Introduction to Deep learning & Deep Generative Models

Thomas Lucas & Jakob Verbeek

Plan for this presentation

1. Introduction to Deep-learning
    - Building blocks (MLP, convolution, pooling, back-propagation)
    - Deep-learning applications
2. Deep Generative Models
    - Generative modelling basics
    - Generative adversarial network
    - Likelihood-based models

Part I

Introduction to deep learning

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, ...

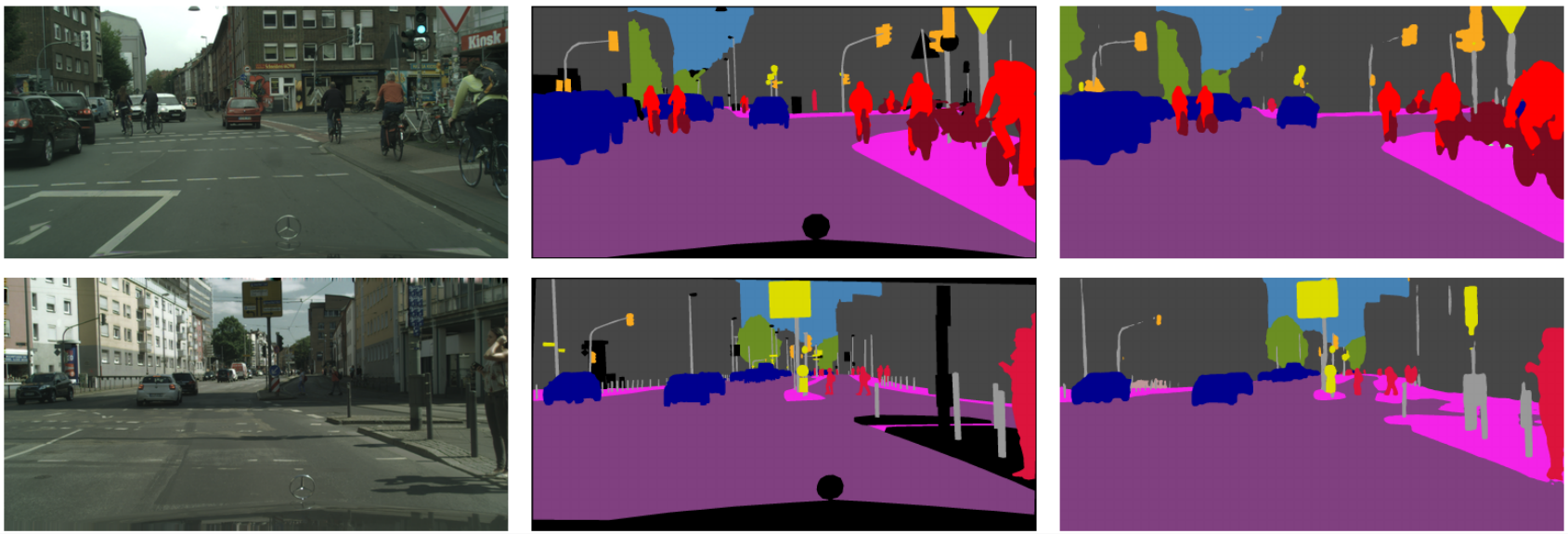
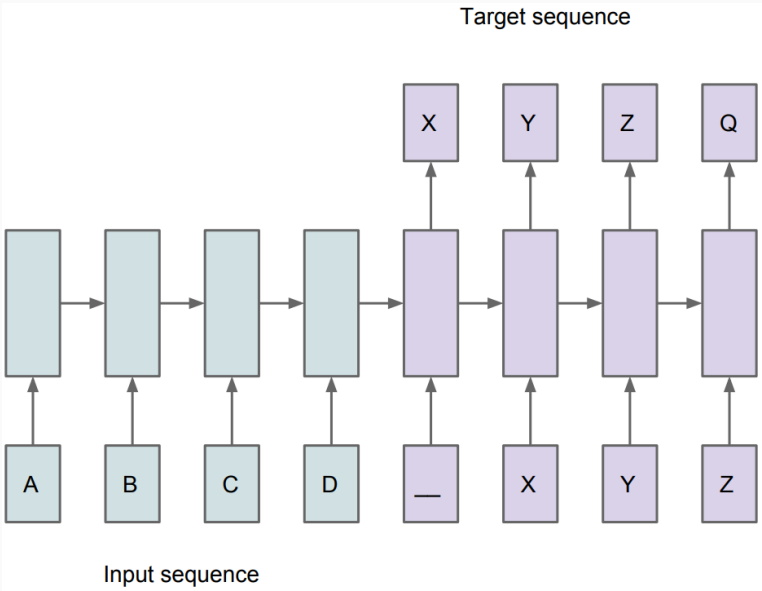


Figure from [Lin et al., 2017]

