\mathbf{z}_s$ differentiable function of inference net param. $\boldsymbol{\phi}$, given unit Gaussian samples ϵ_s

Re-parametrization trick

- Side-step non-differentiable sampling operator by re-parametrizing samples $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}\left(\mathbf{z}; g^\mu_\phi(\mathbf{x}), g^\sigma_\phi(\mathbf{x})\right)$
- Use inference net to modulate samples from a unit Gaussian

$$\mathbf{z}_s = g^\mu_\phi(\mathbf{x}) + g^\sigma_\phi(\mathbf{x}) \odot \epsilon_s, \qquad \epsilon_s \sim \mathcal{N}(\epsilon_s; 0, I) \tag{33}$$

- Samples $\mathbf{z}_s$ differentiable function of inference net param. $\boldsymbol{\phi}$, given unit Gaussian samples ϵ_s
- Unbiased differentiable approximation of ELBO

$$F(\theta, \phi) \approx \frac{1}{S}\sum_{s=1}^{S} \ln p_\theta\left(\mathbf{x}|g^\mu_\phi(\mathbf{x}) + g^\sigma_\phi(\mathbf{x}) \odot \epsilon_s\right) \tag{34}$$

$$-\frac{1}{2}\left[1 + \ln g^\sigma_\phi(\mathbf{x}) - g^\mu_\phi(\mathbf{x}) - g^\sigma_\phi(\mathbf{x})\right] \tag{35}$$

Re-parametrization trick in a cartoon

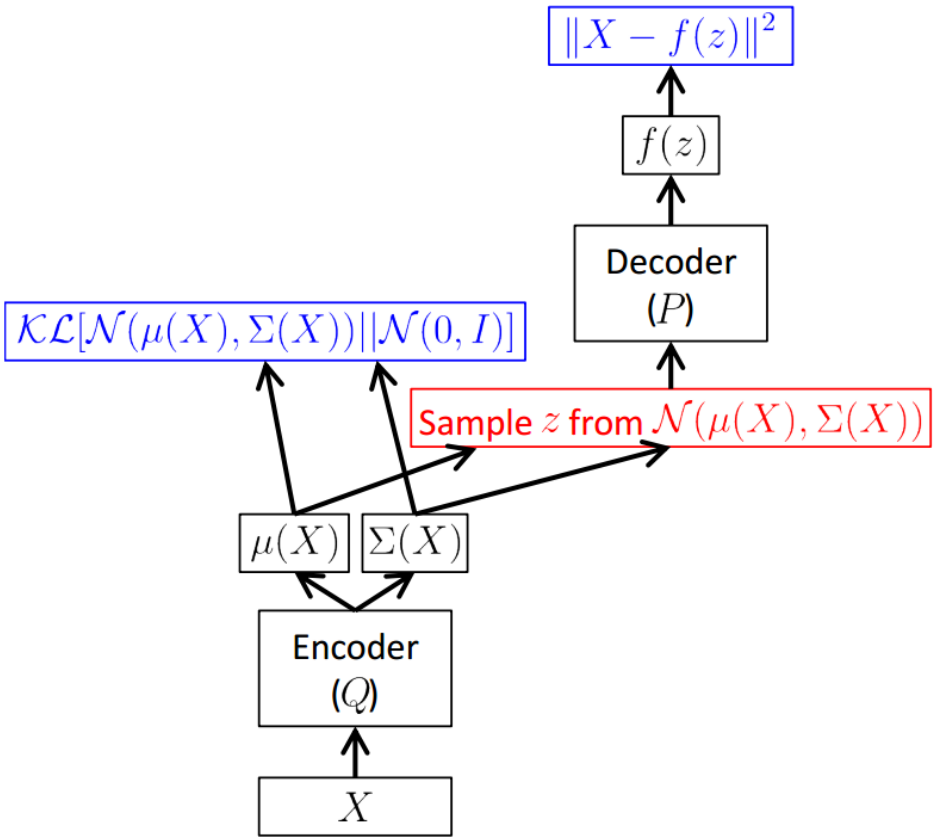


Figure from [Doersch, 2016]

Re-parametrization trick in a cartoon

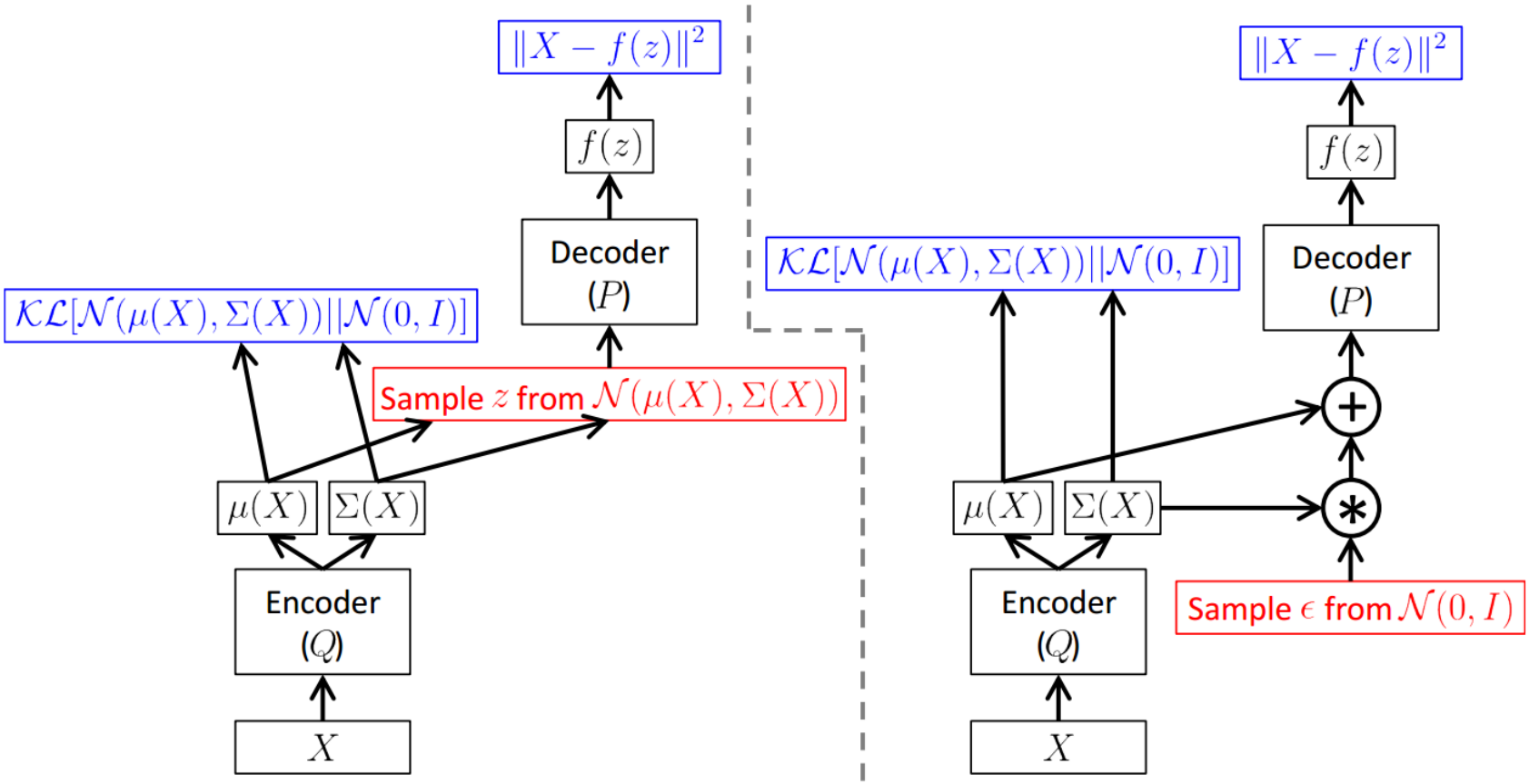


Figure from [Doersch, 2016]

Autoencoding variational Bayes training algorithm

- For each data point $\mathbf{x}$ in a mini-batch
  1. Sample one or multiple values $\{\epsilon_s\}$
  2. Use back-propagation to compute
     $g_\theta = \nabla_\theta F(\theta, \phi, \{\epsilon_s\})$
     $g_\phi = \nabla_\phi F(\theta, \phi, \{\epsilon_s\})$
  3. Gradient-based parameter update

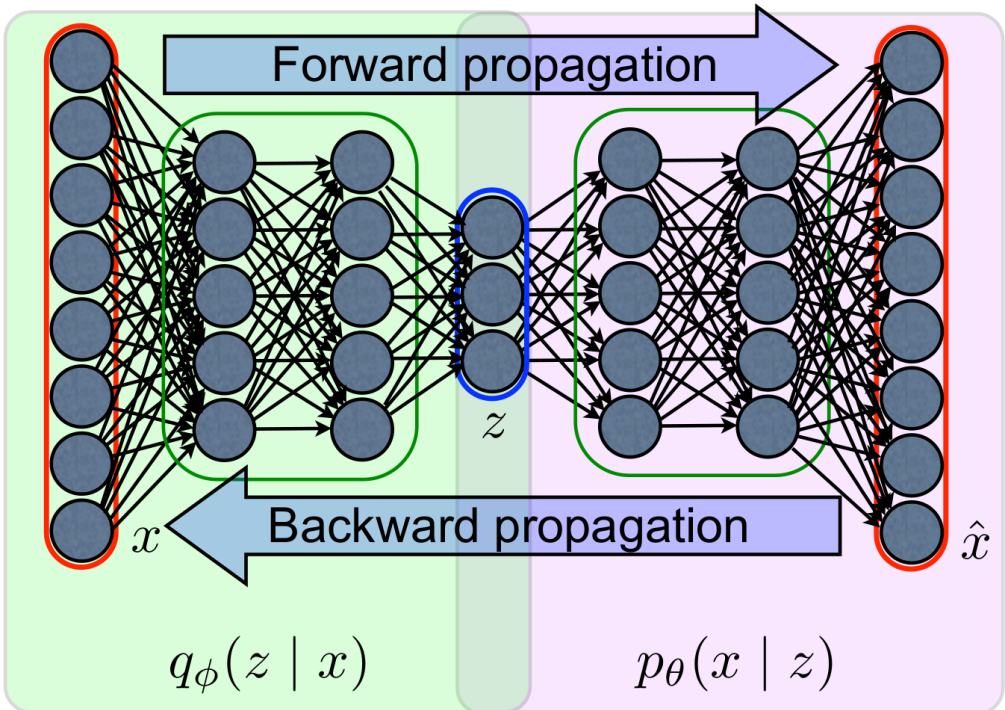


Figure from Aaron Courville

Random samples from VAE and GAN

- Trained from 200k images in CelebA dataset

Figure from [Hou et al., 2016]

Random samples from VAE and GAN

- Trained from 200k images in CelebA dataset
- VAE samples appear overly smooth / blurred

Figure from [Hou et al., 2016]

Random samples from VAE and GAN

- Trained from 200k images in CelebA dataset
- VAE samples appear overly smooth / blurred
- GAN samples show more (imperfect) detail

Figure from [Hou et al., 2016]

Semi-supevised learning with VAE [Kingma et al., 2014]

Semi-supevised learning with VAE [Kingma et al., 2014]

- Generative model: Add class label y,
  latent variable $\mathbf{z}$ models in-class variations

$$p_\pi(y) = \mathrm{Cat}\,(y;\pi) \qquad\qquad p(\mathbf{z}) = \mathcal{N}\,(\mathbf{z};0,I)$$

Semi-supevised learning with VAE [Kingma et al., 2014]

- Generative model: Add class label y,
  latent variable $\mathbf{z}$ models in-class variations

$$p_\pi(y) = \mathrm{Cat}\left(y; \pi\right) \qquad p(\mathbf{z}) = \mathcal{N}\left(\mathbf{z}; 0, I\right)$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_\theta(y, \mathbf{z}), \sigma^2_\theta(y, \mathbf{z})\right)$$

Semi-supevised learning with VAE [Kingma et al., 2014]

- Generative model: Add class label y,
  latent variable $\mathbf{z}$ models in-class variations

$$p_\pi(y) = \mathrm{Cat}\left(y; \pi\right) \qquad p(\mathbf{z}) = \mathcal{N}\left(\mathbf{z}; 0, I\right)$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_\theta(y, \mathbf{z}), \sigma^2_\theta(y, \mathbf{z})\right)$$

- Inference network

Semi-supevised learning with VAE [Kingma et al., 2014]

- Generative model: Add class label y,
  latent variable $\mathbf{z}$ models in-class variations

$$p_\pi(y) = \mathrm{Cat}\left(y; \pi\right) \qquad p(\mathbf{z}) = \mathcal{N}\left(\mathbf{z}; 0, I\right)$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_\theta(y, \mathbf{z}), \sigma^2_\theta(y, \mathbf{z})\right)$$

- Inference network
  1. Posterior on class-label acts as classifier

$$q_\phi(y|\mathbf{x}) = \mathrm{Cat}\left(y; \pi_\phi(\mathbf{x})\right), \tag{36}$$

Semi-supevised learning with VAE [Kingma et al., 2014]

- Generative model: Add class label y,
  latent variable $\mathbf{z}$ models in-class variations

$$p_\pi(y) = \mathrm{Cat}\left(y; \pi\right) \qquad p(\mathbf{z}) = \mathcal{N}\left(\mathbf{z}; 0, I\right)$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_\theta(y, \mathbf{z}), \sigma^2_\theta(y, \mathbf{z})\right)$$

- Inference network
  1. Posterior on class-label acts as classifier

$$q_\phi(y|\mathbf{x}) = \mathrm{Cat}\left(y; \pi_\phi(\mathbf{x})\right), \tag{36}$$

  2. Class-conditional posterior on $\mathbf{z}$

$$q_\phi(\mathbf{z}|\mathbf{x}, y) = \mathcal{N}\left(\mathbf{z}; \mu_\phi(\mathbf{x}, y), \sigma^2_\phi(\mathbf{x}, y)\right) \tag{37}$$

Semi-supevised learning with VAE [Kingma et al., 2014]

- Generative model: Add class label y,
  latent variable $\mathbf{z}$ models in-class variations

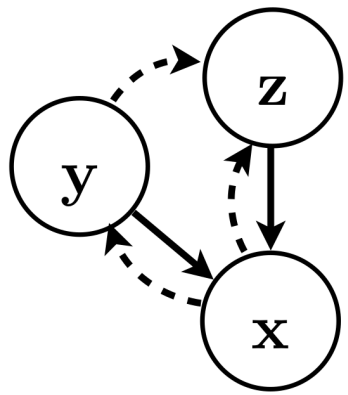
$$p_\pi(y) = \mathrm{Cat}\left(y; \pi\right) \qquad p(\mathbf{z}) = \mathcal{N}\left(\mathbf{z}; 0, I\right)$$
$$p_\theta(\mathbf{x}|y, \mathbf{z}) = \mathcal{N}\left(\mathbf{x}; \mu_\theta(y, \mathbf{z}), \sigma^2_\theta(y, \mathbf{z})\right)$$

- Inference network
  1. Posterior on class-label acts as classifier

$$q_\phi(y|\mathbf{x}) = \mathrm{Cat}\left(y; \pi_\phi(\mathbf{x})\right), \tag{36}$$

  2. Class-conditional posterior on $\mathbf{z}$

$$q_\phi(\mathbf{z}|\mathbf{x}, y) = \mathcal{N}\left(\mathbf{z}; \mu_\phi(\mathbf{x}, y), \sigma^2_\phi(\mathbf{x}, y)\right) \tag{37}$$



Objective function for semi-supervised model

- Objective function has three terms

$$\mathcal{J} = \alpha \underbrace{\sum_{(\mathbf{x},y)\sim\tilde{p}_l} \ln q_\phi(y|\mathbf{x})}_{\text{discriminative}} \tag{38}$$

1. Discriminative term: trains classifier from labeled data

Objective function for semi-supervised model

- Objective function has three terms

$$\mathcal{J} = \alpha \underbrace{\sum_{(\mathbf{x},y)\sim\tilde{p}_l} \ln q_\phi(y|\mathbf{x})}_{\text{discriminative}} + \underbrace{\sum_{(\mathbf{x},y)\sim\tilde{p}_l} \mathcal{L}(\mathbf{x},y)}_{\text{labeled}} \tag{38}$$

1. Discriminative term: trains classifier from labeled data
2. Generative labeled data: infer latent variable z

$$\mathcal{L}(\mathbf{x},y) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},y)}[\ln p_\theta(\mathbf{x},y,\mathbf{z}) - \ln q_\phi(\mathbf{z}|\mathbf{x},y)] \leq \ln p(\mathbf{x},y)$$

Objective function for semi-supervised model

- Objective function has three terms

$$\mathcal{J} = \alpha \underbrace{\sum_{(\mathbf{x},y)\sim\tilde{p}_l} \ln q_\phi(y|\mathbf{x})}_{\text{discriminative}} + \underbrace{\sum_{(\mathbf{x},y)\sim\tilde{p}_l} \mathcal{L}(\mathbf{x},y)}_{\text{labeled}} + \underbrace{\sum_{(\mathbf{x})\sim\tilde{p}_u} \mathcal{U}(\mathbf{x})}_{\text{unlabeled}} \quad (38)$$

1. Discriminative term: trains classifier from labeled data
2. Generative labeled data: infer latent variable z

$$\mathcal{L}(\mathbf{x},y) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x},y)}[\ln p_\theta(\mathbf{x},y,\mathbf{z}) - \ln q_\phi(\mathbf{z}|\mathbf{x},y)] \leq \ln p(\mathbf{x},y)$$

3. Generative unlabeled data: infer label y and latent variable $\mathbf{z}$

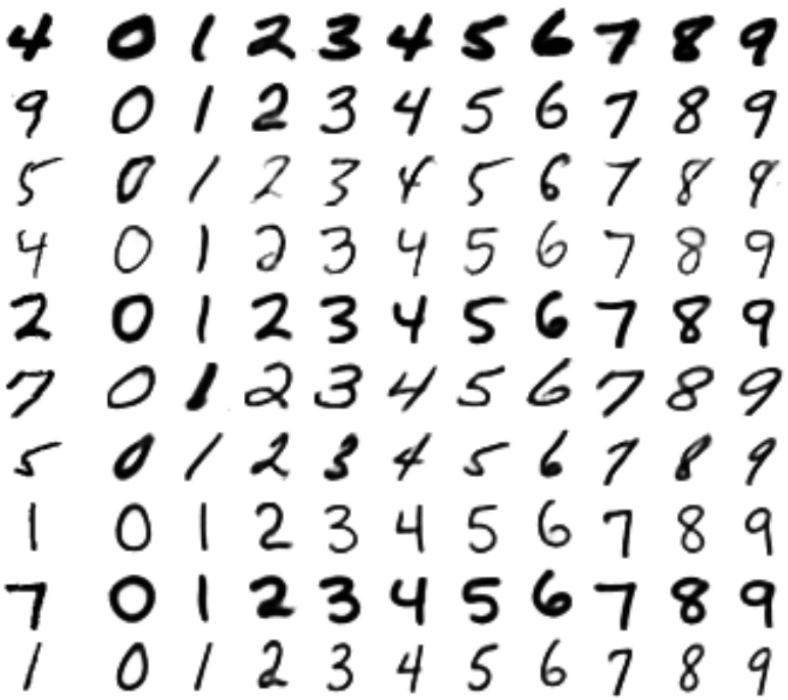
$$\mathcal{U}(\mathbf{x}) = \mathbb{E}_{q_\phi(y,\mathbf{z}|\mathbf{x})}[\ln p_\theta(\mathbf{x},y,\mathbf{z}) - \ln q_\phi(y|\mathbf{x}) - q_\phi(\mathbf{z}|\mathbf{x},y)] \leq \ln p(\mathbf{x})$$

Class-conditional image generation on MNIST

- Fix class label $y \in \{2, 3, 4\}$, vary latent variable $\mathbf{z} \in \mathbb{R}^{\mathbf{2}}$

Cross-class style transfer

- Fix latent variable $\mathbf{z}$, vary class label $y \in \{1, \ldots, 9, 0\}$
- First column: images from the test set, infer $\mathbf{z}$
- Others: images generated for each class using inferred $\mathbf{z}$



Semi-supervised classification on MNIST

- M1+TSVM: use TSVM on learn representation with VAE
- M2: use presented semi-supervised VAE
- M1+M2: use M2 on representation learned by M1

Table 1: Benchmark results of semi-supervised classification on MNIST with few labels.

N	NN	CNN	TSVM	CAE	MTC	AtlasRBF	M1+TSVM	M2	M1+M2
100	25.81	22.98	16.81	13.47	12.03	8.10 ($\pm$ 0.95)	11.82 ($\pm$ 0.25)	11.97 ($\pm$ 1.71)	**3.33** ($\pm$ 0.14)
600	11.44	7.68	6.16	6.3	5.13	–	5.72 ($\pm$ 0.049)	4.94 ($\pm$ 0.13)	**2.59** ($\pm$ 0.05)
1000	10.7	6.45	5.38	4.77	3.64	3.68 ($\pm$ 0.12)	4.24 ($\pm$ 0.07)	3.60 ($\pm$ 0.56)	**2.40** ($\pm$ 0.02)
3000	6.04	3.35	3.45	3.22	2.57	–	3.49 ($\pm$ 0.04)	3.92 ($\pm$ 0.63)	**2.18** ($\pm$ 0.04)

Improving variational autoencoders

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

  - Generally true posterior is not Gaussian: loose bound

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress
  1. More accurate bound for given posterior

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress
  1. More accurate bound for given posterior
  2. Enlarge the family of variational posteriors

Improving variational autoencoders

- ELBO uses KL divergence to bound the data log-likelihood

$$F(\mathbf{x}, \theta, \phi) = \ln p(\mathbf{x}) - D\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\right) \tag{39}$$

  - Generally true posterior is not Gaussian: loose bound
  - Encourages true posterior to match variational factored Gaussian produced by recognition net

- Making progress
  1. More accurate bound for given posterior
  2. Enlarge the family of variational posteriors
  3. Avoid approximate inference altogether

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z})/q_\phi(\mathbf{z}|\mathbf{x})$

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$
3. If the weights are bounded, then $F_k \to \ln p(\mathbf{x})$ as $k \to \infty$

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$
3. If the weights are bounded, then $F_k \to \ln p(\mathbf{x})$ as $k \to \infty$

- Use as objective to train models for $k \approx 10$

Importance weighted autoencoders [Burda et al., 2016]

- Construct tighter lower bound using importance sampling
- Define importance weights $w(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}, \mathbf{z}) / q_\phi(\mathbf{z}|\mathbf{x})$

$$
\begin{aligned}
F_k(\mathbf{x}, \theta, \phi) &= \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\ln \frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&\leq \ln \mathbb{E}_{\mathbf{z}_1, \ldots, \mathbf{z}_k \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\frac{1}{k} \sum_{i=1}^{k} w(\mathbf{x}, \mathbf{z}_i) \right] \\
&= \ln \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [w(\mathbf{x}, \mathbf{z})] \\
&= \ln p(\mathbf{x})
\end{aligned}
$$

1. VAE lower bound recovered for $k = 1$
2. More samples tighten the bound: $F_k \leq F_{k+1} \leq \ln p(\mathbf{x})$
3. If the weights are bounded, then $F_k \to \ln p(\mathbf{x})$ as $k \to \infty$

- Use as objective to train models for $k \approx 10$
- Use as likelihood estimator for (IW-)VAE with $k \approx 10^3$

Training procedure importance weighted autoencoders

- Gradients of importance weighted lower bound

$$\nabla F_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:k} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\sum_{i=1}^{k} \widetilde{w_i} \nabla \big(\ln p(\mathbf{x}, \mathbf{z}_i) - \ln q_\phi(\mathbf{z}_i|\mathbf{x}) \big) \right]$$

Training procedure importance weighted autoencoders

- Gradients of importance weighted lower bound

$$\nabla F_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:k} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\sum_{i=1}^{k} \widetilde{w_i} \nabla \big(\ln p(\mathbf{x}, \mathbf{z}_i) - \ln q_\phi(\mathbf{z}_i|\mathbf{x}) \big) \right]$$

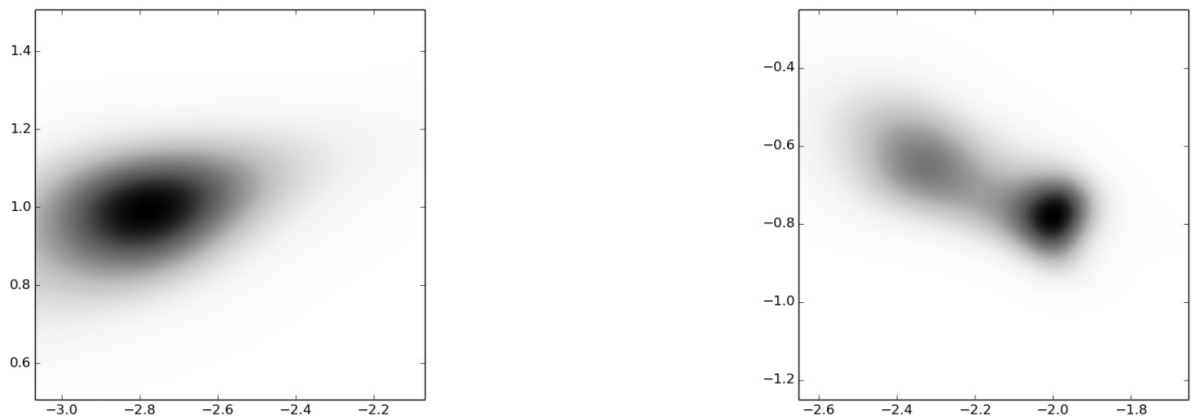
- Similar to VAE, but samples weighted w.r.t. true posterior
  - Normalized importance weights $\widetilde{w_i} = w(\mathbf{x}, \mathbf{z}_i) / \sum_{j=1}^{k} w(\mathbf{x}, \mathbf{z}_j)$

Training procedure importance weighted autoencoders

- Gradients of importance weighted lower bound

$$\nabla F_k(\mathbf{x}) = \mathbb{E}_{\mathbf{z}_{1:k} \sim q_\phi(\mathbf{z}|\mathbf{x})} \left[\sum_{i=1}^{k} \widetilde{w_i} \nabla \big(\ln p(\mathbf{x}, \mathbf{z}_i) - \ln q_\phi(\mathbf{z}_i|\mathbf{x}) \big) \right]$$

- Similar to VAE, but samples weighted w.r.t. true posterior
  - Normalized importance weights $\widetilde{w_i} = w(\mathbf{x}, \mathbf{z}_i) / \sum_{j=1}^{k} w(\mathbf{x}, \mathbf{z}_j)$
- Allows for more accurate models with complex posteriors



True posterior $p(\mathbf{z}|\mathbf{x})$ VAE (left) and IW-VAE (right)

Variational inference with normalizing flows

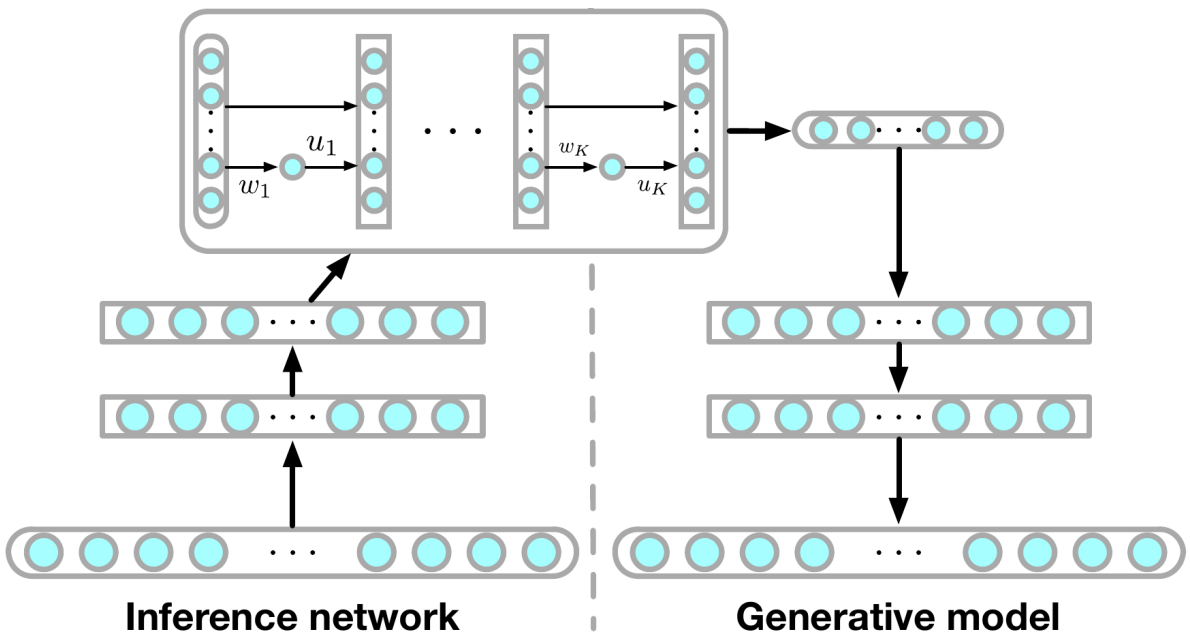
[Rezende and Mohamed, 2015]

- Variational inference (in VAE) uses limited class of posteriors
  - For example, Gaussian with diagonal covariance
  - Optimizing loose bound on data log-likelihood

Variational inference with normalizing flows

[Rezende and Mohamed, 2015]

- Variational inference (in VAE) uses limited class of posteriors
  - For example, Gaussian with diagonal covariance
  - Optimizing loose bound on data log-likelihood
- Improve posterior with series of invertible transformations



Normalizing flows

- Let density "flow" through set of invertible transformations

$$\mathbf{z}_K = f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z}_0),$$

$$\ln q_K(\mathbf{z}_K) = \ln q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_k} \right|$$

Normalizing flows

- Let density "flow" through set of invertible transformations

$$\mathbf{z}_K = f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z}_0),$$

$$\ln q_K(\mathbf{z}_K) = \ln q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_k} \right|$$

- Linear-time determinant for planar and radial flows

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u} h(\mathbf{w}^\top \mathbf{z} + b)$$

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)\,(\mathbf{z} - \mathbf{z_0})$$

Normalizing flows

- Let density "flow" through set of invertible transformations

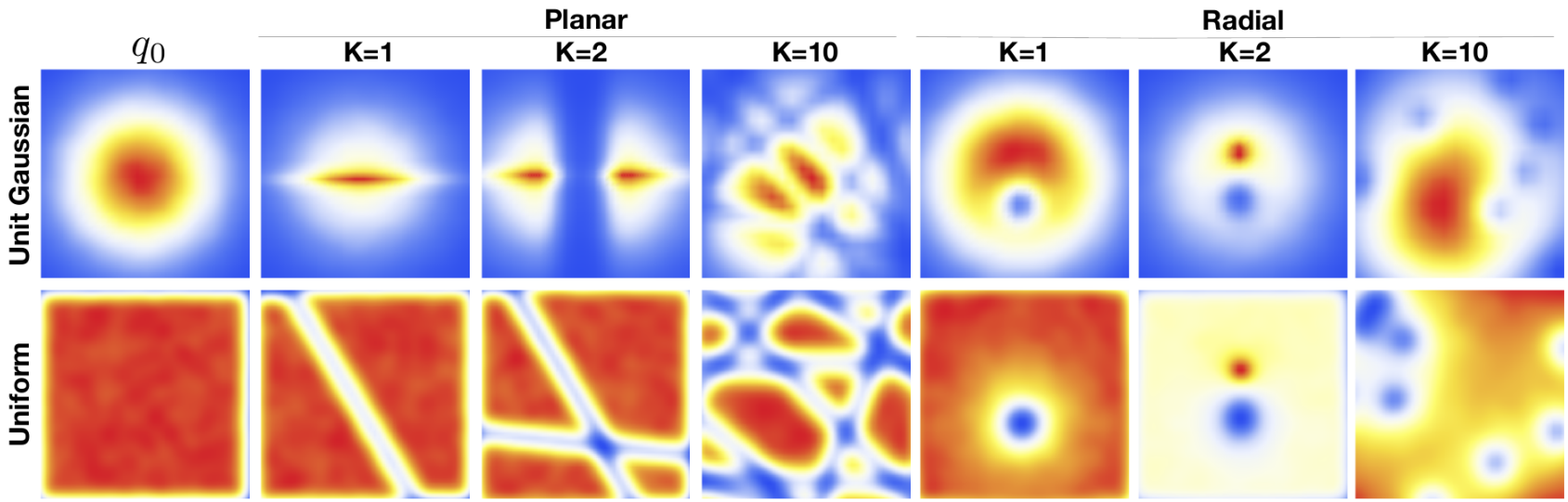
$$\mathbf{z}_K = f_K \circ \cdots \circ f_2 \circ f_1(\mathbf{z}_0),$$

$$\ln q_K(\mathbf{z}_K) = \ln q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \ln \left| \det \frac{\partial f_k}{\partial \mathbf{z}_k} \right|$$

- Linear-time determinant for planar and radial flows

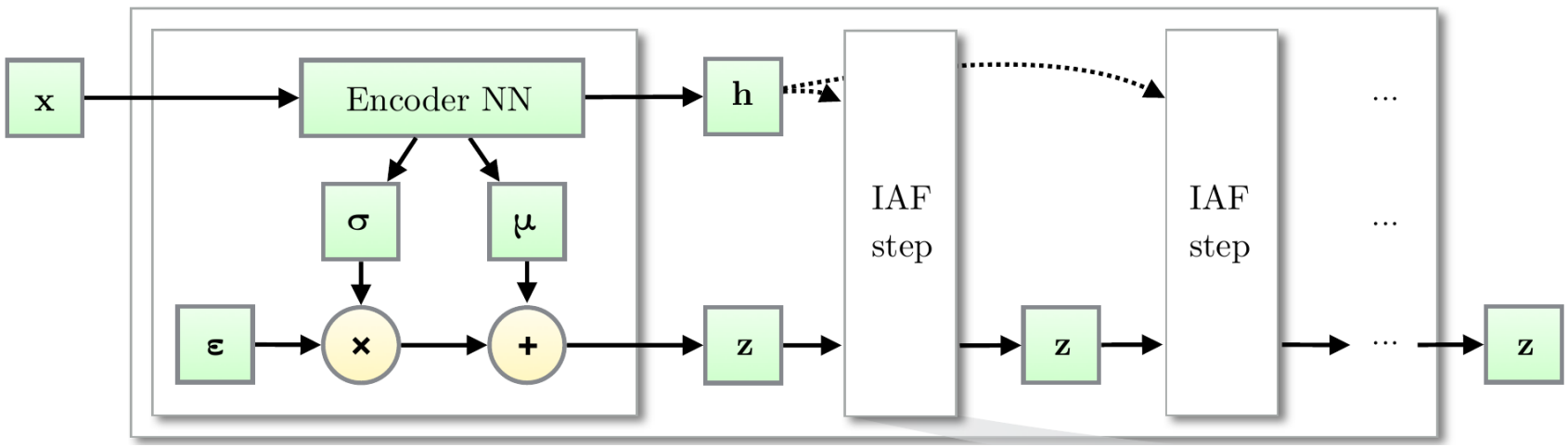
$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u} h(\mathbf{w}^\top \mathbf{z} + b)$$

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)\,(\mathbf{z} - \mathbf{z_0})$$



Autoregressive flow [Kingma et al., 2016]

- Restictive flows in [Rezende and Mohamed, 2015]
  - Planar flow similar to MLP with single hidden unit
- Use autoregressive transformations in flow
  - Rich and tractable class of transformations
  - Fewer transformations needed



Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_t$$

Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_t$$

- Autoregressive computation of affine parameters

$$\boldsymbol{\mu}_{t,i+1} = f(\mathbf{z}_{t,1:i}) \qquad \boldsymbol{\sigma}_{t,i+1} = g(\mathbf{z}_{t,1:i})$$

Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_t$$

- Autoregressive computation of affine parameters

$$\boldsymbol{\mu}_{t,i+1} = f(\mathbf{z}_{t,1:i}) \qquad \boldsymbol{\sigma}_{t,i+1} = g(\mathbf{z}_{t,1:i})$$

- Triangular Jacobian, log-determinant $\sum_{i=1}^{D} \log \sigma_{t,i}$

Autoregressive flow [Kingma et al., 2016]

- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_t$$

- Autoregressive computation of affine parameters

$$\boldsymbol{\mu}_{t,i+1} = f(\mathbf{z}_{t,1:i}) \qquad \boldsymbol{\sigma}_{t,i+1} = g(\mathbf{z}_{t,1:i})$$

- Triangular Jacobian, log-determinant $\sum_{i=1}^{D} \log \sigma_{t,i}$

- Parallel computations of $\mathbf{z}_t, \boldsymbol{\mu}_t, \boldsymbol{\sigma}_t$ across dimensions

Autoregressive flow [Kingma et al., 2016]

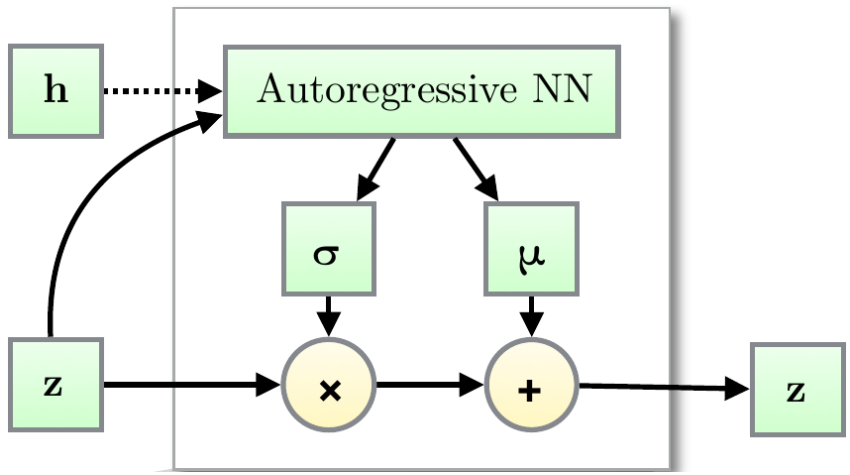
- Class of affine transformations with respect to $\mathbf{z}$

$$\mathbf{z}_{t+1} = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_t$$

- Autoregressive computation of affine parameters

$$\boldsymbol{\mu}_{t,i+1} = f(\mathbf{z}_{t,1:i}) \qquad \boldsymbol{\sigma}_{t,i+1} = g(\mathbf{z}_{t,1:i})$$

- Triangular Jacobian, log-determinant $\sum_{i=1}^{D} \log \sigma_{t,i}$
- Parallel computations of $\mathbf{z}_t, \boldsymbol{\mu}_t, \boldsymbol{\sigma}_t$ across dimensions
- Free to chose form of autoregressive dependency



Non-volume preserving (NVP) transformation [Dinh et al., 2017]

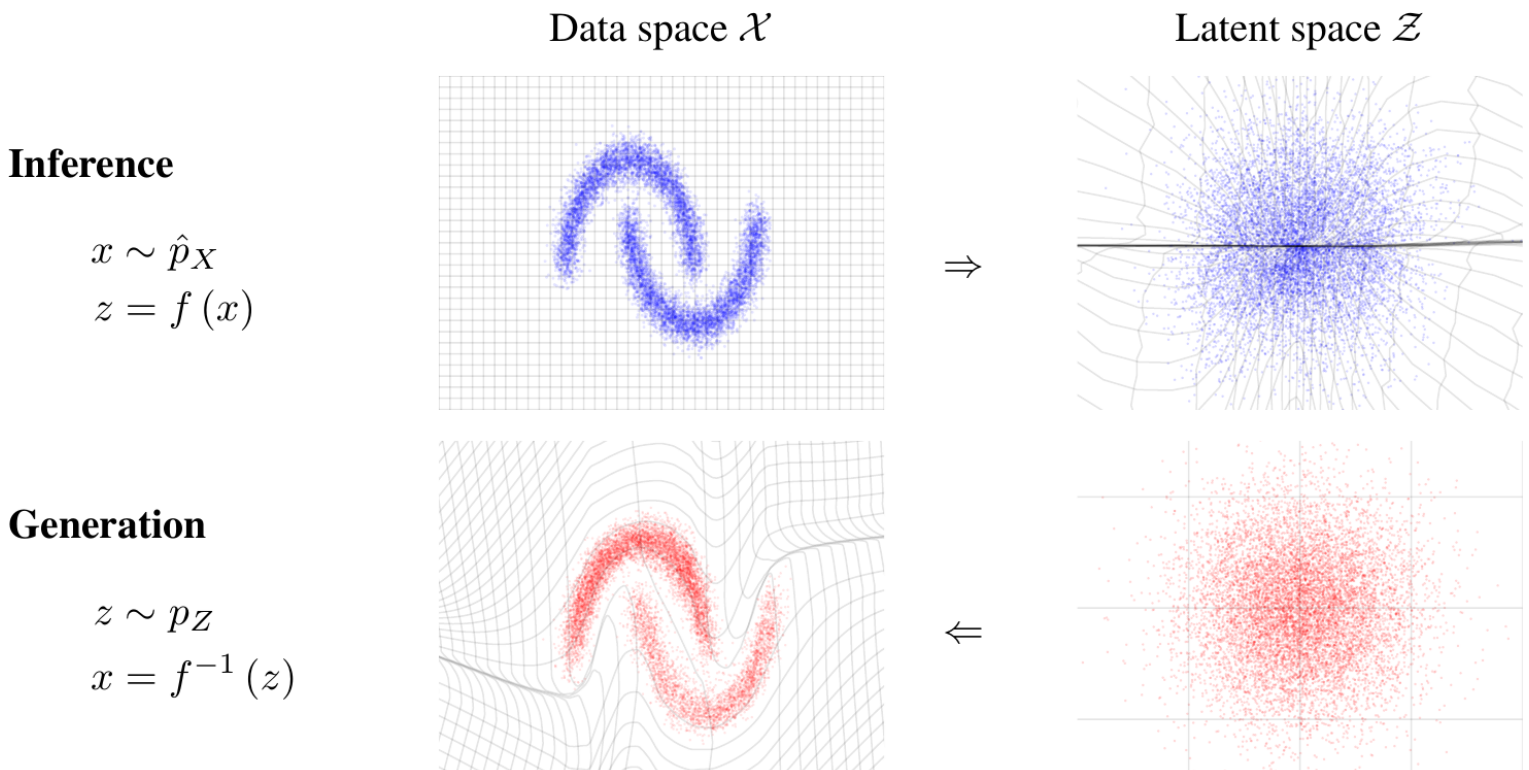
- Learn invertible function from latent to data space

Non-volume preserving (NVP) transformation [Dinh et al., 2017]

- Learn invertible function from latent to data space
- Latent and data space have same dimensionality