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

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

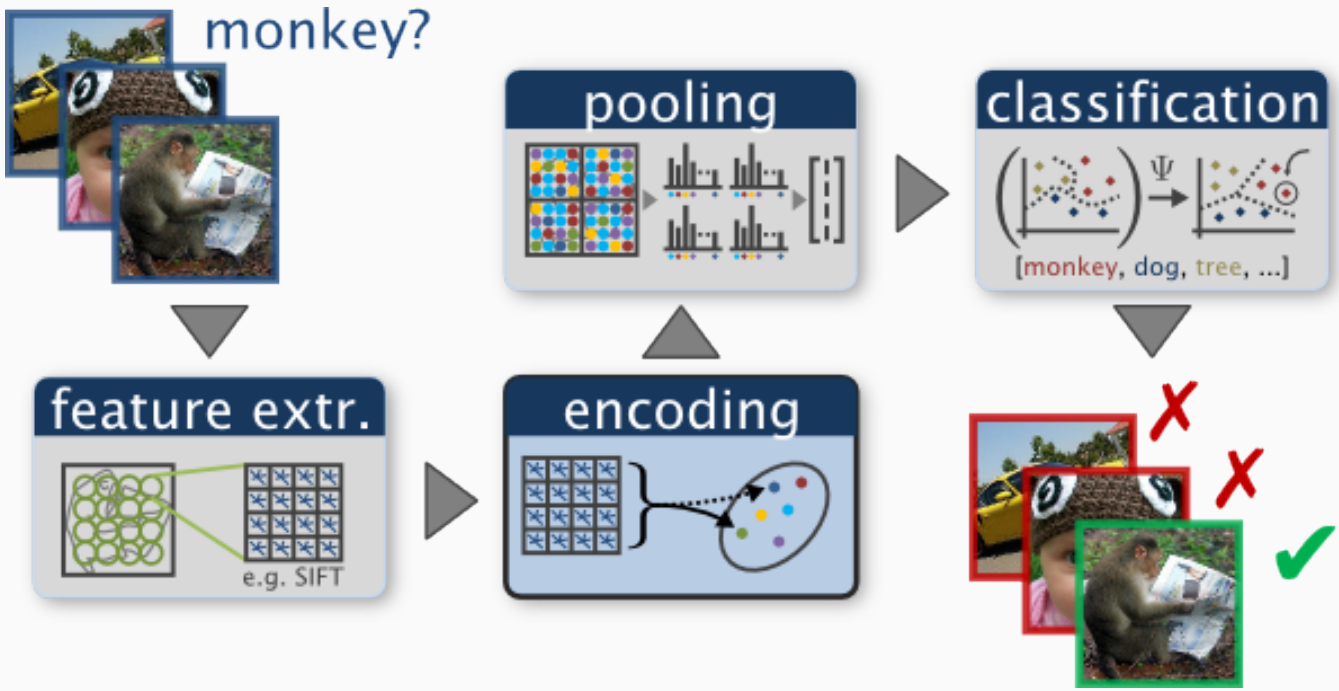
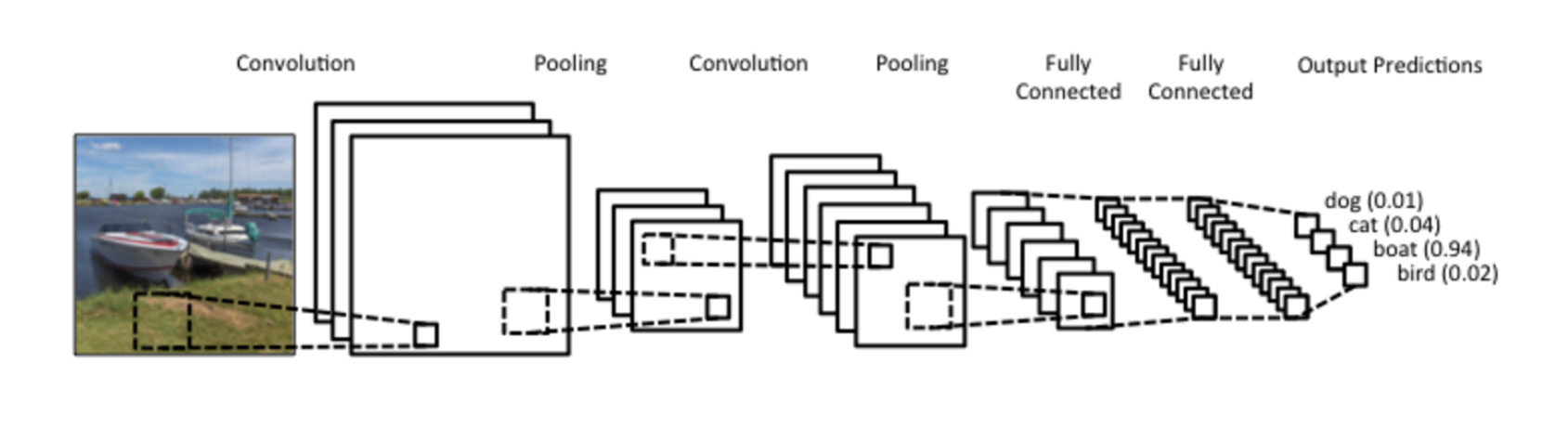


Image from [Chatfield et al., 2011]

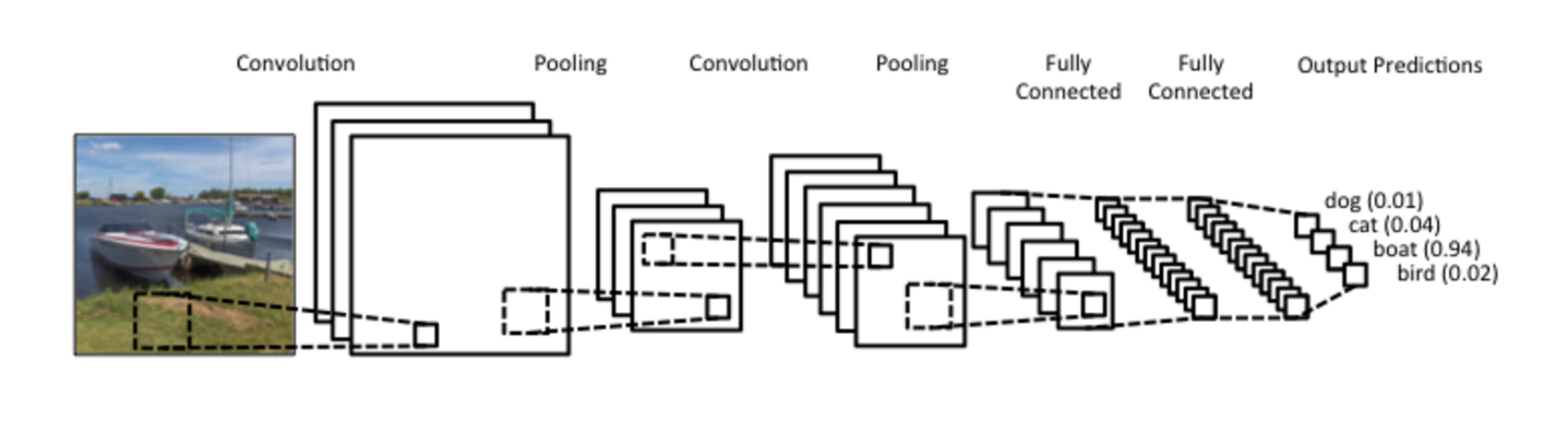
It's all about the features

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



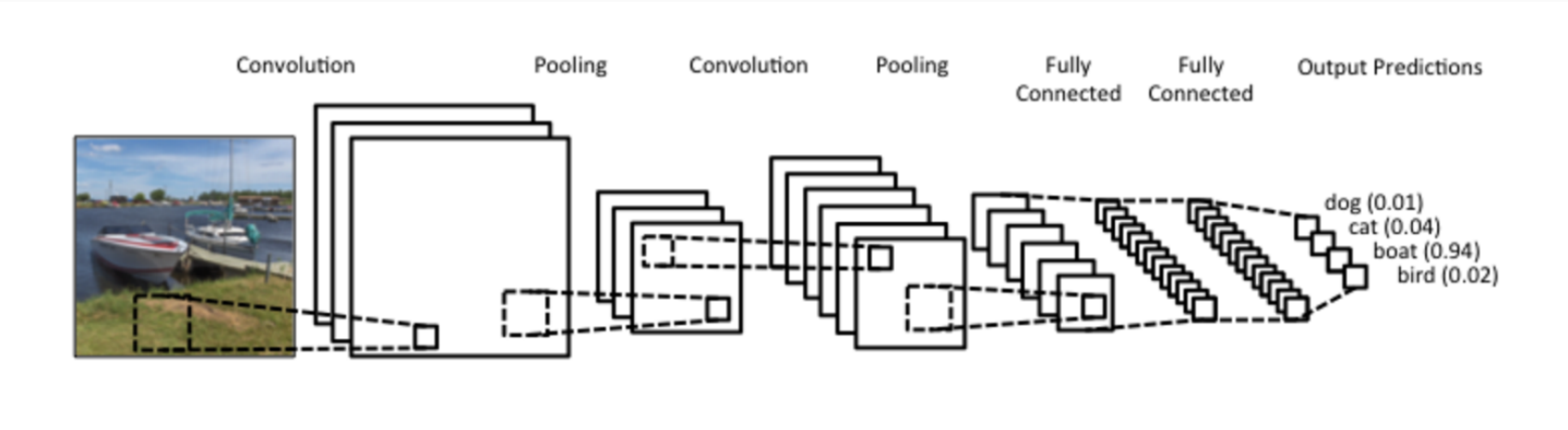
It's all about the features

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



It's all about the features

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



Empirical risk minimization

Training a model by empirical risk minimization

Empirical risk minimization

Training a model by empirical risk minimization

Given labeled training data $(x_i, y_i)_{i=1\ldots n}$ with x_i in X and y_i in Y, learn a prediction function $f : X \to Y$.

Optimize:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}}$$

Empirical risk minimization

Training a model by empirical risk minimization

Given labeled training data $(x_i, y_i)_{i=1...n}$ with x_i in X and y_i in Y, learn a prediction function $f : X \to Y$.

Optimize:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}}$$

The scalars y_i can be in

- $\{$ -1, + 1$\}$: binary classification
- $\{1, \ldots, K\}$: multi-class classification
- $\mathbb{R}$: regression
- $\mathbb{R}^n$: multivariate regression

L evaluates predictions, often convex

Empirical risk minimization

Training a model by empirical risk minimization

Given labeled training data $(x_i, y_i)_{i=1\ldots n}$ with x_i in X and y_i in Y, learn a prediction function $f : X \to Y$.

Optimize:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}}$$

Not just optimization:

- Need to generalize to unseen examples
- Occam's razor (favor simplicity)
- Regularization: control the complexity of solutions

Empirical risk minimization

Training a model by empirical risk minimization

Given labeled training data $(x_i, y_i)_{i=1...n}$ with x_i in X and y_i in Y, learn a prediction function $f : X \to Y$.

Optimize:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}}$$

Linear regression example.

- Assume linear relation between y and features $x \in \mathbb{R}^p$
- $f(x) = w^T x + b$, parametrized by w, b in $\mathbb{R}^{p+1}$
- L is often convex, $\Omega(f)$ often squared l_2-norm $||w||^2$.
- Optimize by gradient descent: follow the steepest direction.
- The problem is convex: local optimum is global.
- Features and classification are decoupled

Empirical risk minimization

Training a model by empirical risk minimization

Given labeled training data $(x_i, y_i)_{i=1...n}$ with x_i in X and y_i in Y, learn a prediction function $f : X \to Y$.

Optimize:

$$\min_{f \in \mathcal{F}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i))}_{\text{empirical risk, data fit}} + \underbrace{\lambda \Omega(f)}_{\text{regularization}}$$

Deep learning example.

- Composition of linear regressions, Non-linearities in between
- Parametrization of the "deep-learning space"

$$\mathcal{F} : f(x) = \sigma_k(A_k \sigma_{k-1}(A_{k-1} \dots \sigma_2(A_2 \sigma_1(A_1 x))))$$