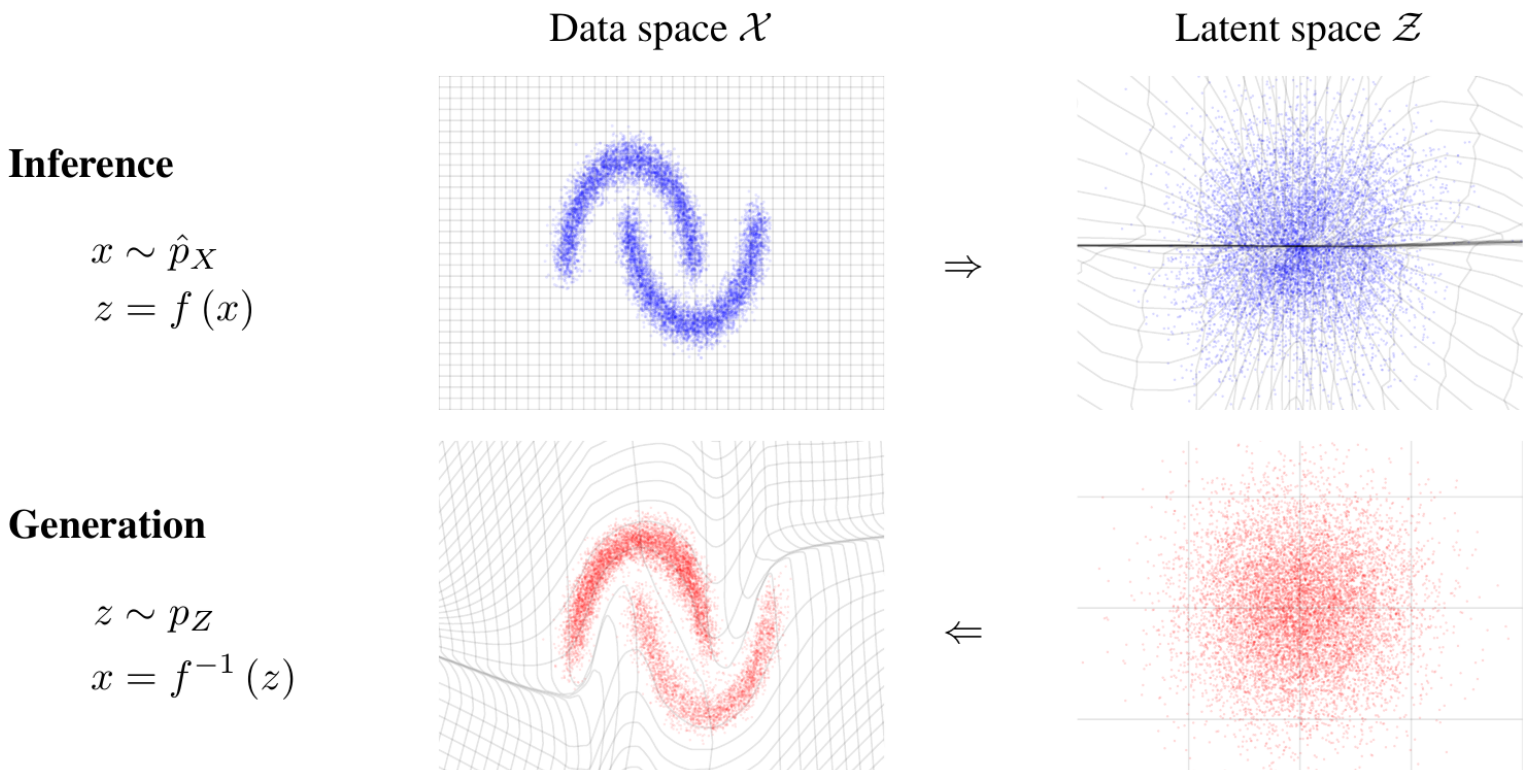
Non-volume preserving (NVP) transformation [Dinh et al., 2017]

- Learn invertible function from latent to data space
- Latent and data space have same dimensionality
- Unit Gaussian prior on latent variables



Non-volume preserving (NVP) transformation [Dinh et al., 2017]

- Learn invertible function from latent to data space
- Latent and data space have same dimensionality
- Unit Gaussian prior on latent variables
- Tractable sampling and exact inference



Change of variable formula for invertible function

- Ensuring efficient computation of $\mathbf{y} = f(\mathbf{x})$ and determinant

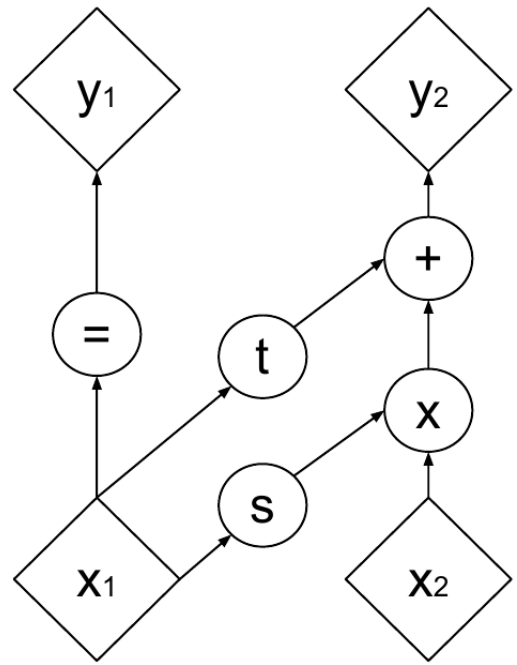
$$p_X(\mathbf{x}) = p_Y(f(\mathbf{x})) \times \left| \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} \right) \right|$$

Change of variable formula for invertible function

- Ensuring efficient computation of $\mathbf{y} = f(\mathbf{x})$ and determinant

$$p_X(\mathbf{x}) = p_Y(f(\mathbf{x})) \times \left| \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} \right) \right|$$

1. Partition variables in two groups

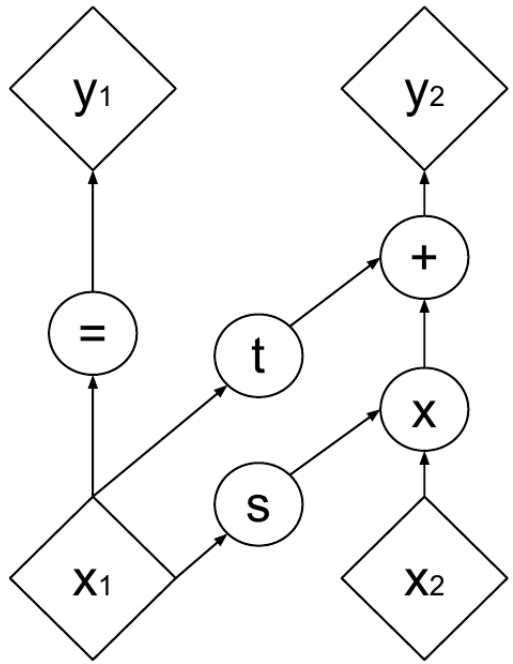


Change of variable formula for invertible function

- Ensuring efficient computation of $\mathbf{y} = f(\mathbf{x})$ and determinant

$$p_X(\mathbf{x}) = p_Y(f(\mathbf{x})) \times \left| \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} \right) \right|$$

1. Partition variables in two groups
2. Keep one group unchanged



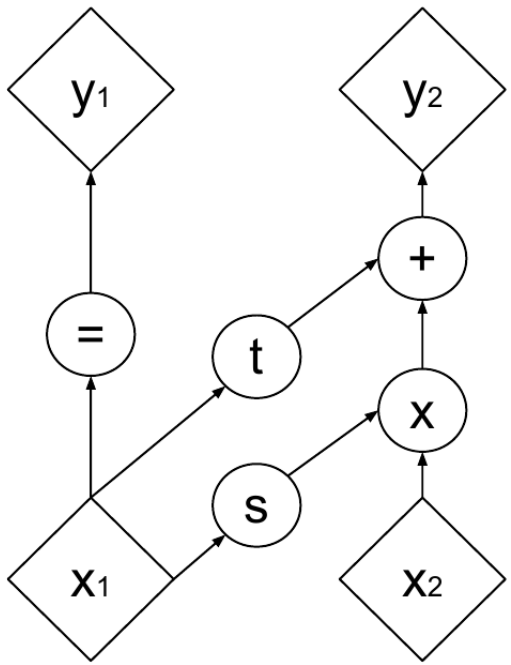
Change of variable formula for invertible function

- Ensuring efficient computation of $\mathbf{y} = f(\mathbf{x})$ and determinant

$$p_X(\mathbf{x}) = p_Y(f(\mathbf{x})) \times \left| \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} \right) \right|$$

1. Partition variables in two groups
2. Keep one group unchanged
3. Let one group transform the other via translation ans scaling

$$\mathbf{y}_1 = \mathbf{x}_1$$
$$\mathbf{y}_2 = t(\mathbf{x}_1) + \mathbf{x}_2 \odot \exp(s(\mathbf{x}_1))$$

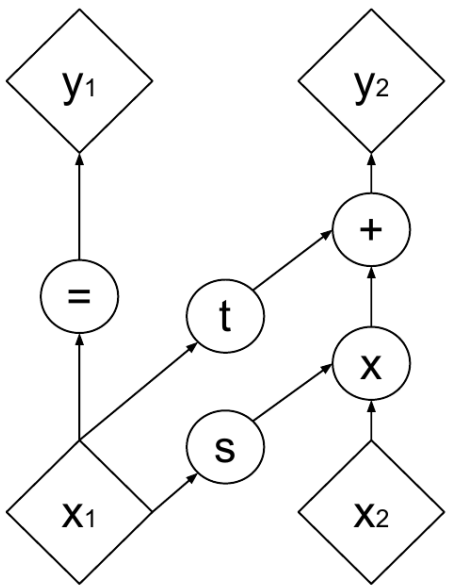


Properties: Efficient inversion

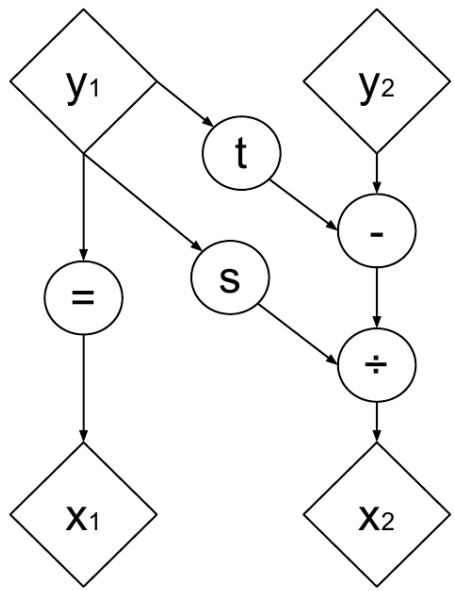
- Inverse transformation

$$\mathbf{x}_1 = \mathbf{y}_1 \tag{40}$$

$$\mathbf{x}_2 = (\mathbf{y}_2 - t(\mathbf{x}_1)) \odot \exp(-s(\mathbf{x}_1)) \tag{41}$$



(a) Forward propagation



(b) Inverse propagation

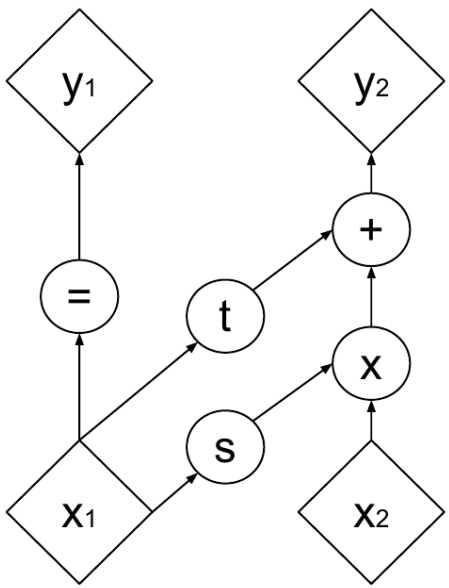
Properties: Efficient inversion

- Inverse transformation

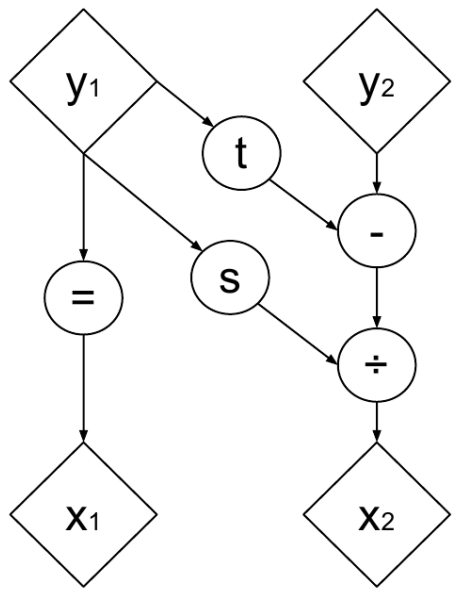
$$\mathbf{x}_1 = \mathbf{y}_1 \tag{40}$$

$$\mathbf{x}_2 = (\mathbf{y}_2 - t(\mathbf{x}_1)) \odot \exp(-s(\mathbf{x}_1)) \tag{41}$$

- No need to invert $s(\cdot)$ and $t(\cdot)$
- Can use complex non-invertible functions, *e.g.* deep CNN



(a) Forward propagation



(b) Inverse propagation

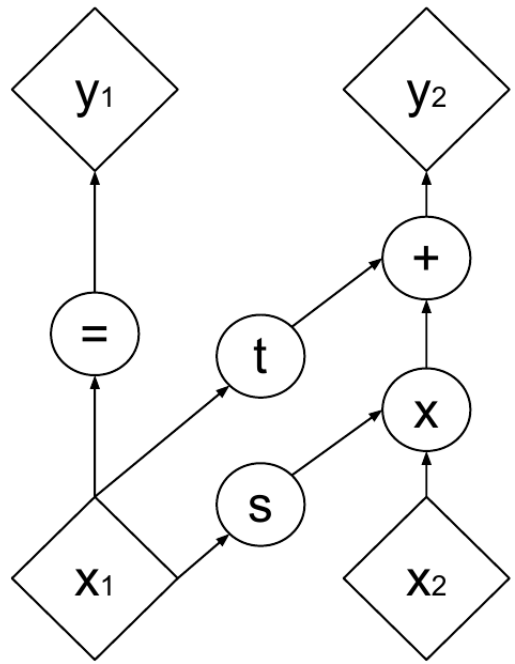
Properties: Efficient determinant computation

- Triangular structure of Jacobian

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} = \begin{bmatrix} I_d & 0 \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1^\top} & \text{diag}(\exp(s(\mathbf{x}_1))) \end{bmatrix}$$

- Determinant given by product of Jacobian's diagonal terms

$$\ln \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} \right) = \mathbf{1}^\top s(\mathbf{x}_1)$$



Properties: Efficient determinant computation

- Triangular structure of Jacobian

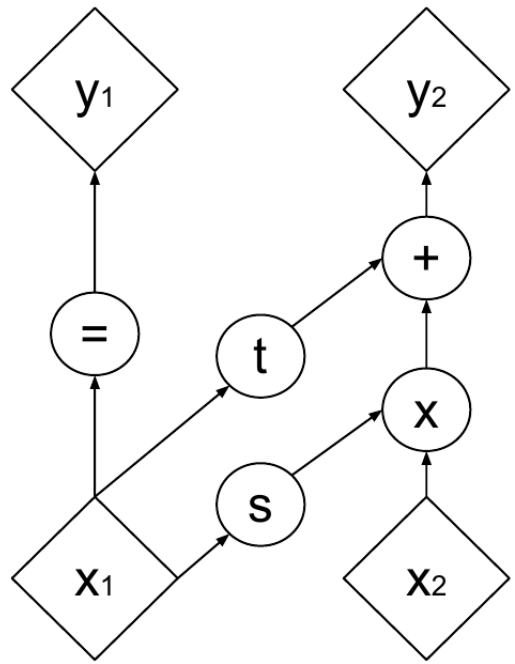
$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} = \begin{bmatrix} I_d & 0 \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1^\top} & \text{diag}(\exp(s(\mathbf{x}_1))) \end{bmatrix}$$

- Determinant given by product of Jacobian's diagonal terms

$$\ln \det \left(\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} \right) = \mathbf{1}^\top s(\mathbf{x}_1)$$

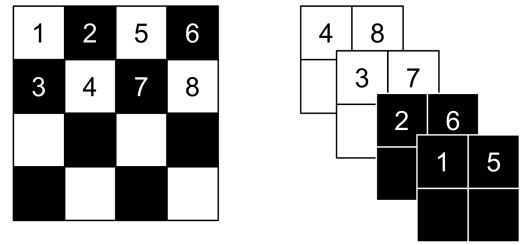
- Log-likelihood easily computed, optimize using stochastic gradient decent

$$\ln p_X(\mathbf{x}) = \ln p_Y(f(\mathbf{x})) + \mathbf{1}^\top s(\mathbf{x}_1)$$



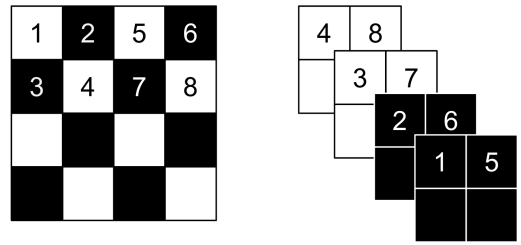
Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask



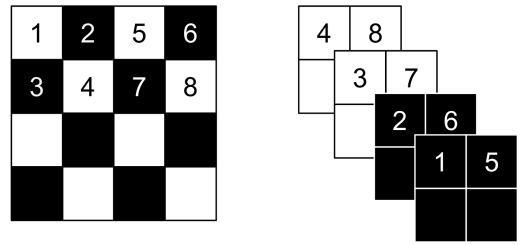
Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output

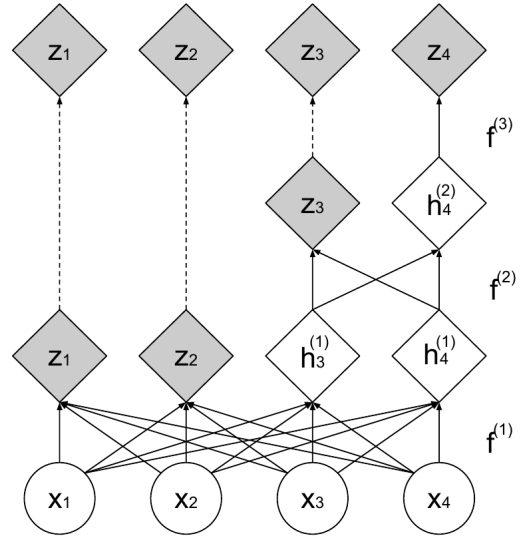


Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output

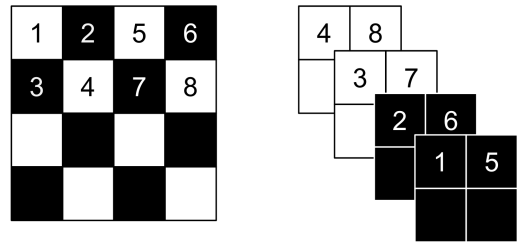


- Stack multiple layers of transformations

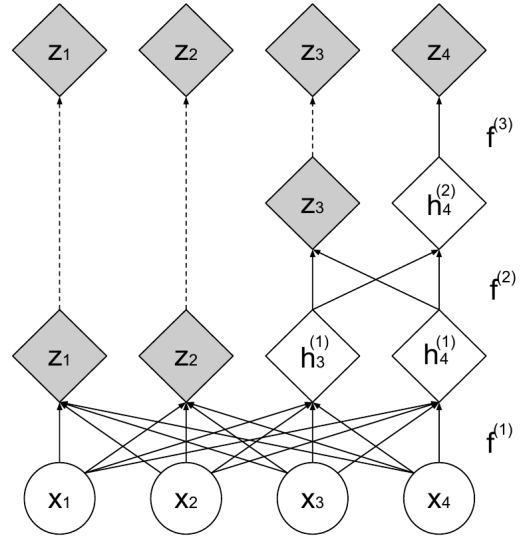


Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output

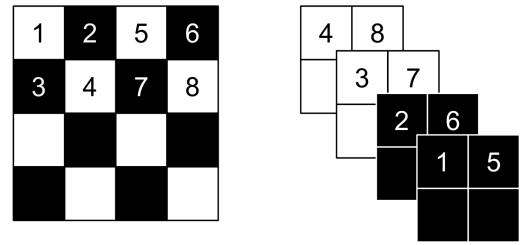


- Stack multiple layers of transformations
  - Alternating the masking patterns

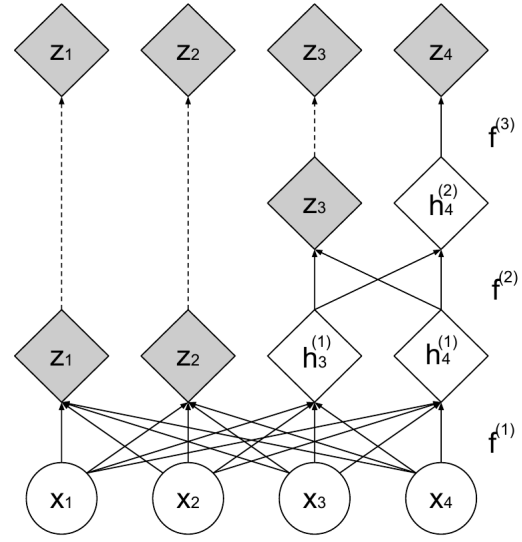


Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output

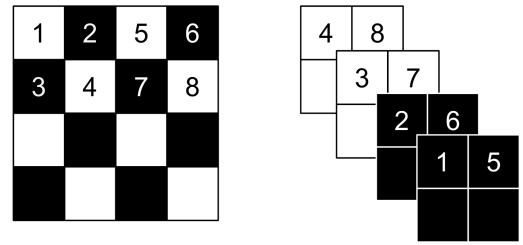


- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants



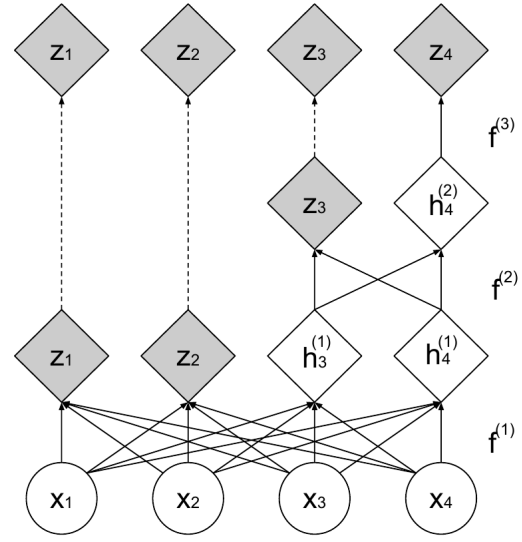
Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output