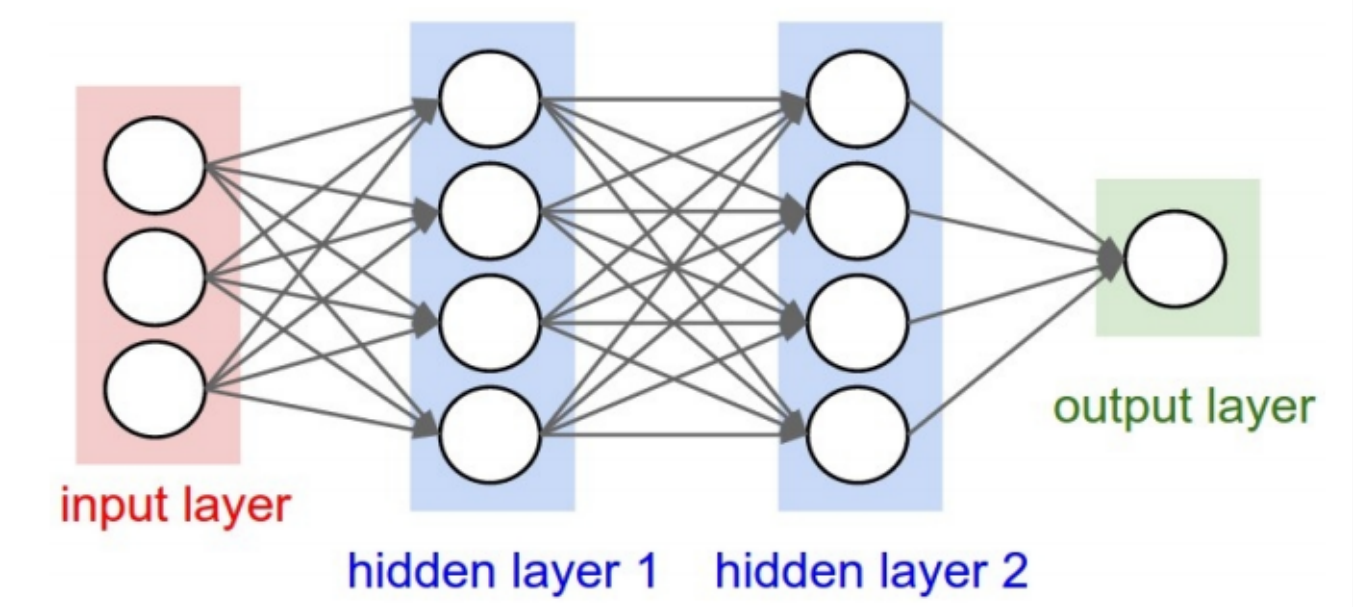
- Adaptive features, Universal approximation theorem, but generalization poorly understood
- Hard to optimize: Non-convex, high-dimensional.

Fully connected networks

Representation of a fully connected network:

Fully connected networks

Representation of a fully connected network:



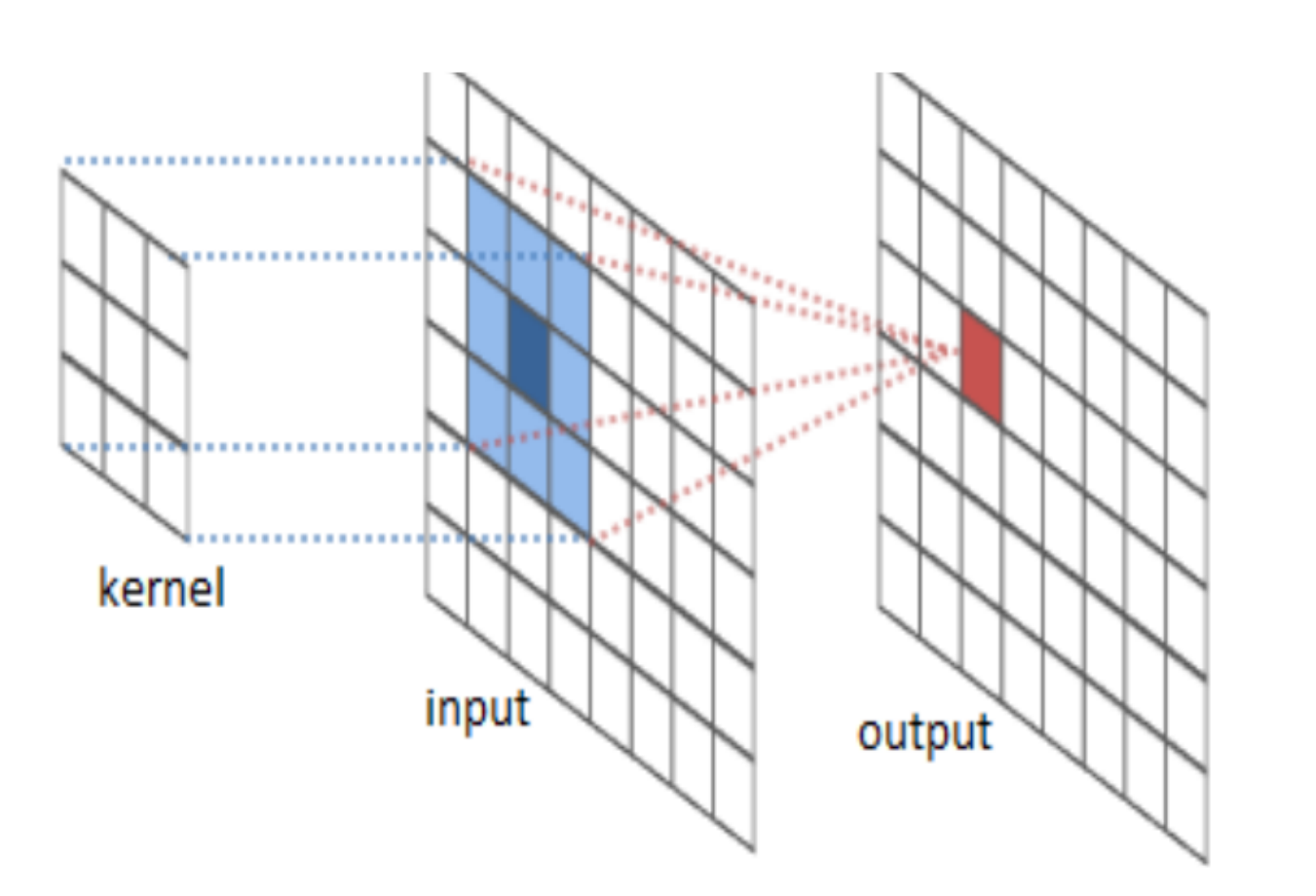
- Stack of linear operations
- Non-linearities in between
- One connection = one parameter

Convolutional networks

The convolution operation:

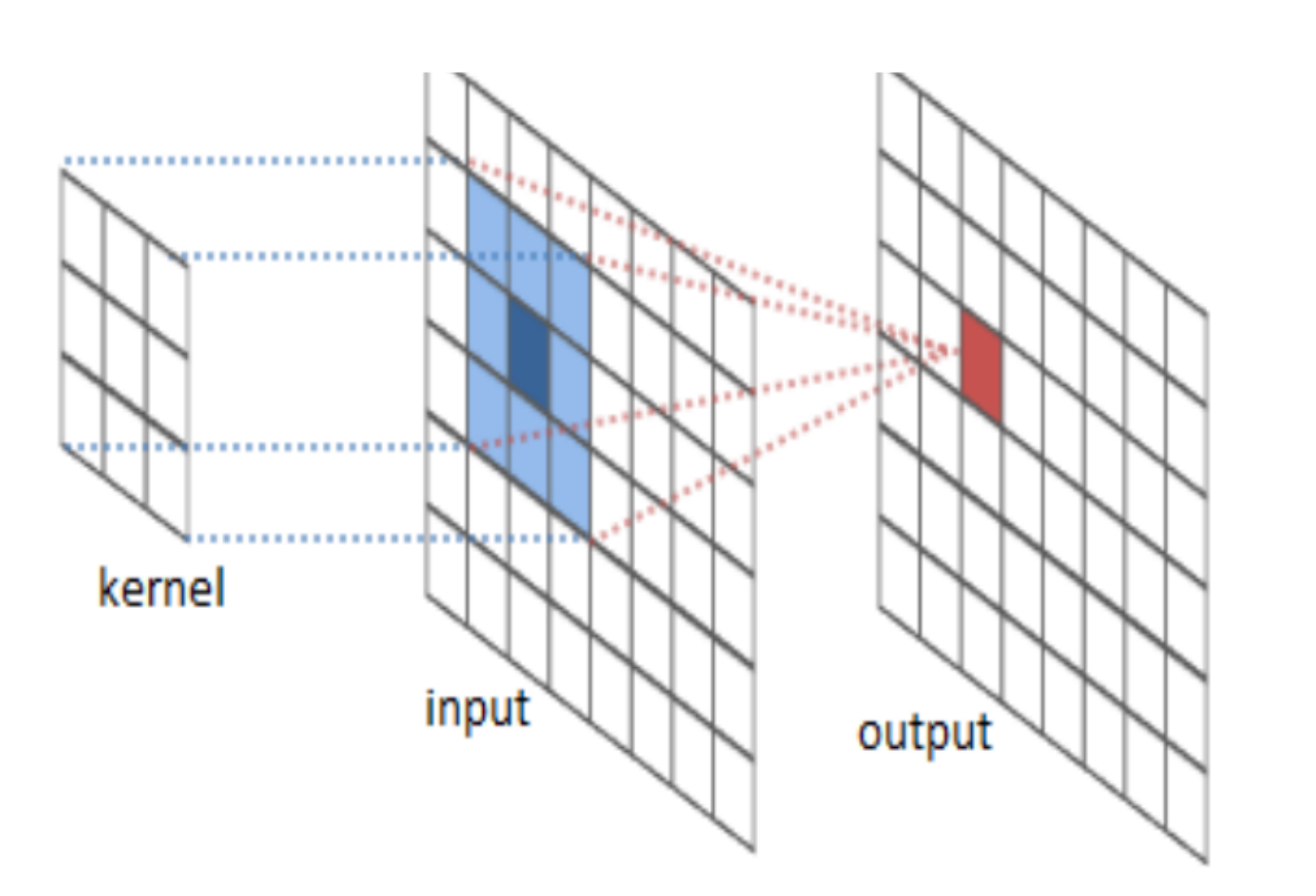
Convolutional networks

The convolution operation:



Convolutional networks

The convolution operation:



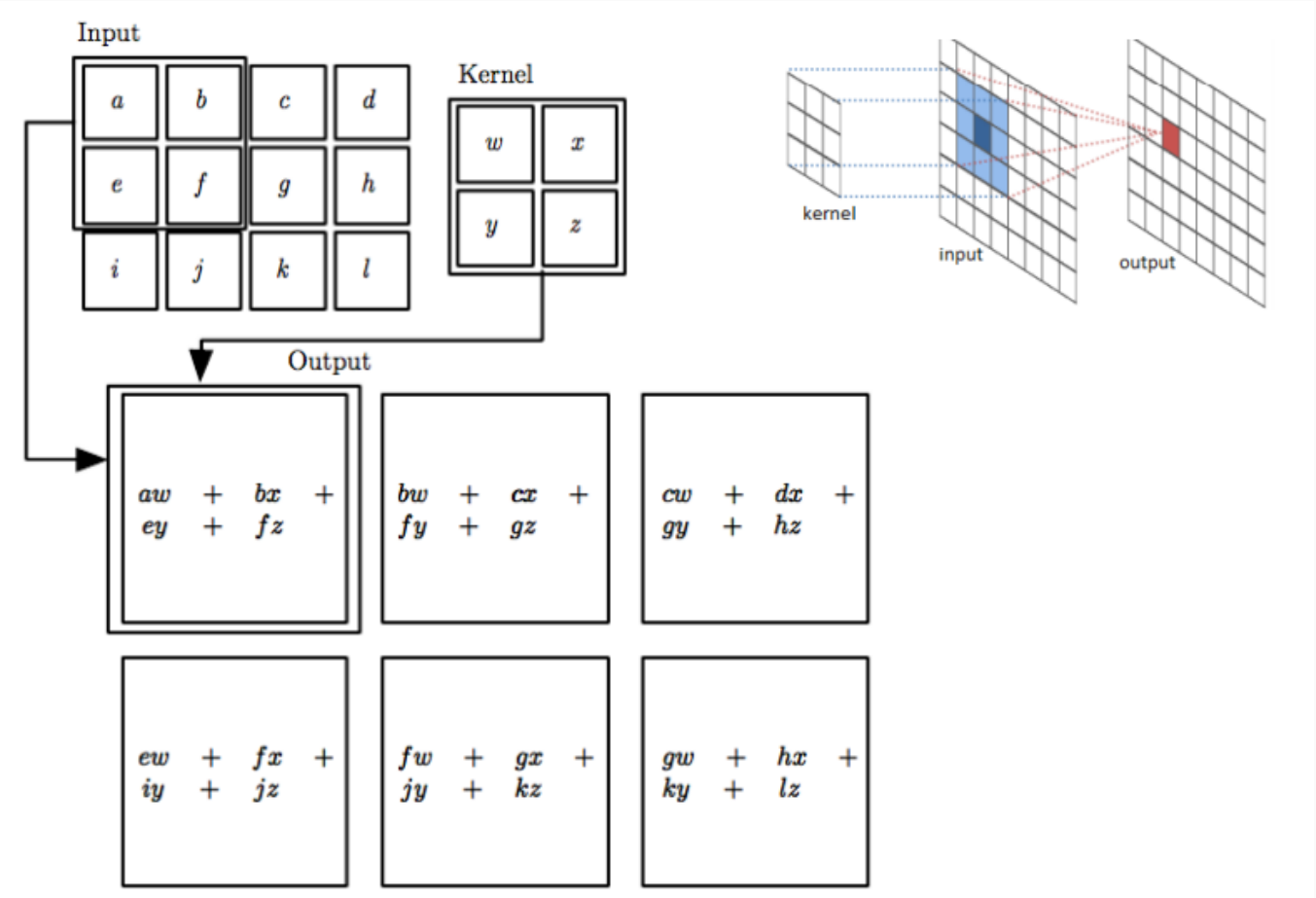
- A lot of weight sharing
- Implements Translation equivariance
- Many such features in parallel