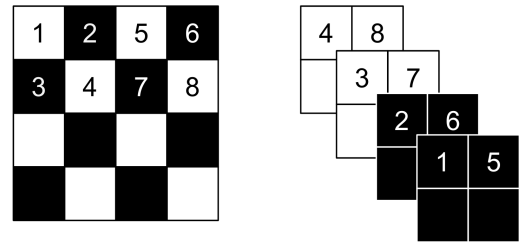
- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants

- Feature abstraction hierarchy



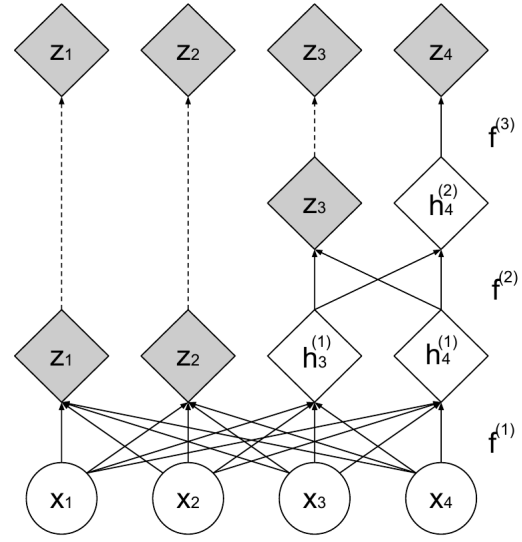
Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output



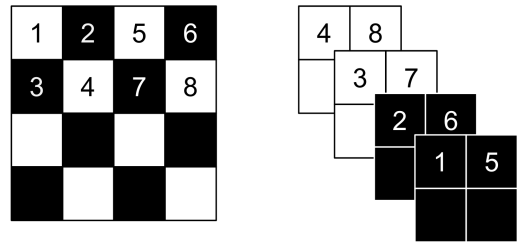
- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants

- Feature abstraction hierarchy
  - Squeeze $2n \times 2n \times c$ map into $n \times n \times 4c$



Implementation

- Variable partitioning schemes
  - Checkerboard mask, channel-wise mask
  - Input to CNN masked, masked application of CNN output



- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants

- Feature abstraction hierarchy
  - Squeeze $2n \times 2n \times c$ map into $n \times n \times 4c$
  - Factor out half of latent variables at regular intervals

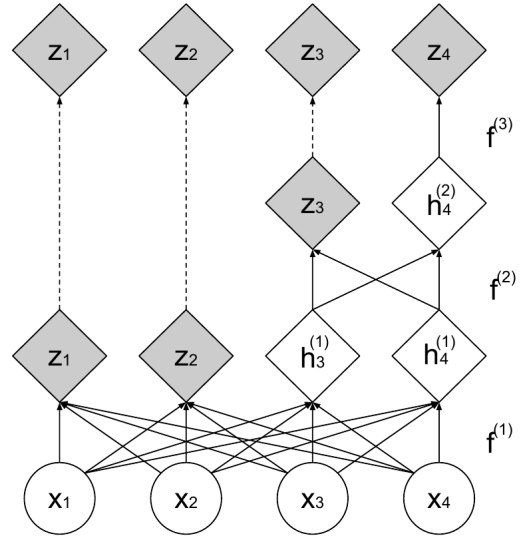


Illustration multi-scale feature hierarchy

- Images obtained after re-sampling part of latent variables

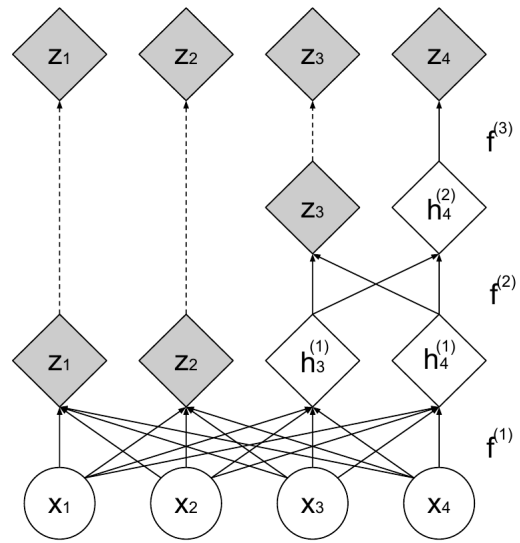


Illustration multi-scale feature hierarchy

- Images obtained after re-sampling part of latent variables
- From left to right: original, keeping $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$

ImageNet 64×64

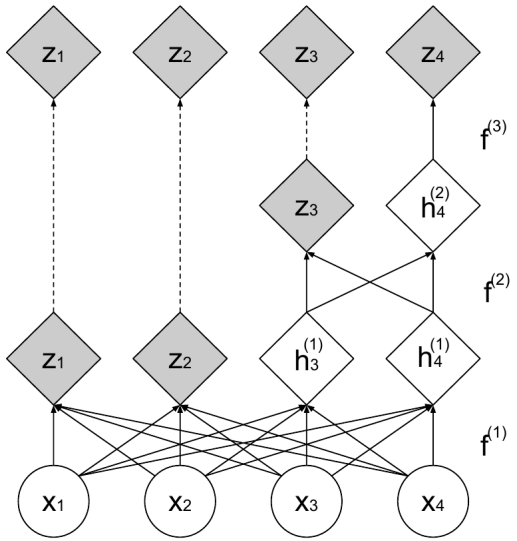
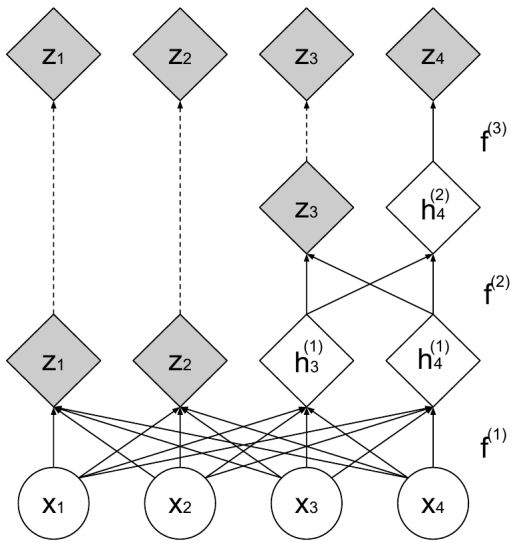


Illustration multi-scale feature hierarchy

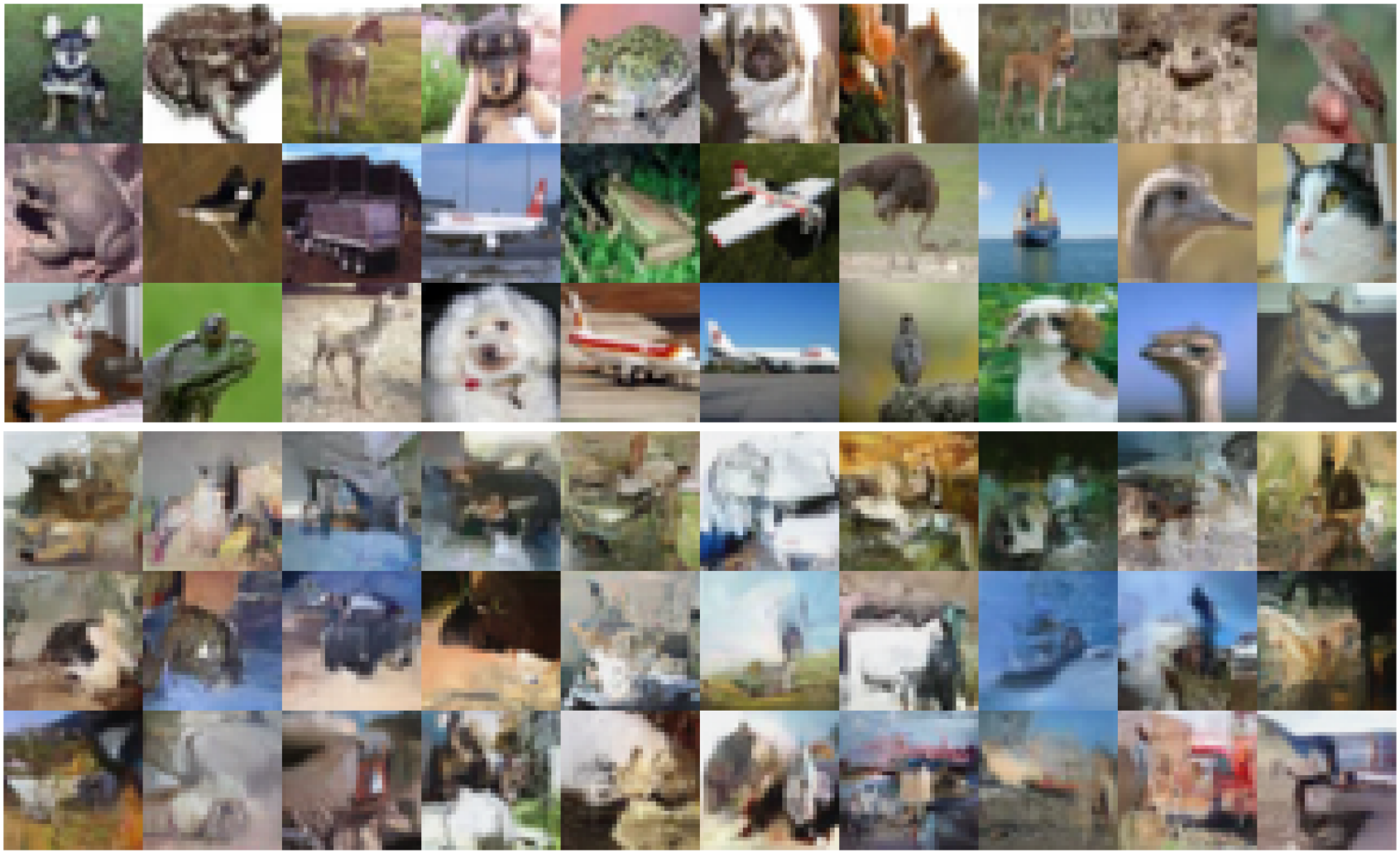
- Images obtained after re-sampling part of latent variables
- From left to right: original, keeping $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$

ImageNet 64×64

CelebA 64×64



Images & Samples NVP: CIFAR10 Dataset 32×32



Part III

Autoregressive density estimation

Autoregressive modeling

- Consider generic factorization of joint probability

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{42}$$

with $\mathbf{x}_{<i} = \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$

Autoregressive modeling

- Consider generic factorization of joint probability

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{42}$$

  with $\mathbf{x}_{<i} = \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$

- Use (deep) neural net to model dependencies in $p(x_i|\mathbf{x}_{<i})$

Autoregressive modeling

- Consider generic factorization of joint probability

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{42}$$

  with $\mathbf{x}_{<i} = \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$

- Use (deep) neural net to model dependencies in $p(x_i|\mathbf{x}_{<i})$
- Tractable exact likelihood computations
  - No complex integral over latent variables in likelihood

Autoregressive modeling

- Consider generic factorization of joint probability

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{42}$$

  with $\mathbf{x}_{<i} = \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$

- Use (deep) neural net to model dependencies in $p(x_i|\mathbf{x}_{<i})$
- Tractable exact likelihood computations
  - No complex integral over latent variables in likelihood
- Slow sequential sampling process
  - Cannot rely on latent variables to couple pixels

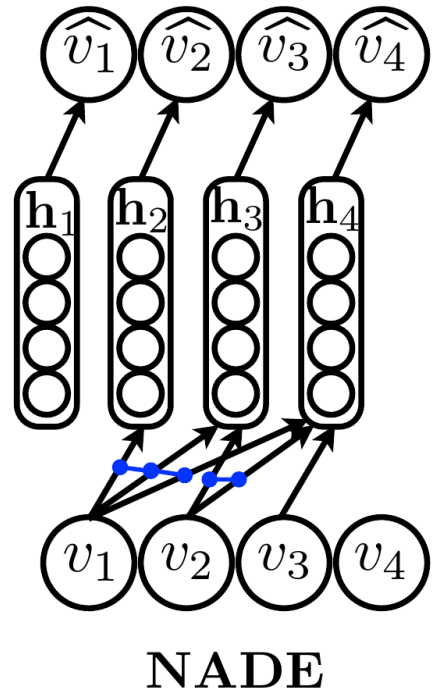
Neural Autoregressive Distribution Estimator (NADE)

[Larochelle and Murray, 2011]

Neural Autoregressive Distribution Estimator (NADE)

[Larochelle and Murray, 2011]

- Assume ordering on binary variables
- Estimate conditional distributions using 2-layer sigmoid network



NADE

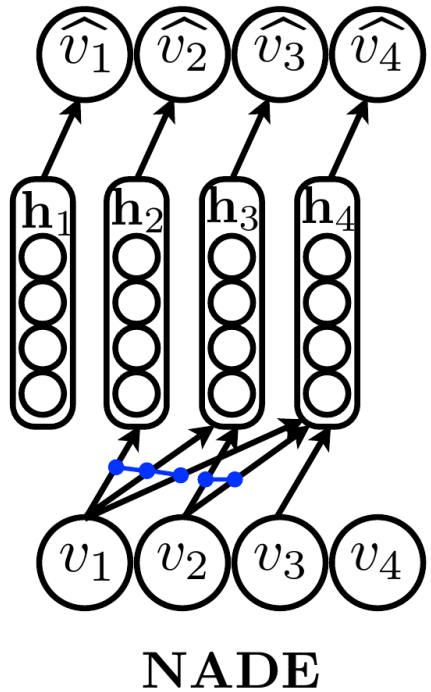
Neural Autoregressive Distribution Estimator (NADE)

[Larochelle and Murray, 2011]

- Assume ordering on binary variables
- Estimate conditional distributions using 2-layer sigmoid network

$$p(v_i = 1|\mathbf{v}_{<i}) = \sigma\left(b_i + \mathbf{a}_i^\top \mathbf{h}_i\right),$$

$$\mathbf{h}_i = \sigma\left(\mathbf{c} + \sum_{j<i} v_j \mathbf{w}_j\right)$$



NADE

Neural Autoregressive Distribution Estimator (NADE)

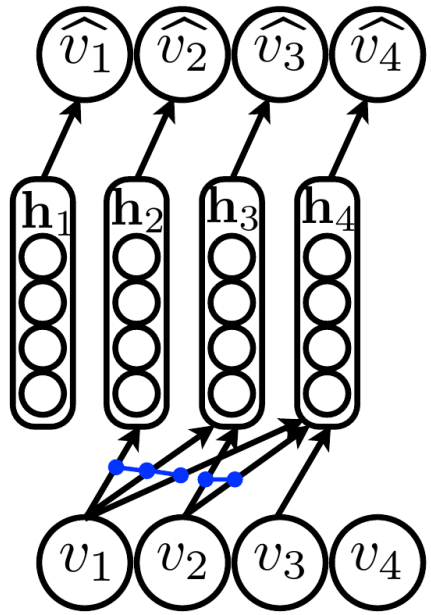
[Larochelle and Murray, 2011]

- Assume ordering on binary variables
- Estimate conditional distributions using 2-layer sigmoid network

$$p(v_i = 1|\mathbf{v}_{<i}) = \sigma\left(b_i + \mathbf{a}_i^\top \mathbf{h}_i\right),$$

$$\mathbf{h}_i = \sigma\left(\mathbf{c} + \sum_{j<i} v_j \mathbf{w}_j\right)$$

- History $\mathbf{v}_{<i}$ compressed in $\mathbf{h}_i$



NADE

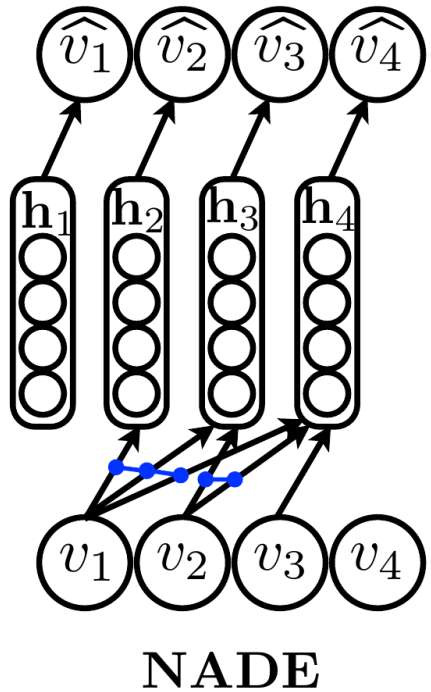
Neural Autoregressive Distribution Estimator (NADE)

[Larochelle and Murray, 2011]

- Assume ordering on binary variables
- Estimate conditional distributions using 2-layer sigmoid network

$$p(v_i = 1|\mathbf{v}_{<i}) = \sigma\left(b_i + \mathbf{a}_i^\top \mathbf{h}_i\right),$$

$$\mathbf{h}_i = \sigma\left(\mathbf{c} + \sum_{j<i} v_j \mathbf{w}_j\right)$$



NADE

- History $\mathbf{v}_{<i}$ compressed in $\mathbf{h}_i$
- Input-to-hidden parameters $\mathbf{w}_j$ shared

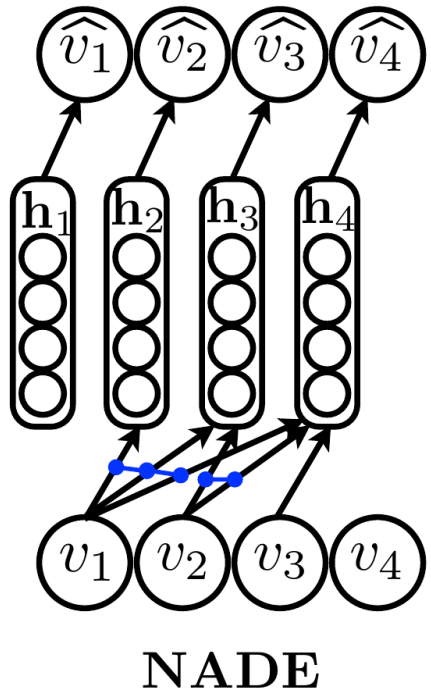
Neural Autoregressive Distribution Estimator (NADE)

[Larochelle and Murray, 2011]

- Assume ordering on binary variables
- Estimate conditional distributions using 2-layer sigmoid network

$$p(v_i = 1|\mathbf{v}_{<i}) = \sigma\left(b_i + \mathbf{a}_i^\top \mathbf{h}_i\right),$$

$$\mathbf{h}_i = \sigma\left(\mathbf{c} + \sum_{j<i} v_j \mathbf{w}_j\right)$$



NADE

- History $\mathbf{v}_{<i}$ compressed in $\mathbf{h}_i$
- Input-to-hidden parameters $\mathbf{w}_j$ shared
- Hidden-to-output parameters $\mathbf{a}_i$ not shared

NADE scales as $O(DH)$

- Nr. of parameters linear in data and hidden dimension

NADE scales as $O(DH)$

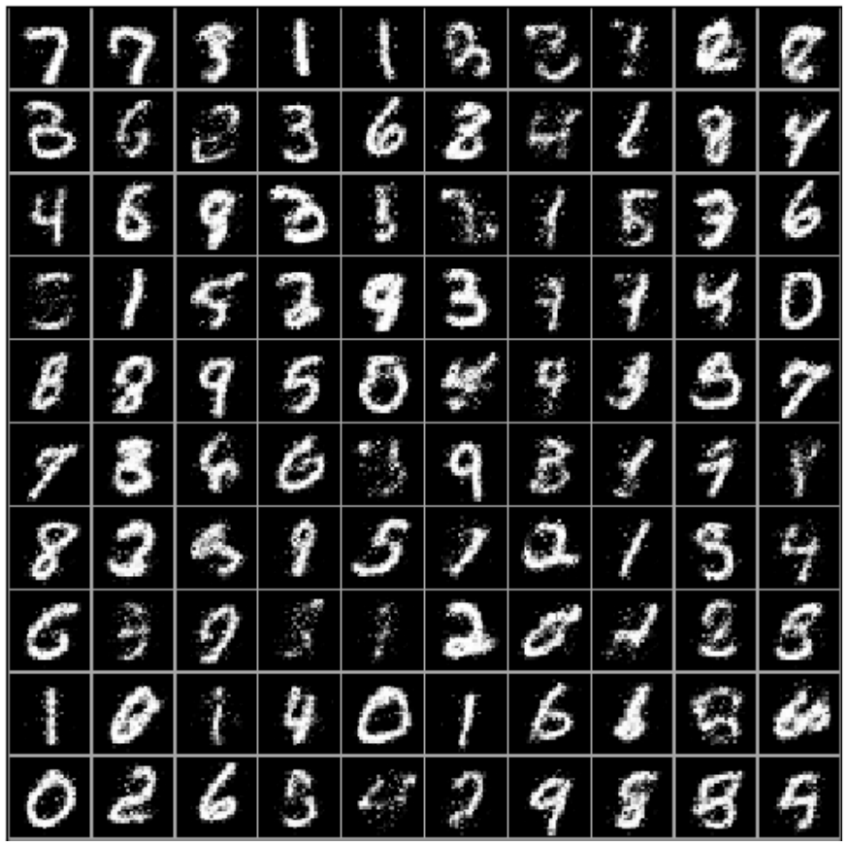
- Nr. of parameters linear in data and hidden dimension
- Computation of $\ln p(\mathbf{v})$ and gradient also linear