Convolutional networks

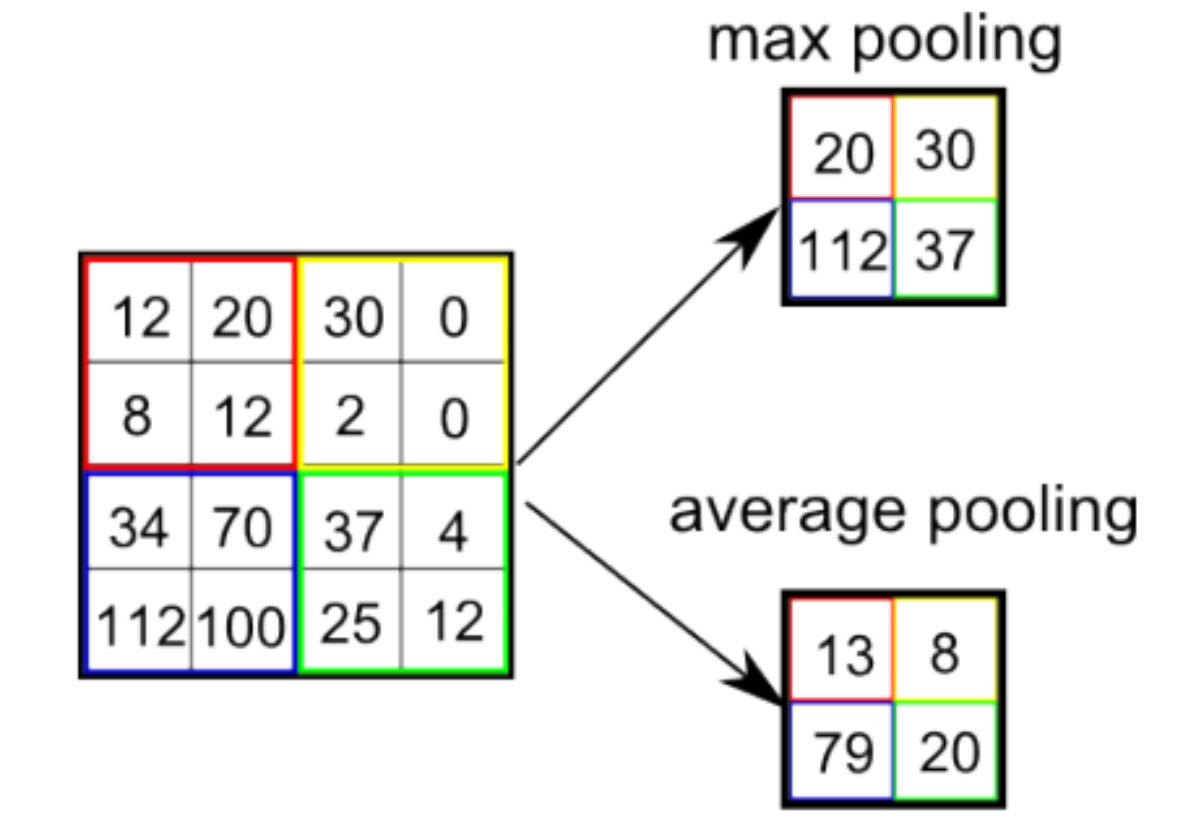
Detailed computation:

Convolutional networks

Detailed computation:

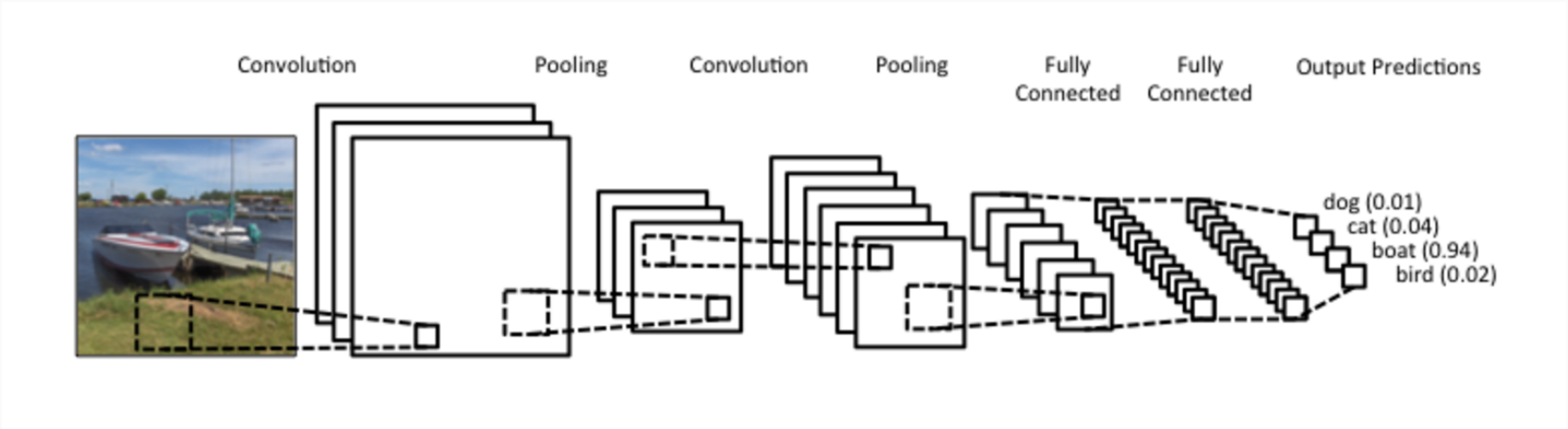


Pooling operations

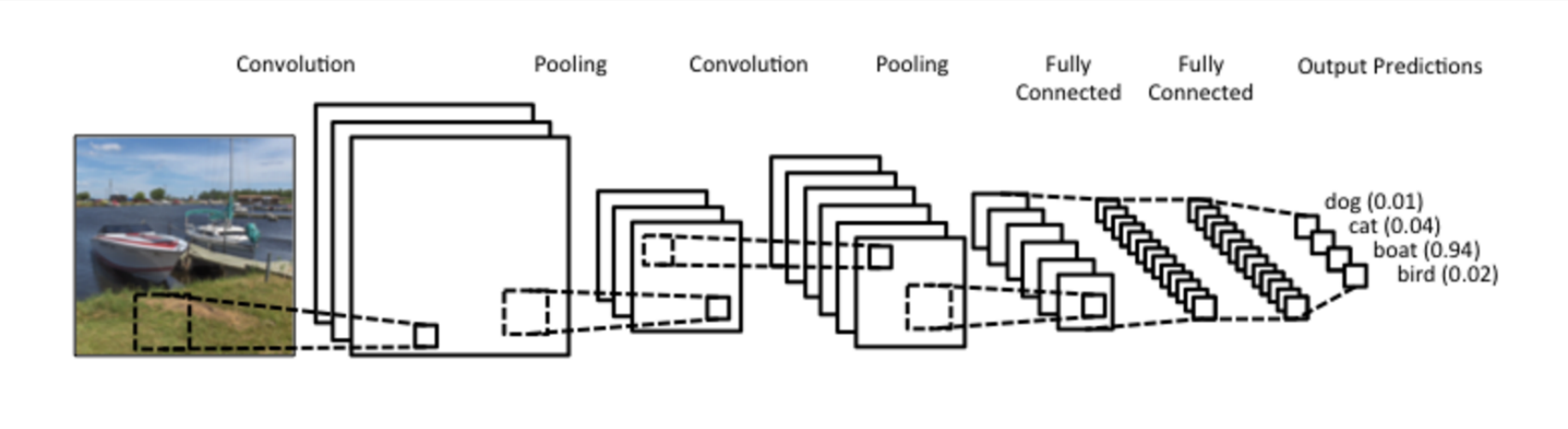


- Reduce spatial dimension
- Increase receptive field
- Gives robustness to small variations

Building networks and feature visualisation



Building networks and feature visualisation

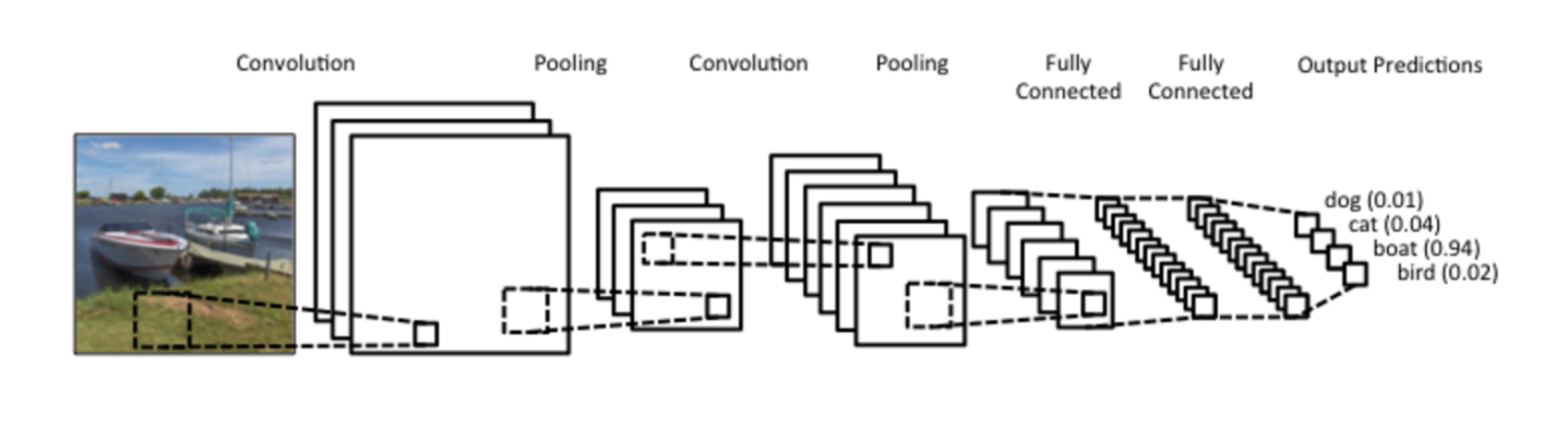


- Stack convolutions, pooling and non-linearities ($\sigma(x) = x1_{x>0}$)
- Train by gradient descent (use the chain rule)

$$D_\theta(g \circ f) = (D_{f(\theta)}(g)) \circ D_\theta f$$

- Efficient computations (see the backpropagation algorithm)