NADE scales as $O(DH)$

- Nr. of parameters linear in data and hidden dimension
- Computation of $\ln p(\mathbf{v})$ and gradient also linear
- Sequential sampling from $p(\mathbf{v})$
  - Incremental computation of history vector $\tilde{\mathbf{h}}_{i+1} = \tilde{\mathbf{h}}_i + v_i \mathbf{w}_i$
  - Compute $p(v_i|\mathbf{v}_{<i})$ using $\tilde{\mathbf{h}}_i$, and sample v_i

NADE scales as $O(DH)$

- Nr. of parameters linear in data and hidden dimension
- Computation of $\ln p(\mathbf{v})$ and gradient also linear
- Sequential sampling from $p(\mathbf{v})$
  - Incremental computation of history vector $\tilde{\mathbf{h}}_{i+1} = \tilde{\mathbf{h}}_i + v_i \mathbf{w}_i$
  - Compute $p(v_i|\mathbf{v}_{<i})$ using $\tilde{\mathbf{h}}_i$, and sample v_i



Samples from NADE trained on MNIST digits

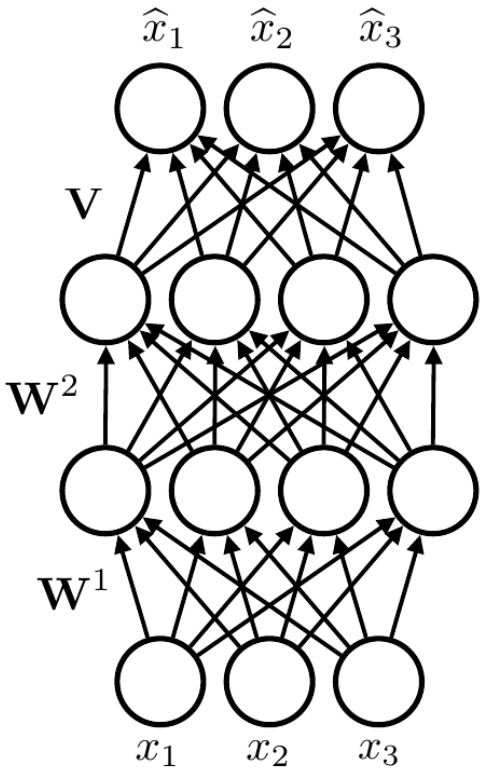
Masked Autoencoder for Distribution Estimation (MADE)

[Germain et al., 2015]

Masked Autoencoder for Distribution Estimation (MADE)

[Germain et al., 2015]

- Adapt autoencoders for distribution estimation

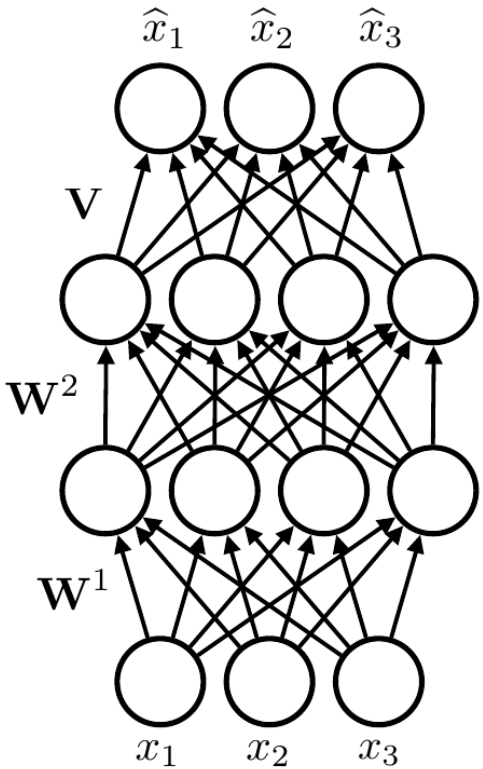


Autoencoder

Masked Autoencoder for Distribution Estimation (MADE)

[Germain et al., 2015]

- Adapt autoencoders for distribution estimation
- Output conditional distributions $p(x_i|\mathbf{x}_{<i})$

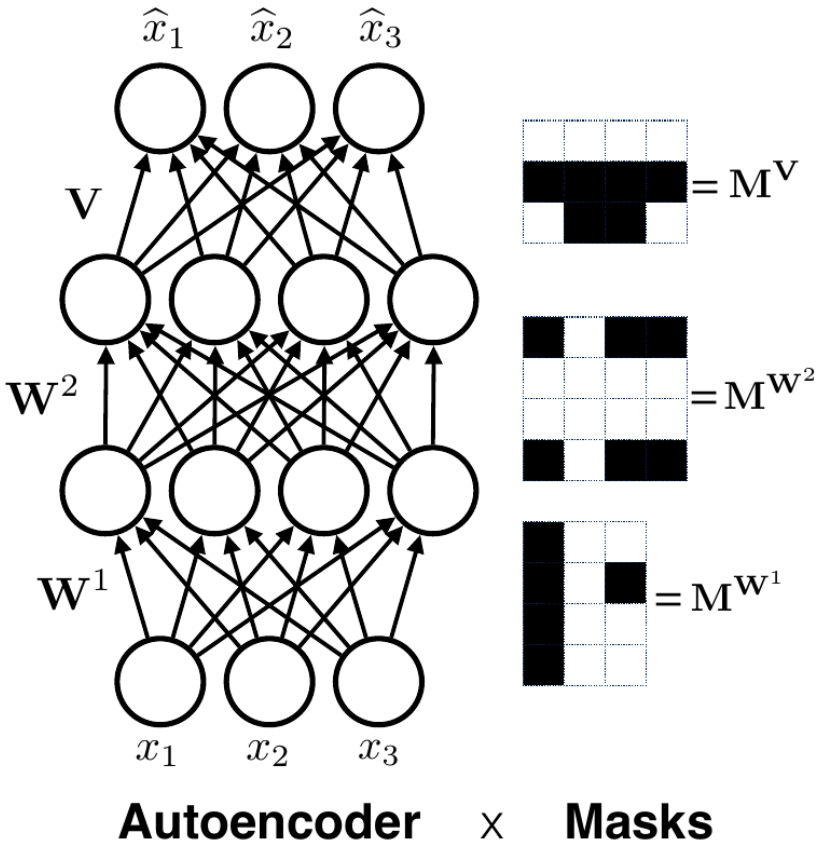


Autoencoder

Masked Autoencoder for Distribution Estimation (MADE)

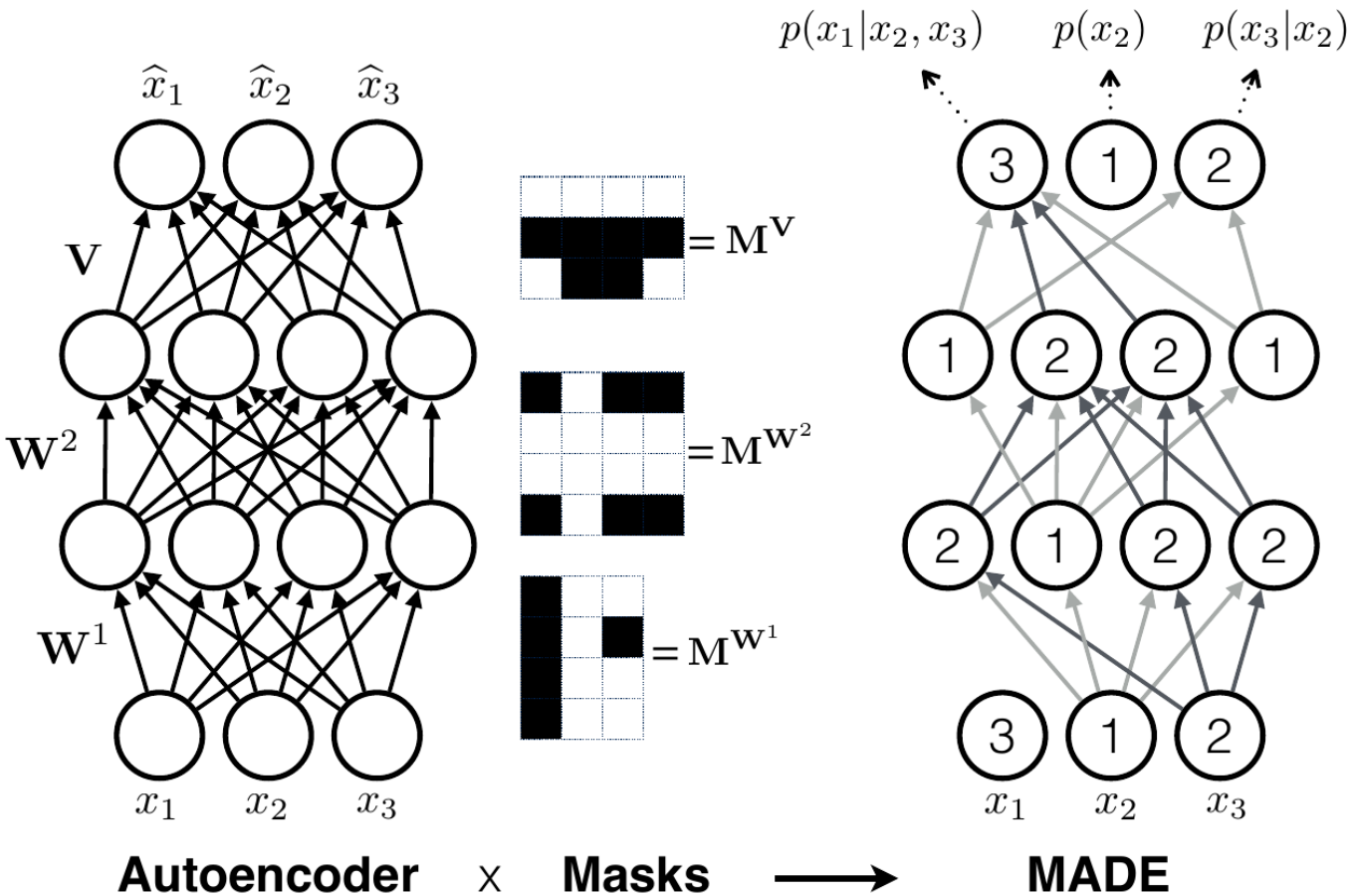
[Germain et al., 2015]

- Adapt autoencoders for distribution estimation
- Output conditional distributions $p(x_i|\mathbf{x}_{<i})$
- Mask connections to ensure proper conditionals



Masked Autoencoder for Distribution Estimation (MADE)

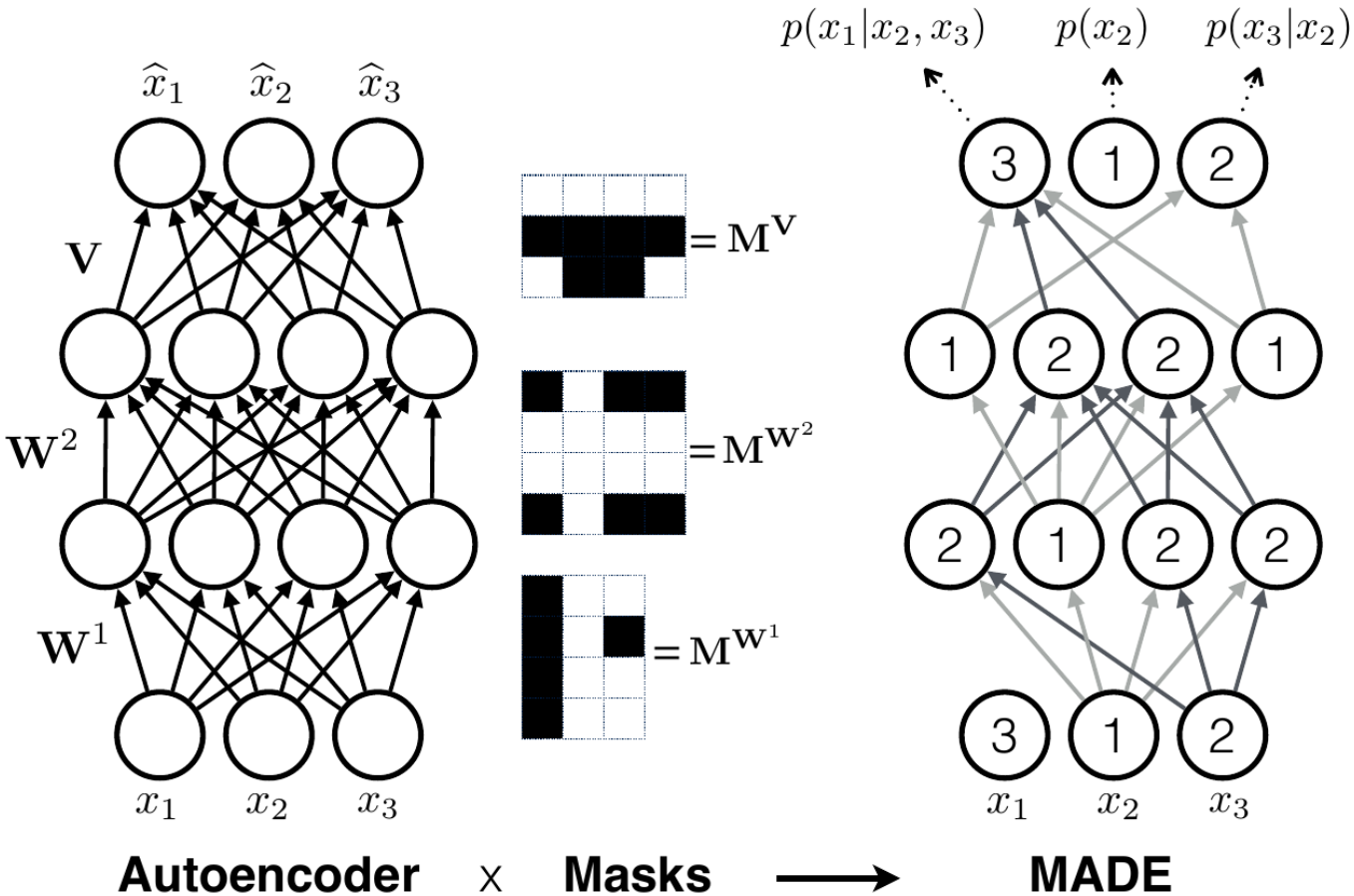
- Set max. input index $1 \leq m(k) < D$ for each hidden node
  - Mask connections to ensure consistency of $m(k)$



Masked Autoencoder for Distribution Estimation (MADE)

- Set max. input index $1 \leq m(k) < D$ for each hidden node
  - Mask connections to ensure consistency of $m(k)$

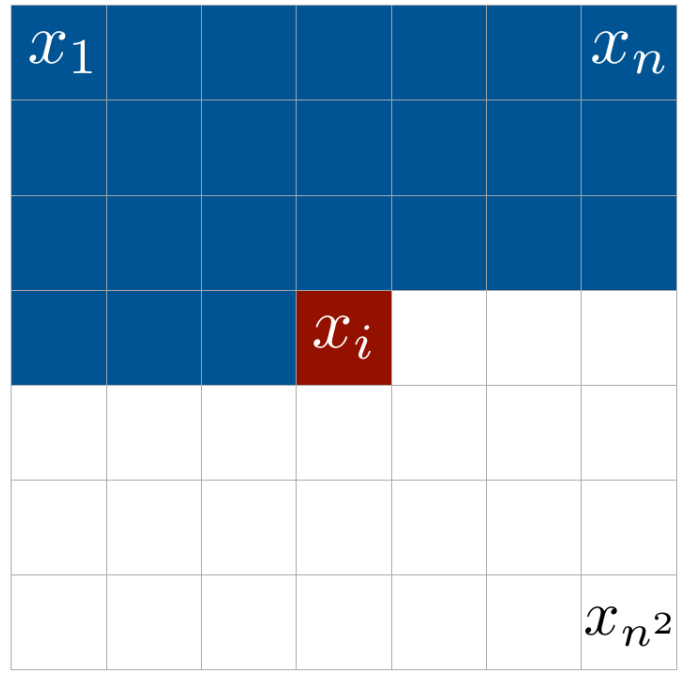
- Vary ordering and connectivity during training
  - Ensemble of models sharing parameters on connections



Pixel Recurrent/Convolutional Neural Networks

[Oord et al., 2016b]

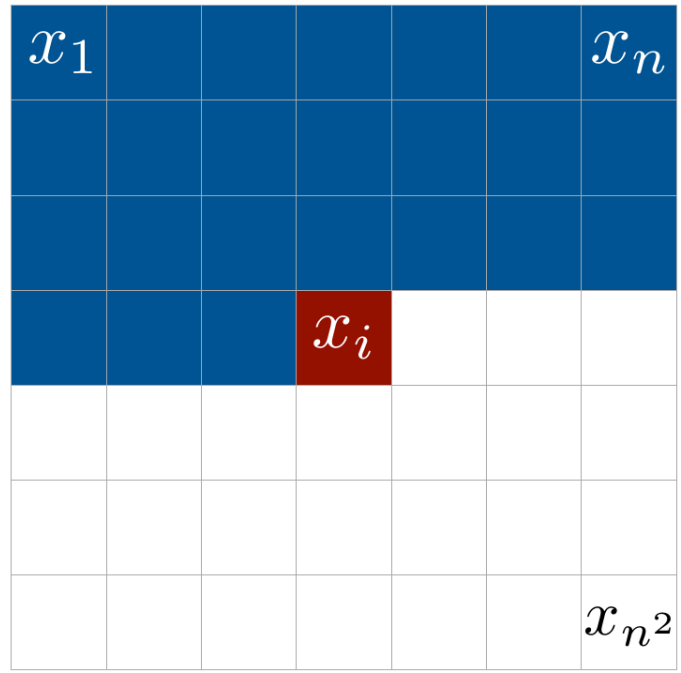
- Predict pixels one-by-one in row-major ordering



Pixel Recurrent/Convolutional Neural Networks

[Oord et al., 2016b]

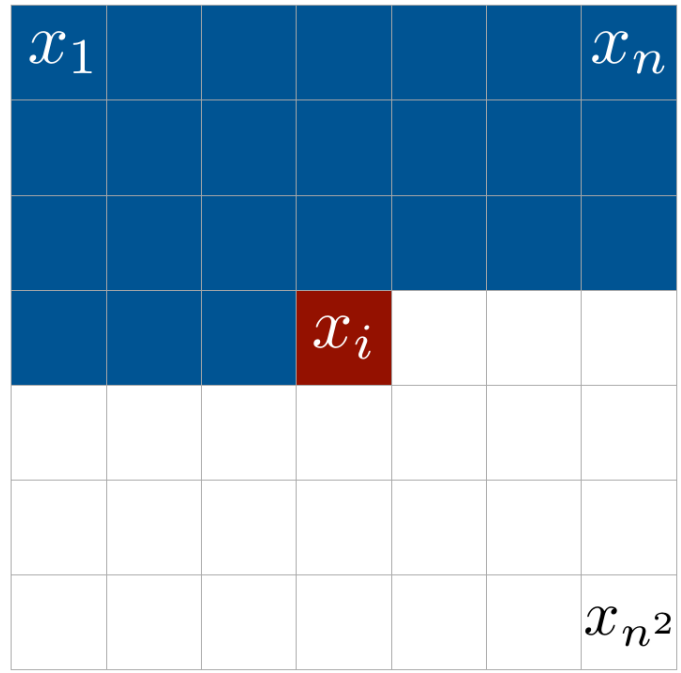
- Predict pixels one-by-one in row-major ordering
- Translation invariant definition of conditionals $p(x_i|\mathbf{x}_{<i})$



Pixel Recurrent/Convolutional Neural Networks

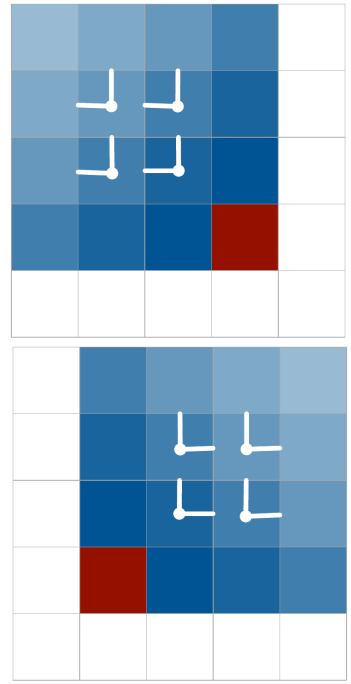
[Oord et al., 2016b]

- Predict pixels one-by-one in row-major ordering
- Translation invariant definition of conditionals $p(x_i|\mathbf{x}_{<i})$
- Decouple number of pixels from number of parameters



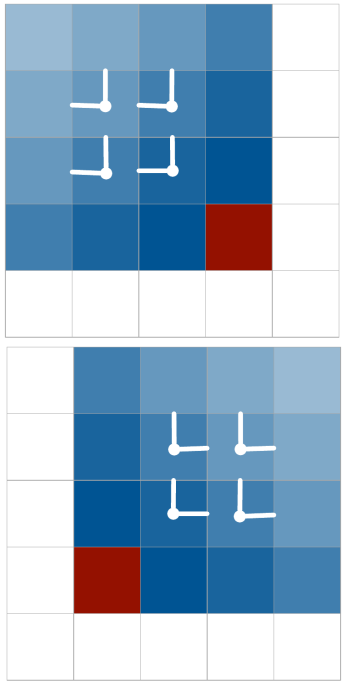
Pixel RNN: Bi-directional LSTM

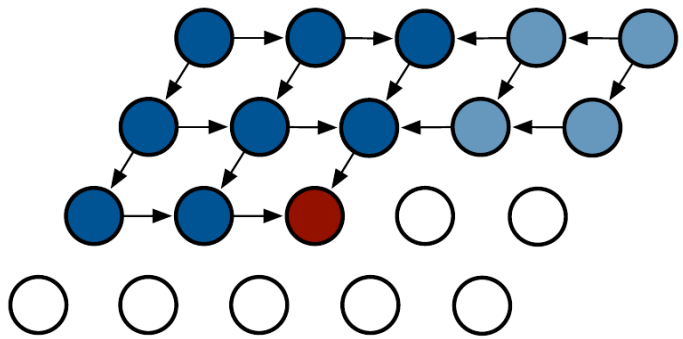
- Two sets of LSTM units, working down-right and down-left
  - Input up and left/right state
  - Input up and left/right pixels



Pixel RNN: Bi-directional LSTM

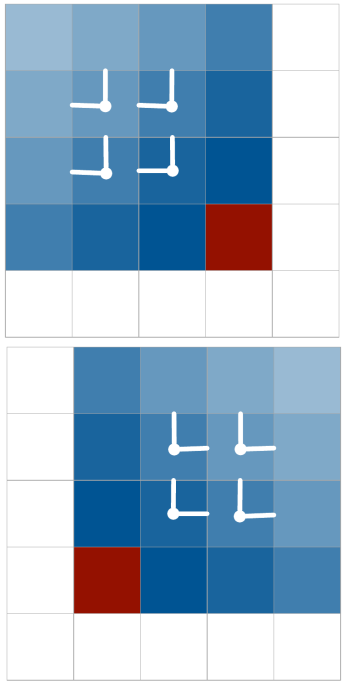
- Two sets of LSTM units, working down-right and down-left
  - Input up and left/right state
  - Input up and left/right pixels
- Receptive field
  - In each stream: all pixels above and to the right/left
  - Combined: all previous pixels

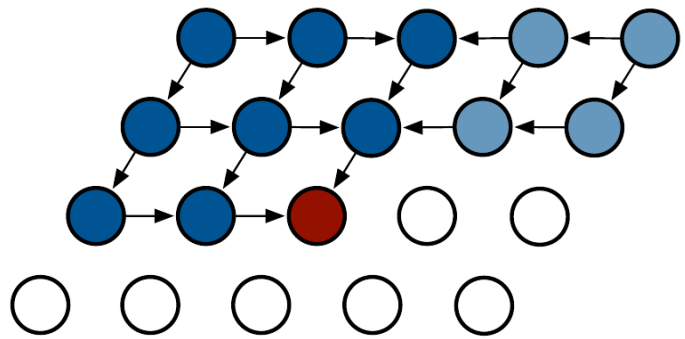




Pixel RNN: Bi-directional LSTM

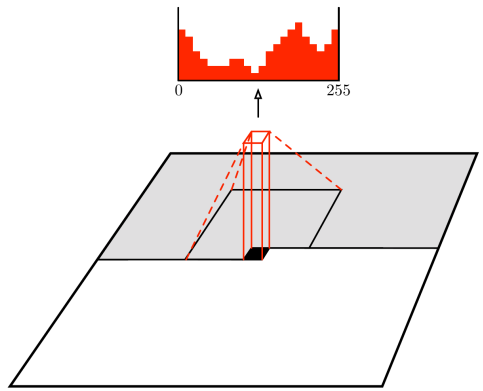
- Two sets of LSTM units, working down-right and down-left
  - Input up and left/right state
  - Input up and left/right pixels
- Receptive field
  - In each stream: all pixels above and to the right/left
  - Combined: all previous pixels
- Slow sequential training process
  - Due to sequential state updates





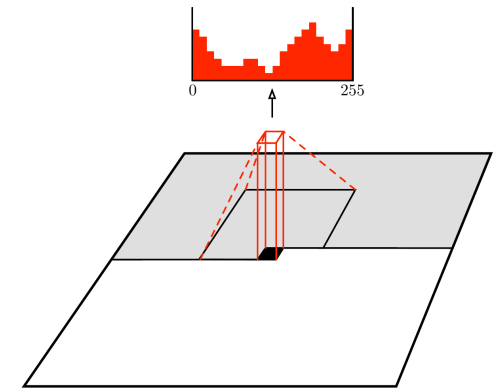
Pixel Convolutional Neural Networks

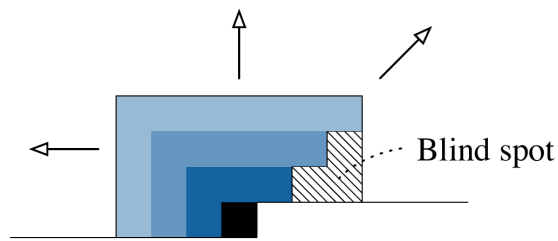
- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property



Pixel Convolutional Neural Networks

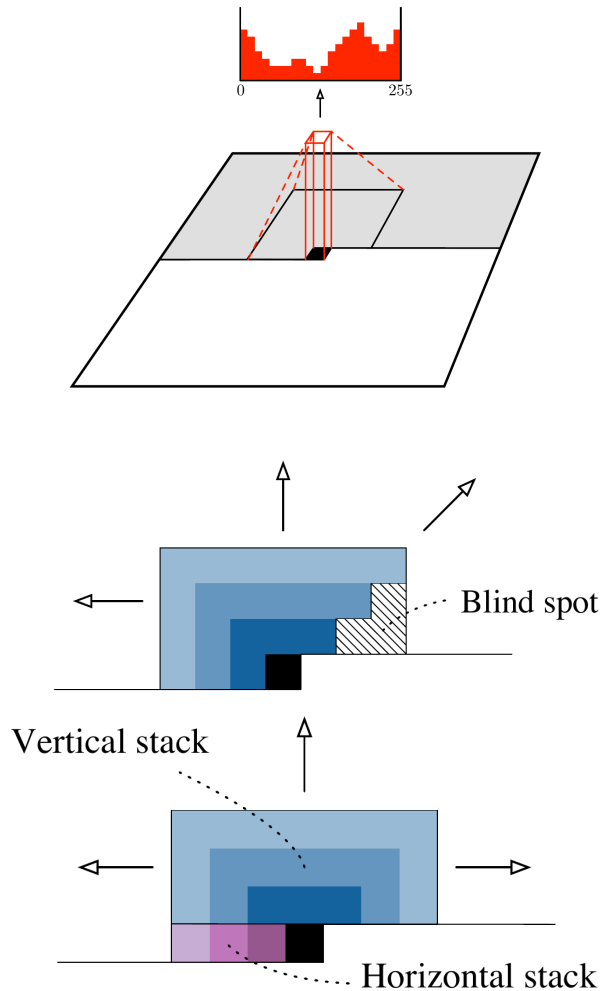
- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property
  - Layers increase receptive field





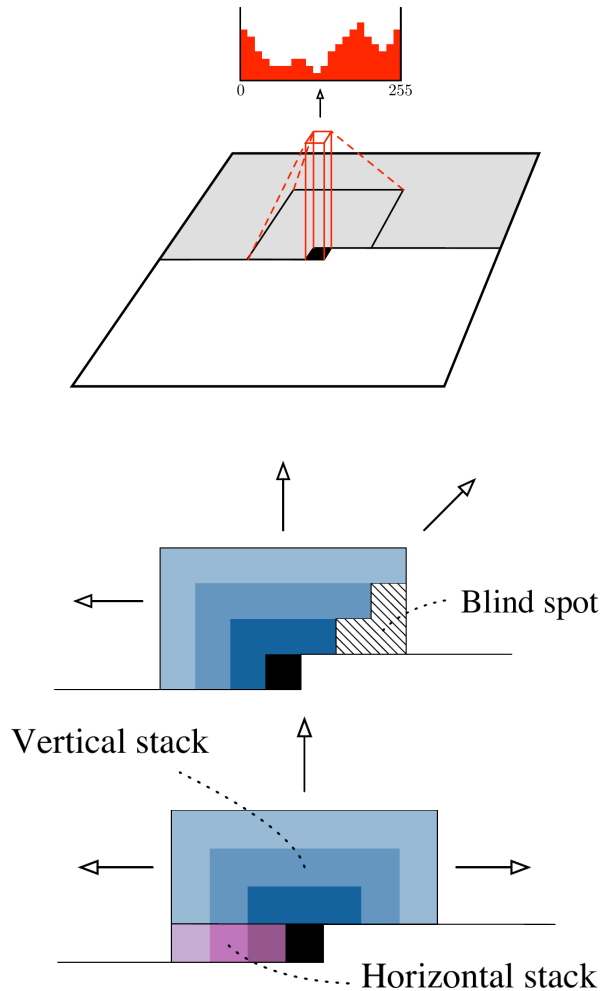
Pixel Convolutional Neural Networks

- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property
  - Layers increase receptive field
  - Two stacks to fill blind spot: horizontal stack reads from vertical stack, not vice-versa



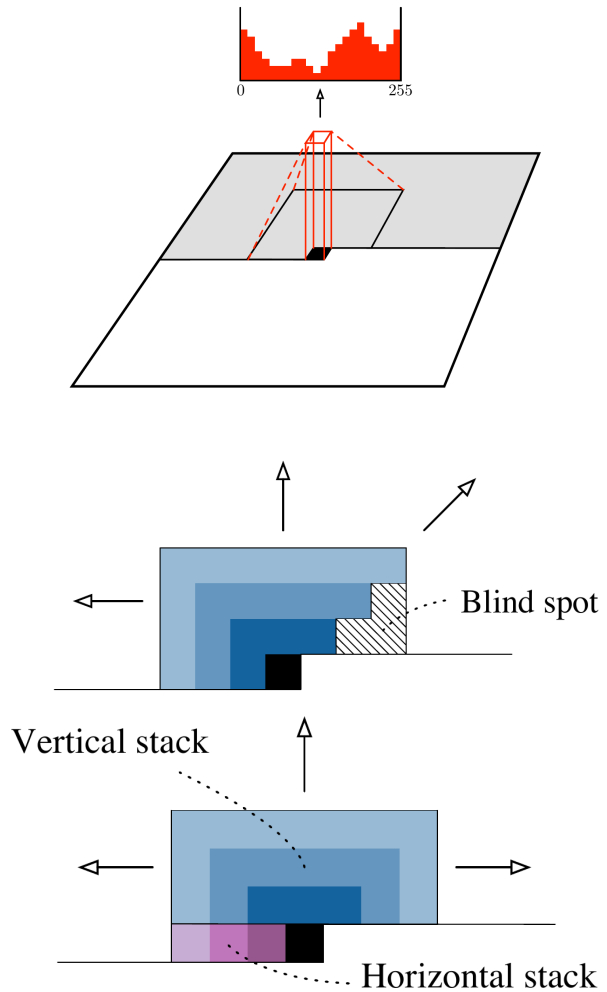
Pixel Convolutional Neural Networks

- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property
  - Layers increase receptive field
  - Two stacks to fill blind spot: horizontal stack reads from vertical stack, not vice-versa
- Efficient parallel training, but sampling remains sequential and slow



Pixel Convolutional Neural Networks

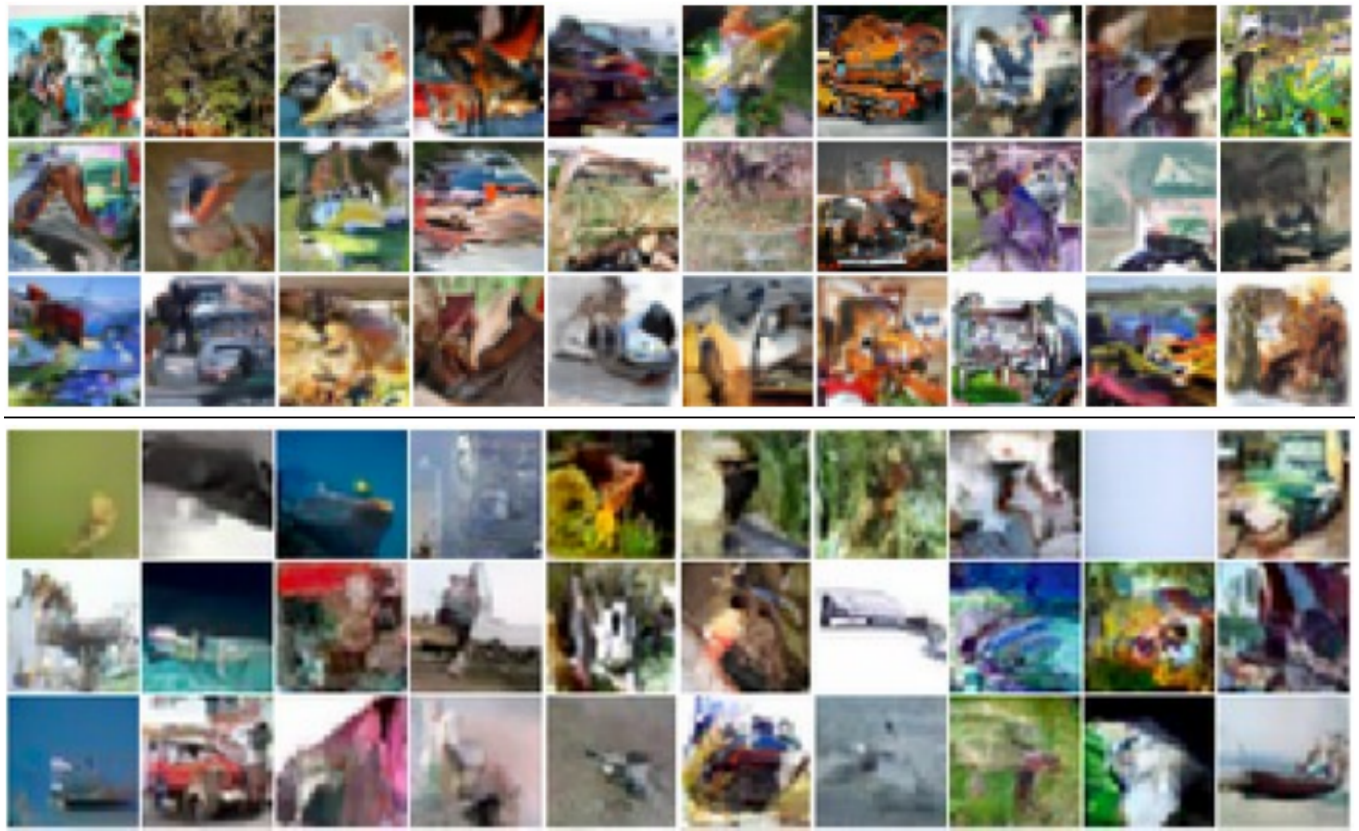
- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property
  - Layers increase receptive field
  - Two stacks to fill blind spot: horizontal stack reads from vertical stack, not vice-versa
- Efficient parallel training, but sampling remains sequential and slow
- Extensions: WaveNet (audio) [Oord et al., 2016a], Video Pixel Networks [Kalchbrenner et al., 2017]



Class-conditional pixelCNN [Oord et al., 2016c]

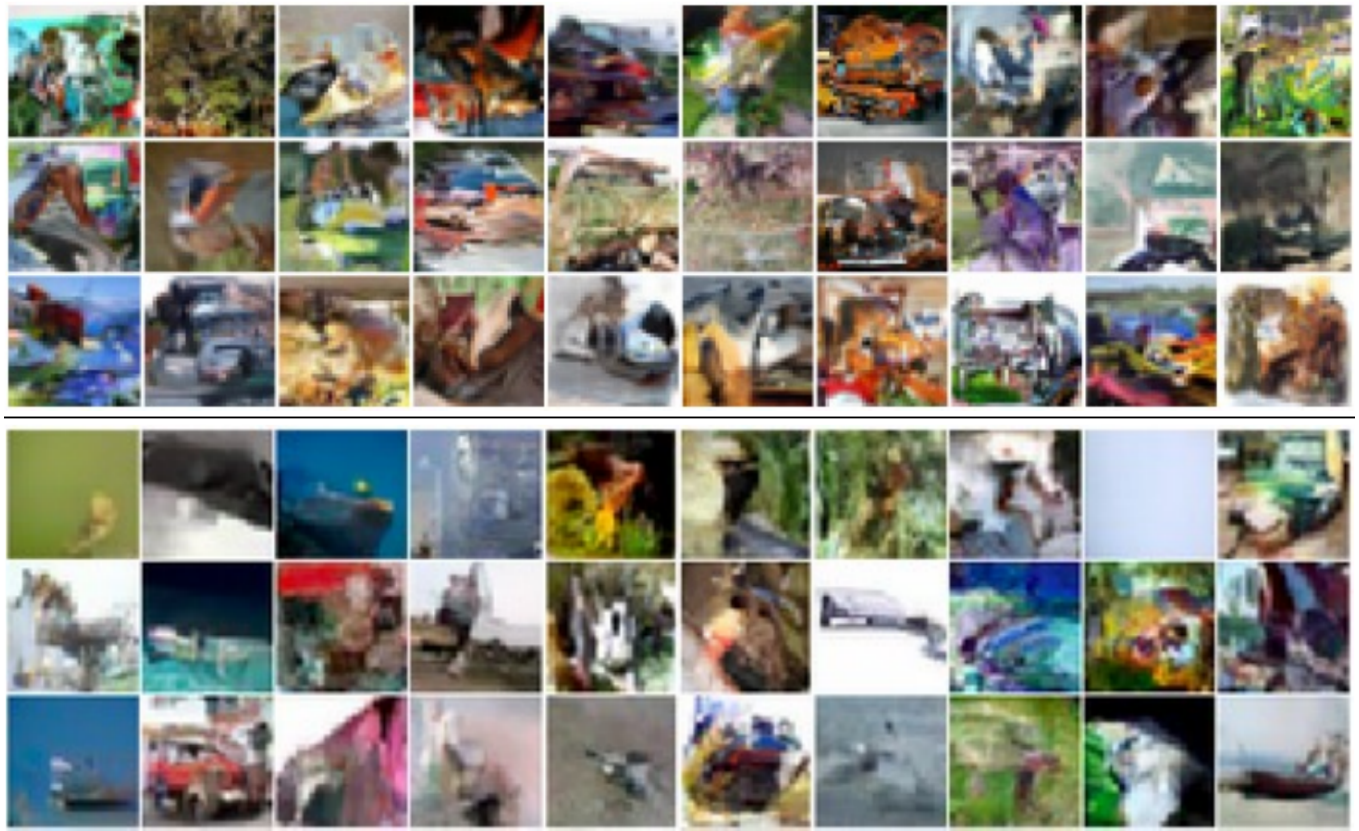
- Samples single model trained across 1,000 ImageNet classes

Images generated by PixelCNNs trained on CIFAR10



[Oord et al., 2016b] (top) and [Salimans et al., 2017] (bottom)

Images generated by PixelCNNs trained on CIFAR10



[Oord et al., 2016b] (top) and [Salimans et al., 2017] (bottom)

- Models capture texture and details relatively well
- Lacking in global structure / long range dependencies

Hierarchical Pixel-CNN [Kolesnikov and Lampert, 2017]

- Introduce an intermediate "high-level" representation $\hat{\mathbf{x}}$
  - Gray scale version, low resolution version, *etc.*

Hierarchical Pixel-CNN [Kolesnikov and Lampert, 2017]

- Introduce an intermediate "high-level" representation $\hat{\mathbf{x}}$
  - Gray scale version, low resolution version, *etc.*
- Learn two-stage Pixel-CNN model
  - High-level model for $\hat{\mathbf{x}}$
  - Conditional model on $\mathbf{x}$ for the details

Hierarchical Pixel-CNN [Kolesnikov and Lampert, 2017]

- Introduce an intermediate "high-level" representation $\hat{\mathbf{x}}$
  - Gray scale version, low resolution version, *etc.*
- Learn two-stage Pixel-CNN model
  - High-level model for $\hat{\mathbf{x}}$
  - Conditional model on $\mathbf{x}$ for the details
- Train independently: $\ln p(\mathbf{x}) \geq \ln p(\hat{\mathbf{x}}) + \ln p(\mathbf{x}|\hat{\mathbf{x}})$

Hierarchical Pixel-CNN [Kolesnikov and Lampert, 2017]

- Introduce an intermediate "high-level" representation $\hat{\mathbf{x}}$
  - Gray scale version, low resolution version, *etc.*
- Learn two-stage Pixel-CNN model
  - High-level model for $\hat{\mathbf{x}}$
  - Conditional model on $\mathbf{x}$ for the details
- Train independently: $\ln p(\mathbf{x}) \geq \ln p(\hat{\mathbf{x}}) + \ln p(\mathbf{x}|\hat{\mathbf{x}})$

CIFAR images (left), gray version $\hat{\mathbf{x}}$ (middle), samples $p(\mathbf{x}|\hat{\mathbf{x}})$ (right)

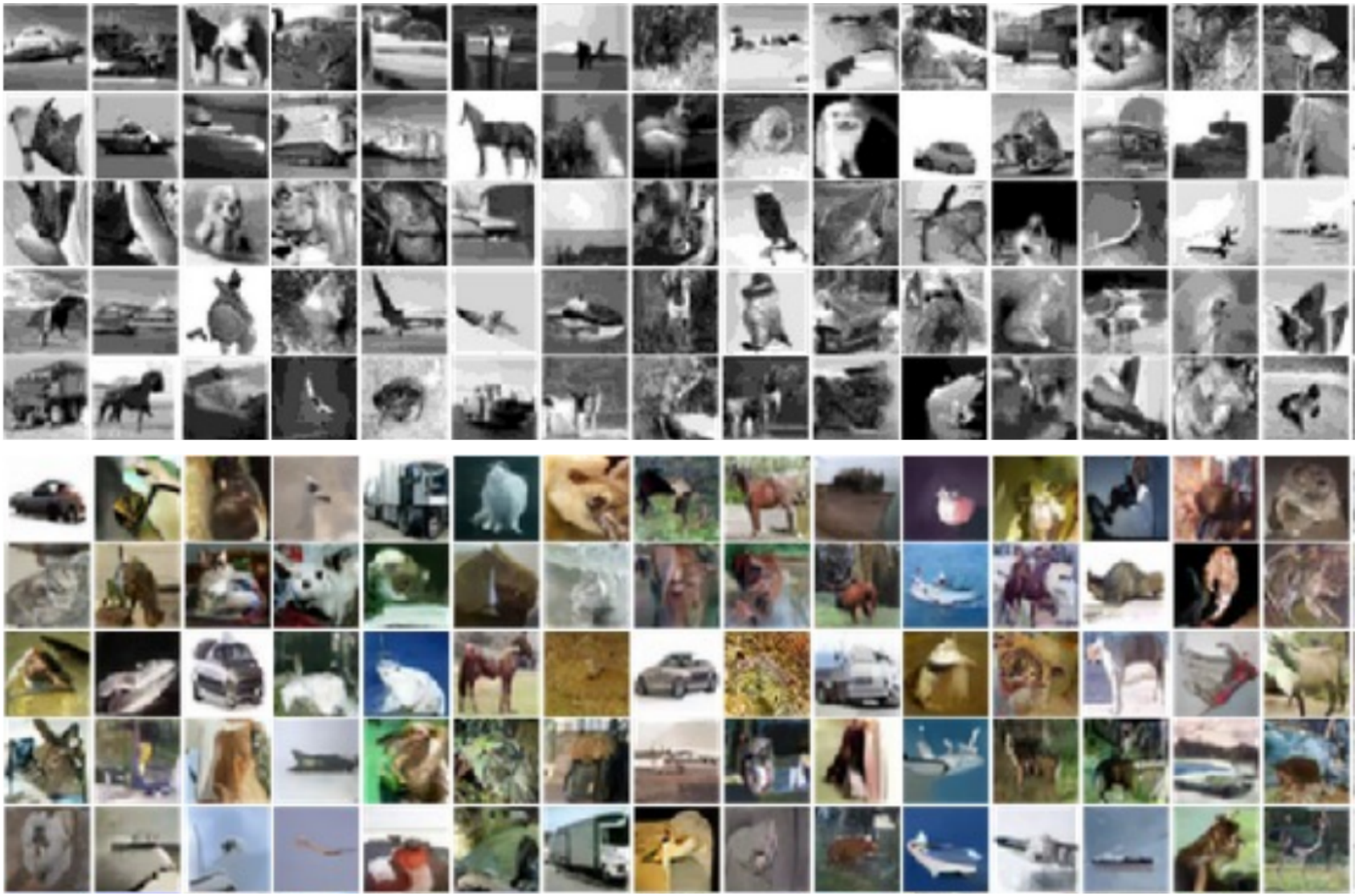
Hierarchical Pixel-CNN [Kolesnikov and Lampert, 2017]

- Introduce an intermediate "high-level" representation $\hat{\mathbf{x}}$
  - Gray scale version, low resolution version, *etc.*
- Learn two-stage Pixel-CNN model
  - High-level model for $\hat{\mathbf{x}}$
  - Conditional model on $\mathbf{x}$ for the details
- Train independently: $\ln p(\mathbf{x}) \geq \ln p(\hat{\mathbf{x}}) + \ln p(\mathbf{x}|\hat{\mathbf{x}})$
- Loss dominated by low-level model

CIFAR images (left), gray version $\hat{\mathbf{x}}$ (middle), samples $p(\mathbf{x}|\hat{\mathbf{x}})$ (right)

Hierarchical Pixel-CNN

- Samples from 4-bit gray model, and conditional color model



Parallel multiscale autoregressive density estimation

[Reed et al., 2017]

- Address the inherently limited sampling efficiency of autoregressive models

$$p(\mathbf{x}_{1:N}) = \prod_{i=1}^{N} p(x_i|\mathbf{x}_{<i})$$

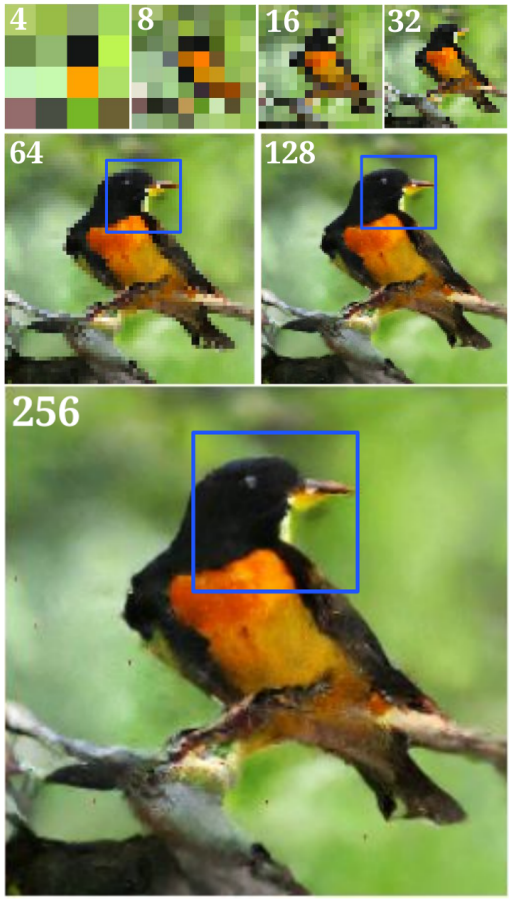
Parallel multiscale autoregressive density estimation

[Reed et al., 2017]

- Address the inherently limited sampling efficiency of autoregressive models

$$p(\mathbf{x}_{1:N}) = \prod_{i=1}^{N} p(x_i|\mathbf{x}_{<i})$$

- Sample image along a scale pyramid
  - Pixel-CNN for base resolution, *e.g.* 4×4
  - Autoregressive upsampling networks

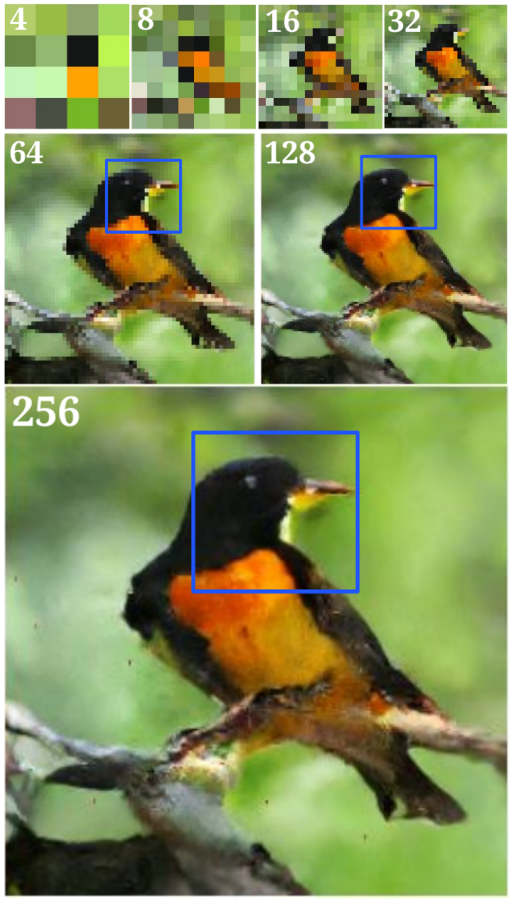


Parallel multiscale autoregressive density estimation

[Reed et al., 2017]

- Address the inherently limited sampling efficiency of autoregressive models

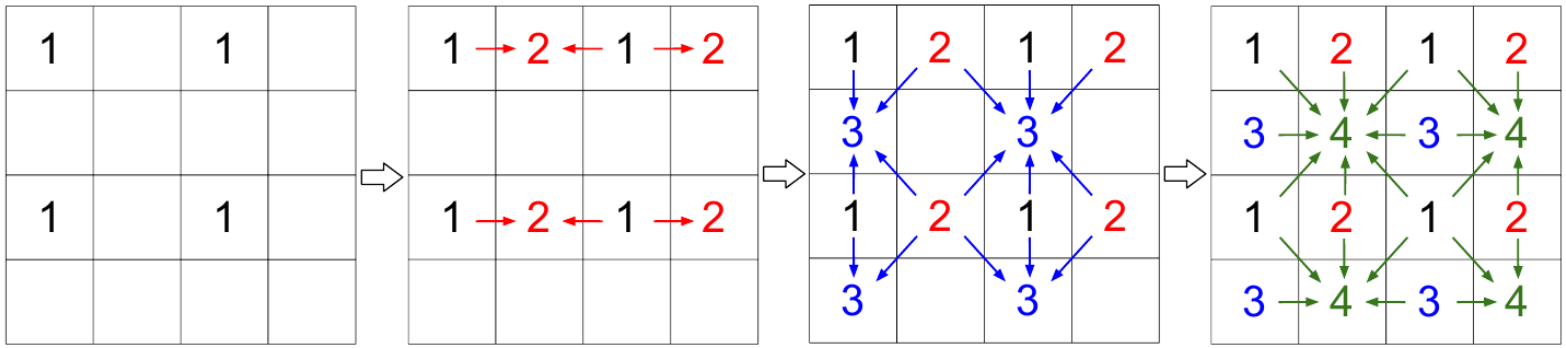
$$p(\mathbf{x}_{1:N}) = \prod_{i=1}^{N} p(x_i|\mathbf{x}_{<i})$$



- Sample image along a scale pyramid
  - Pixel-CNN for base resolution, *e.g.* 4×4
  - Autoregressive upsampling networks
- Impose group structure among pixels
  - Independent sampling within each group
  - Autoregressive sampling across groups

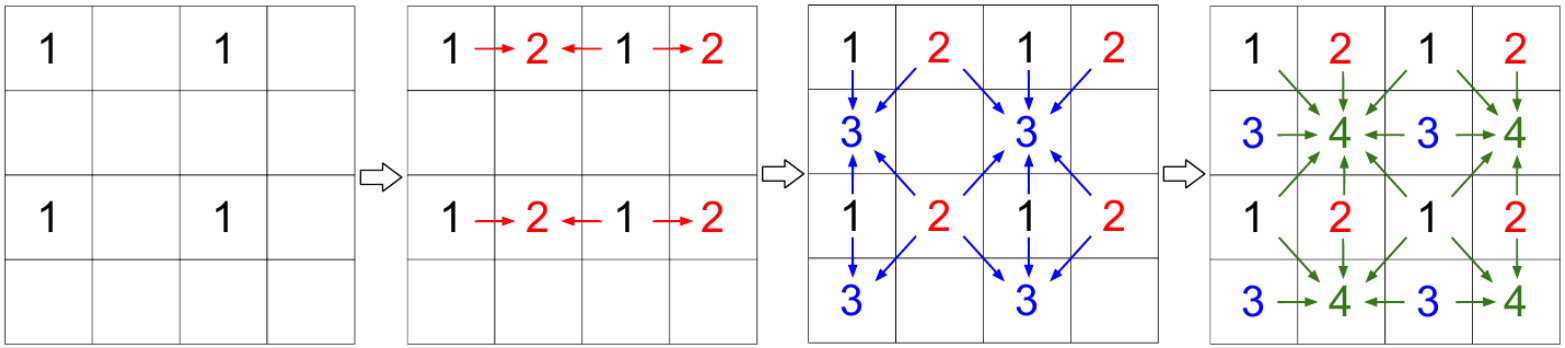
Sampling pixels in groups

- Group pixels along position in 2×2 blocks
  - Group 1 given from previous resolution
  - Sample remaining pixels in three steps

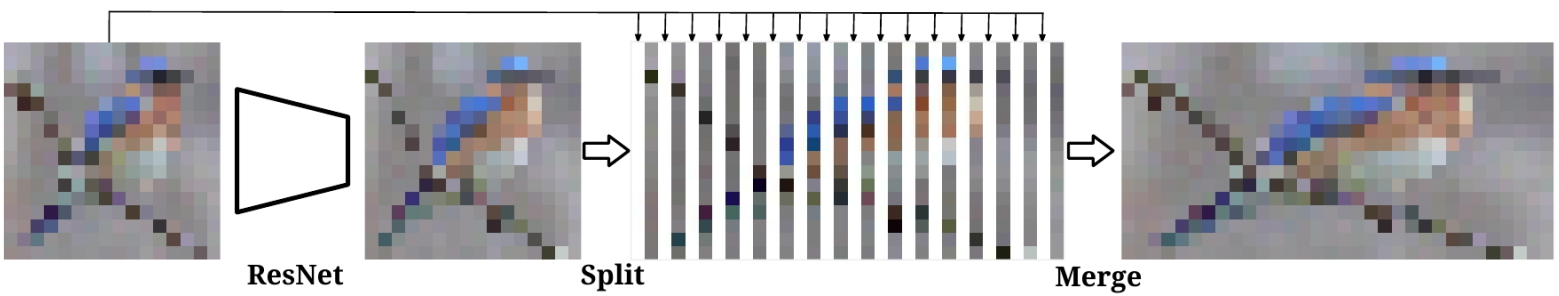


Sampling pixels in groups

- Group pixels along position in 2×2 blocks
  - Group 1 given from previous resolution
  - Sample remaining pixels in three steps

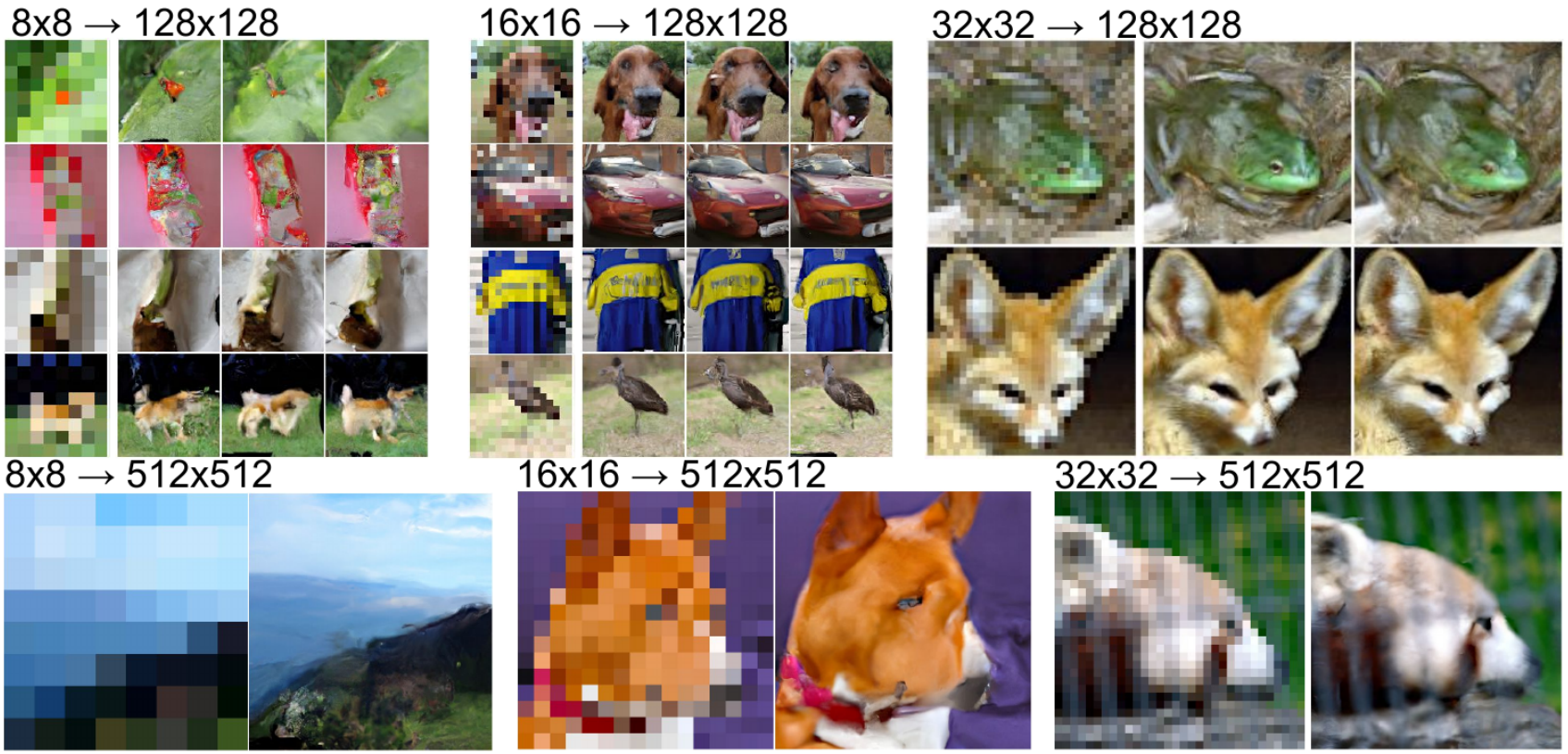


- Example network to predict group 2 from group 1
  - Use CNN without pooling to predict/sample new columns
  - Interleave pixel columns from group 1 and 2



Example results of upsampling real low-resolution images

- About 100× speed-up w.r.t. pixel-CNN sampling



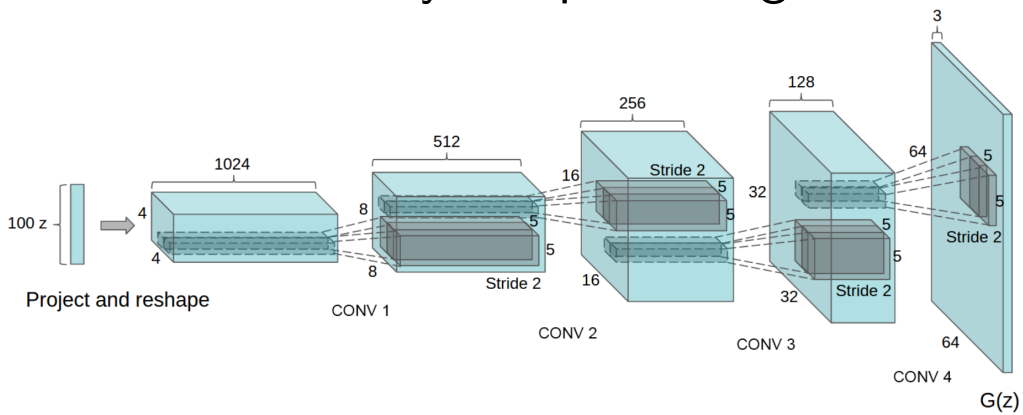
Hybrid VAE autoregressive models

[Gulrajani et al., 2017b, Chen et al., 2017]

Hybrid VAE autoregressive models

[Gulrajani et al., 2017b, Chen et al., 2017]

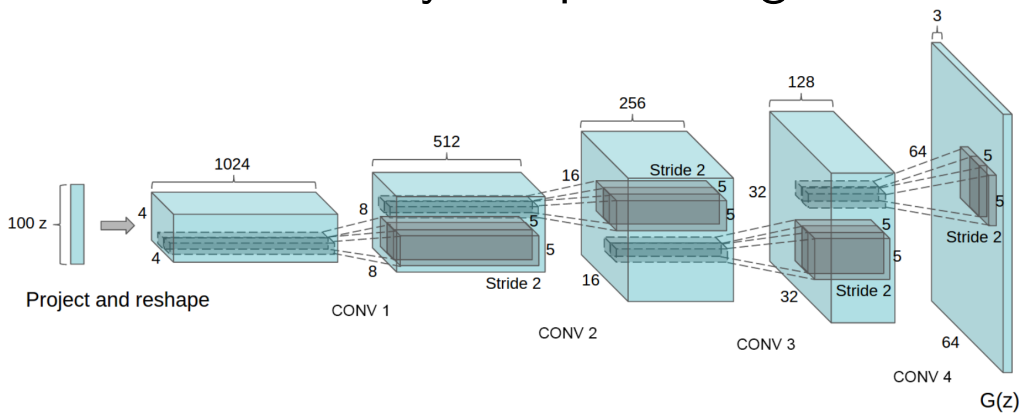
- Variational autoencoder
  - Latent variable $\mathbf{z}$ generates global dependencies
  - Pixels conditionally independent given code



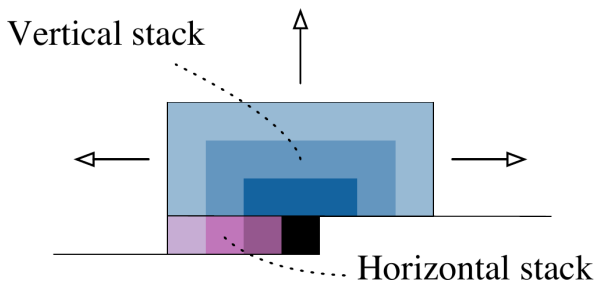
Hybrid VAE autoregressive models

[Gulrajani et al., 2017b, Chen et al., 2017]

- Variational autoencoder
  - Latent variable $\mathbf{z}$ generates global dependencies
  - Pixels conditionally independent given code

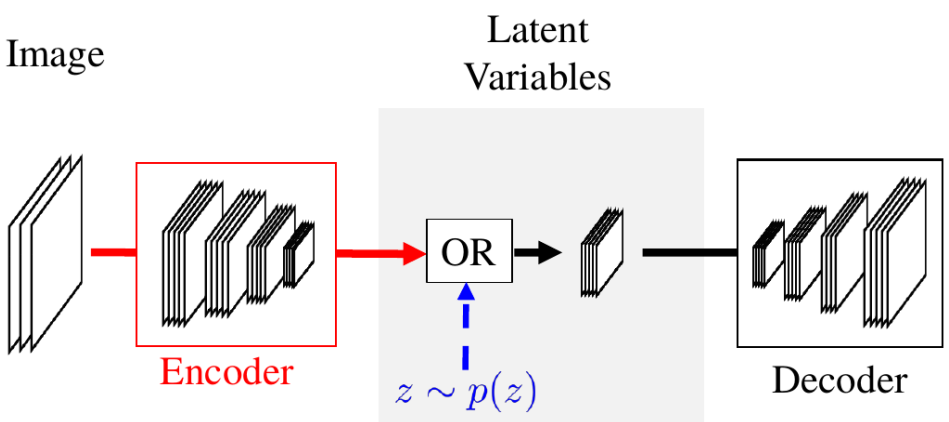


- Autoregressive PixelCNN
  - Needs many layers to induce long-range dependencies
  - Doesn't learn latent representation



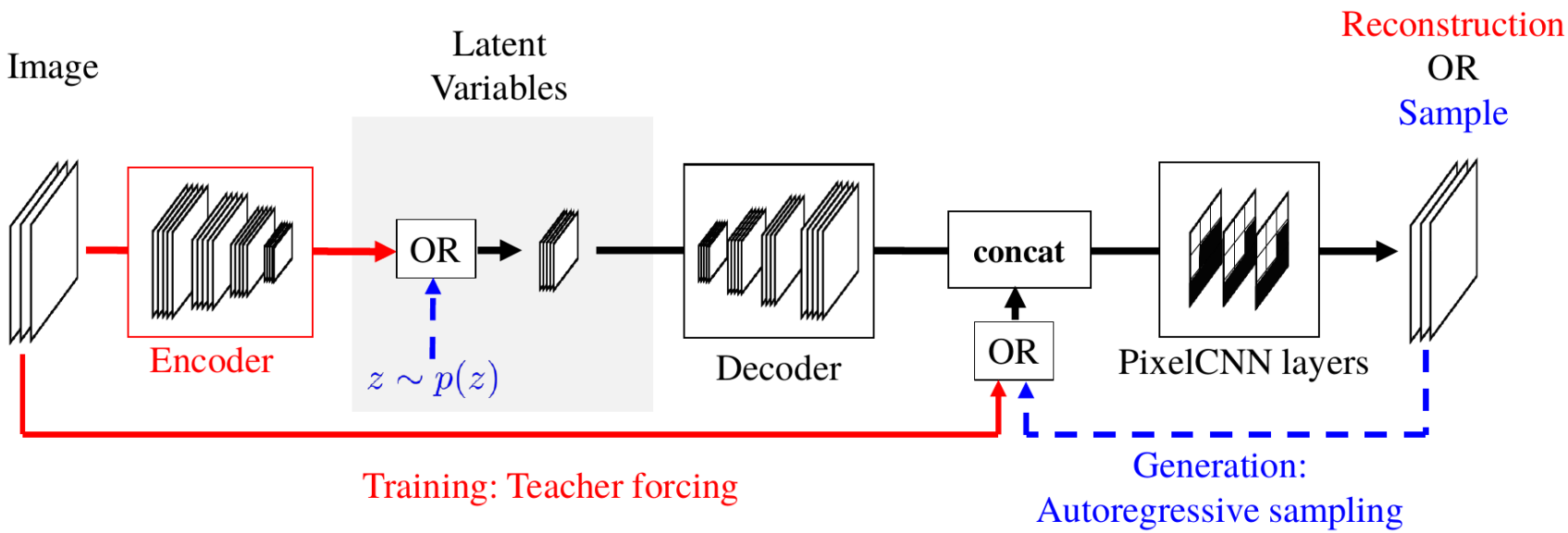
Hybrid PixelVAE model [Gulrajani et al., 2017b]

- Latent var. input to deterministic upsampling decoder $f(\mathbf{z})$



Hybrid PixelVAE model [Gulrajani et al., 2017b]

- Latent var. input to deterministic upsampling decoder $f(\mathbf{z})$
- Pixel-CNN layers induce local pixel dependencies

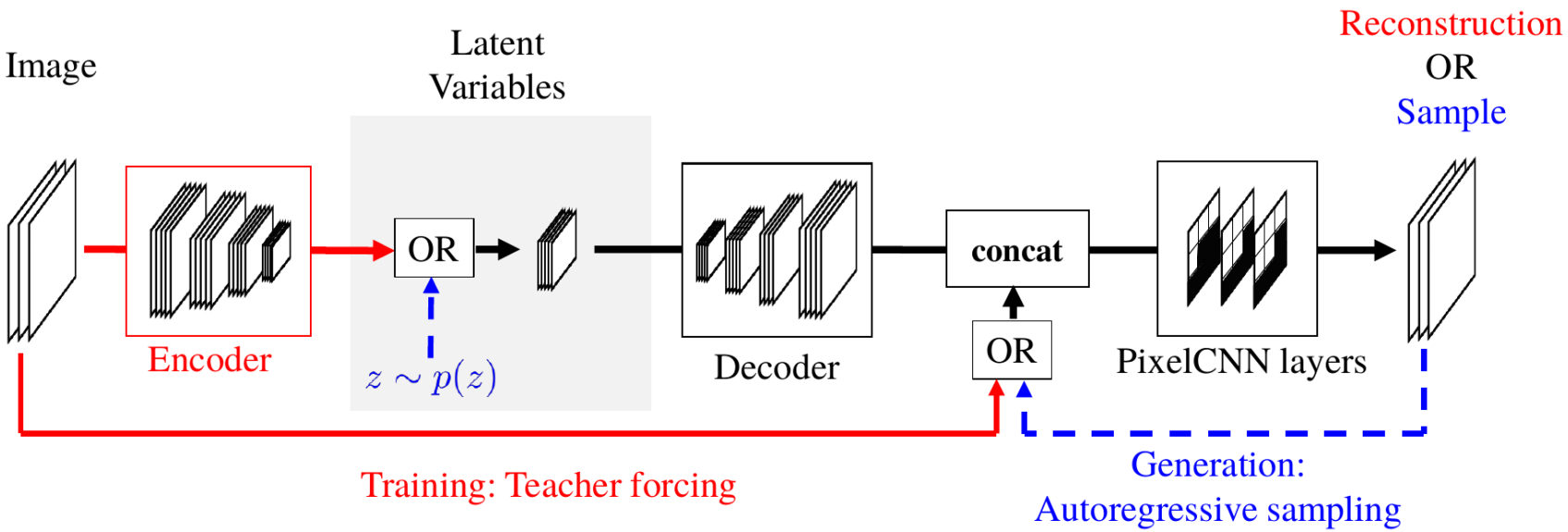


Hybrid PixelVAE model [Gulrajani et al., 2017b]

- Latent var. input to deterministic upsampling decoder $f(\mathbf{z})$
- Pixel-CNN layers induce local pixel dependencies

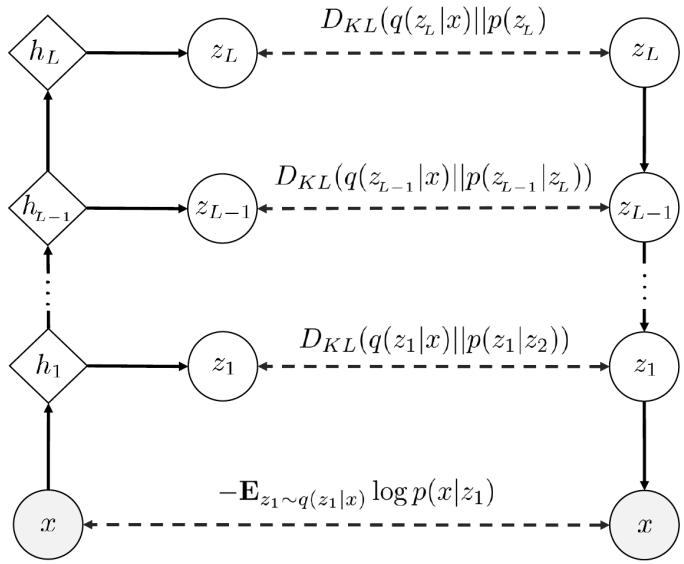
$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, I), \tag{43}$$

$$p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) \prod_i p(x_i | \mathbf{x}_{<i}, f(\mathbf{z})) \tag{44}$$



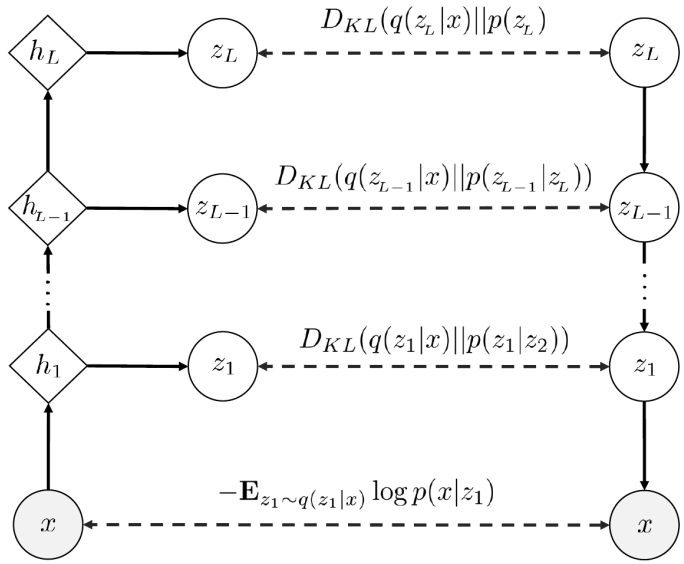
Hierarchical PixelVAE model

- Multiple levels of latent variables at increasing resolutions



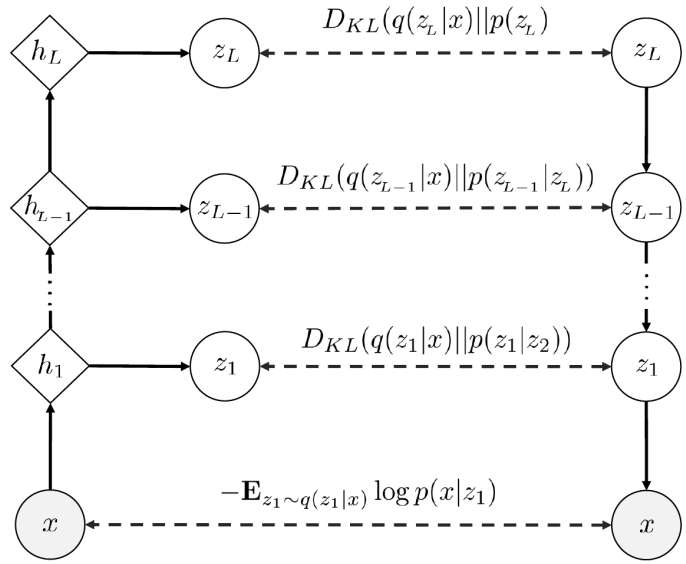
Hierarchical PixelVAE model

- Multiple levels of latent variables at increasing resolutions
- Autoregressive distribution over latent variables in 2D grid



Hierarchical PixelVAE model

- Multiple levels of latent variables at increasing resolutions
- Autoregressive distribution over latent variables in 2D grid



- Extended VAE log-likelihood bound

$$
\begin{aligned}
F &= \ln p(\mathbf{x}) - D_{KL}(q(\mathbf{z}_{1:L}|\mathbf{x})||p(\mathbf{z}_{1:L}|\mathbf{x}) \\
&= \underbrace{\mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}[\ln p(\mathbf{x}|\mathbf{z}_1)]}_{\text{Reconstruction}} - \underbrace{\sum_{i=1}^{L} \mathbb{E}_{q(\mathbf{z}_{i+1})}[D_{KL}(q(\mathbf{z}_i|\mathbf{x})||p(\mathbf{z}_i|\mathbf{z}_{i+1})])}_{\text{Regularization}}
\end{aligned}
$$

Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Samples PixelVAE model LSUN dataset

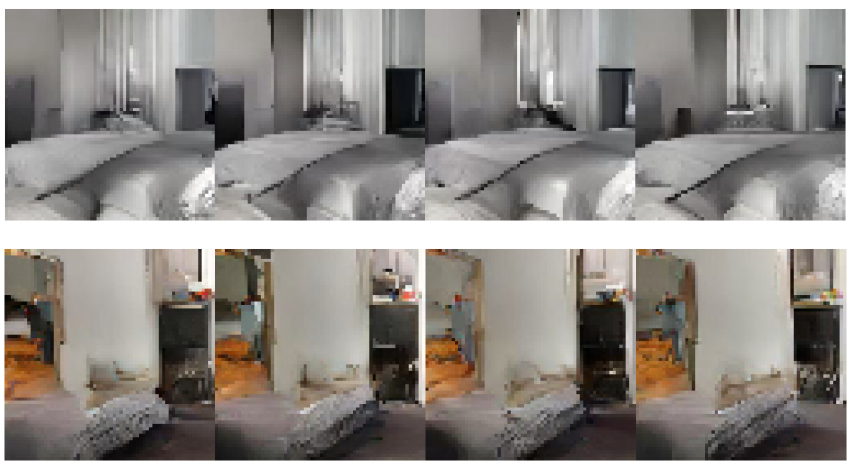
- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Re-sampling PixelCNN only

Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Re-sampling PixelCNN only



Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

Re-sampling PixelCNN only

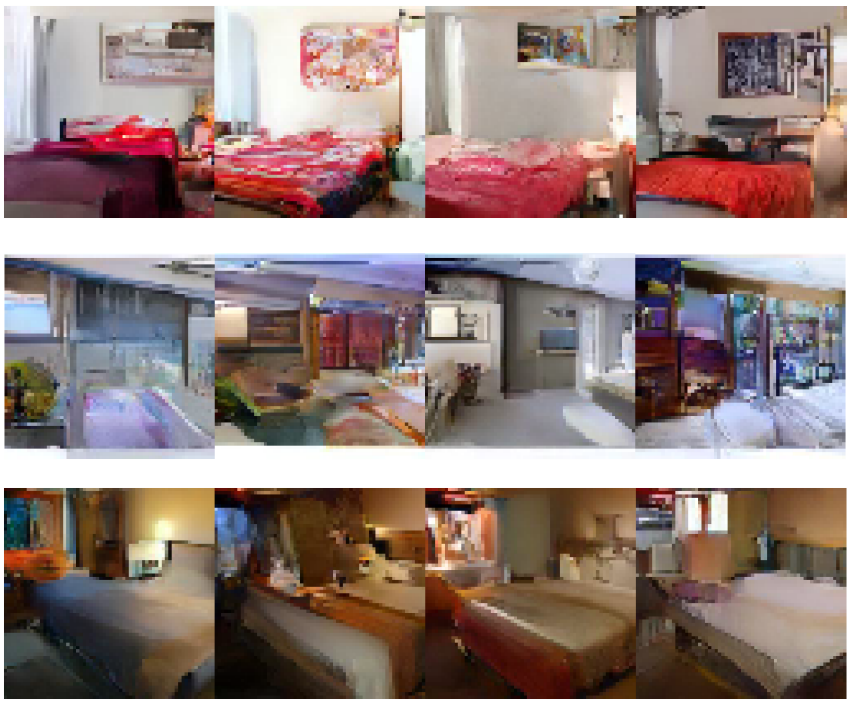
Samples PixelVAE model LSUN dataset

- Model with three levels of stochasticity
  - Latent variables at 1×1
  - Latent variables at 8×8
  - PixelCNN at 64×64

- Hierarchical representation learning

Re-sampling PixelCNN only

Re-sampling 8×8 + PixelCNN



References I

[Arjovsky et al., 2017] Arjovsky, M., Chintala, S., and Bottou, L. (2017).
Wasserstein generative adversarial networks.
In *ICML*.

[Baldi and Hornik, 1989] Baldi, P. and Hornik, K. (1989).
Neural networks and principal component analysis: Learning from examples without local minima.
Neural Networks.

[Bishop, 2006] Bishop, C. (2006).
Pattern recognition and machine learning.
Spinger-Verlag.

[Burda et al., 2016] Burda, Y., Salakhutdinov, R., and Grosse, R. (2016).
Importance weighted autoencoders.
In *ICLR*.

[Chatfield et al., 2011] Chatfield, K., Lempitsky, V., Vedaldi, A., and Zisserman, A. (2011).
The devil is in the details: an evaluation of recent feature encoding methods.
In *BMVC*.

[Chen et al., 2017] Chen, X., Kingma, D., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. (2017).
Variational lossy autoencoder.
In *ICLR*.

[Dinh et al., 2017] Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2017).
Density estimation using real NVP.
In *ICLR*.

[Doersch, 2016] Doersch, C. (2016).
Tutorial on variational autoencoders.
arXiv:1606.05908.

References II

[Doersch et al., 2015] Doersch, C., Gupta, A., and Efros, A. (2015).
Unsupervised visual representation learning by context prediction.
In *ICCV*.

[Donahue et al., 2017] Donahue, J., Krähenbühl, P., and Darrell, T. (2017).
Adversarial feature learning.
In *ICLR*.

[Fernando et al., 2017] Fernando, B., Bilen, H., Gavves, E., and Gould, S. (2017).
Self-supervised video representation learning with odd-one-out networks.
In *CVPR*.

[Germain et al., 2015] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015).
MADE: Masked autoencoder for distribution estimation.
In *ICML*.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
Generative adversarial nets.
In *NIPS*.

[Gulrajani et al., 2017a] Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017a).
Improved training of Wasserstein GANs.
In *NIPS*.

[Gulrajani et al., 2017b] Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. (2017b).
PixelVAE: A latent variable model for natural images.
In *ICLR*.

[Hou et al., 2016] Hou, X., Shen, L., Sun, K., and Qiu, G. (2016).
Deep feature consistent variational autoencoder.
CoRR, abs/1610.00291.

References III

[Kalchbrenner et al., 2017] Kalchbrenner, N., van den Oord, A., Simonyan, K., Danihelka, I., Vinyals, O., Graves, A., and Kavukcuoglu, K. (2017).
Video pixel networks.
In *ICML*.

[Kingma et al., 2014] Kingma, D., Rezende, D., Mohamed, S., and Welling, M. (2014).
Semi-supervised learning with deep generative models.
In *NIPS*.

[Kingma et al., 2016] Kingma, D., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016).
Improved variational inference with inverse autoregressive flow.
In *NIPS*.

[Kingma and Welling, 2014] Kingma, D. and Welling, M. (2014).
Auto-encoding variational Bayes.
In *ICLR*.

[Kolesnikov and Lampert, 2017] Kolesnikov, A. and Lampert, C. (2017).
PixelCNN models with auxiliary variables for natural image modeling.
In *ICML*.

[Larochelle and Murray, 2011] Larochelle, H. and Murray, I. (2011).
The neural autoregressive distribution estimator.

[Lin et al., 2017] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017).
Feature pyramid networks for object detection.
In *CVPR*.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013).
Efficient estimation of word representations in vector space.
In *ICLR*.

References IV

[Oord et al., 2016a] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a).
Wavenet: a generative model for raw audio.
In *ISCA Speech Syntesis Workshop*.

[Oord et al., 2016b] Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016b).
Pixel recurrent neural networks.
In *ICML*.

[Oord et al., 2016c] Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016c).
Conditional image generation with PixelCNN decoders.
In *NIPS*.

[Pathak et al., 2016] Pathak, D., Krähenbühl, P., Donahue, J., Darrell, T., and Efros, A. (2016).
Context encoders: Feature learning by inpainting.
In *CVPR*.

[Radford et al., 2016] Radford, A., Metz, L., and Chintala, S. (2016).
Unsupervised representation learning with deep convolutional generative adversarial networks.
In *ICLR*.

[Reed et al., 2017] Reed, S., van den Oord, A., Kalchbrenner, N., Colmenarejo, S. G., Wang, Z., Belov, D., and de Freitas, N. (2017).
Parallel multiscale autoregressive density estimation.
In *ICML*.

[Rezende and Mohamed, 2015] Rezende, D. and Mohamed, S. (2015).
Variational inference with normalizing flows.
In *ICML*.

[Roweis, 1997] Roweis, S. (1997).
EM Algorithms for PCA and SPCA.
In *NIPS*.

References V

[Royer et al., 2017] Royer, A., Kolesnikov, A., and Lampert, C. (2017).
Probabilistic image colorization.
In *BMVC*.

[Salimans et al., 2017] Salimans, T., Karpathy, A., Chen, X., and Kingma, D. (2017).
PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications.
In *ICLR*.

[Tipping and Bishop, 1999] Tipping, M. E. and Bishop, C. M. (1999).
Mixtures of probabilistic principal component analysers.
Neural Computation, 11(2):443–482.

[Zhu et al., 2017] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. (2017).
Unpaired image-to-image translation using cycle-consistent adversarial networks.
In *ICCV*.