Building networks and feature visualisation



Features visualisation

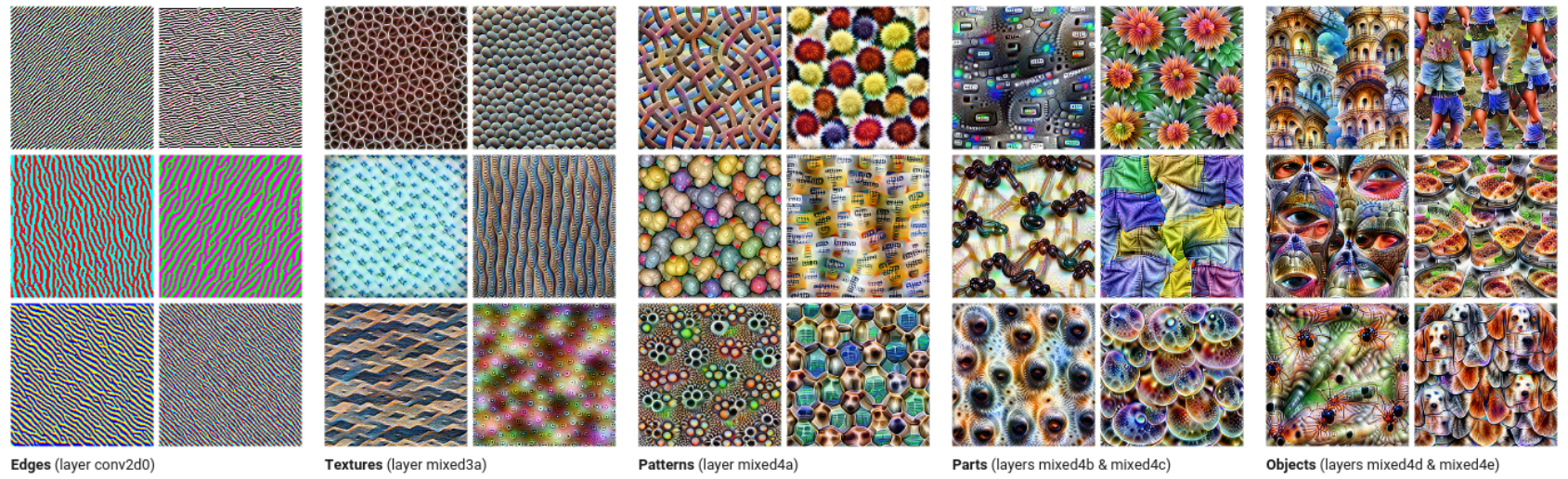


Figure from distill.pub

Take-home messages

- Core idea:
  - Many processing layers from raw input to output
  - Features building and classification done together

Take-home messages

- Core idea:
  - Many processing layers from raw input to output
  - Features building and classification done together
- In practice:
  - Strategy efficient across disciplines (vision, speech, NLP, games etc.)
  - Large-scale applications widely adopted in industry
  - Computation and data hungry

Take-home messages

- Core idea:
  - Many processing layers from raw input to output
  - Features building and classification done together
- In practice:
  - Strategy efficient across disciplines (vision, speech, NLP, games etc.)
  - Large-scale applications widely adopted in industry
  - Computation and data hungry
- In Theory:
  - Optimization still poorly understood
  - Generalization still poorly understood
  - Experimental results 'ahead' of theory

Part II

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**.

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

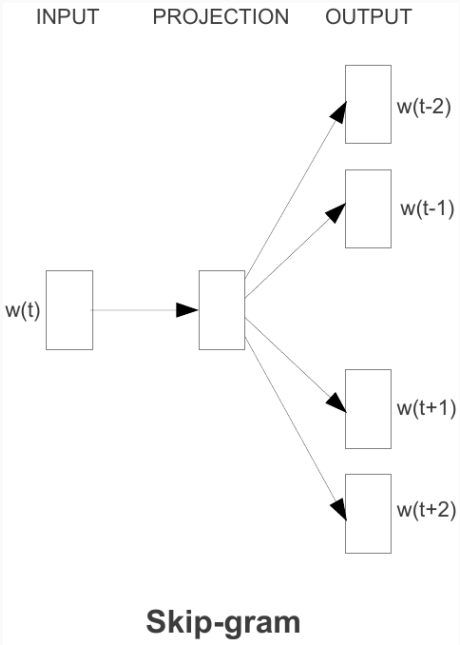
$$\max_\theta \sum_{\mathbf{x}\sim\mathcal{D}} \ln p_\theta(\mathbf{x}) \tag{2}$$

Self-supervised learning

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

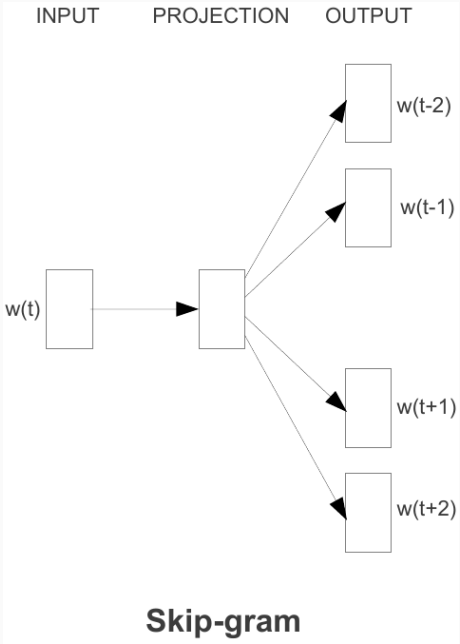
Self-supervised learning

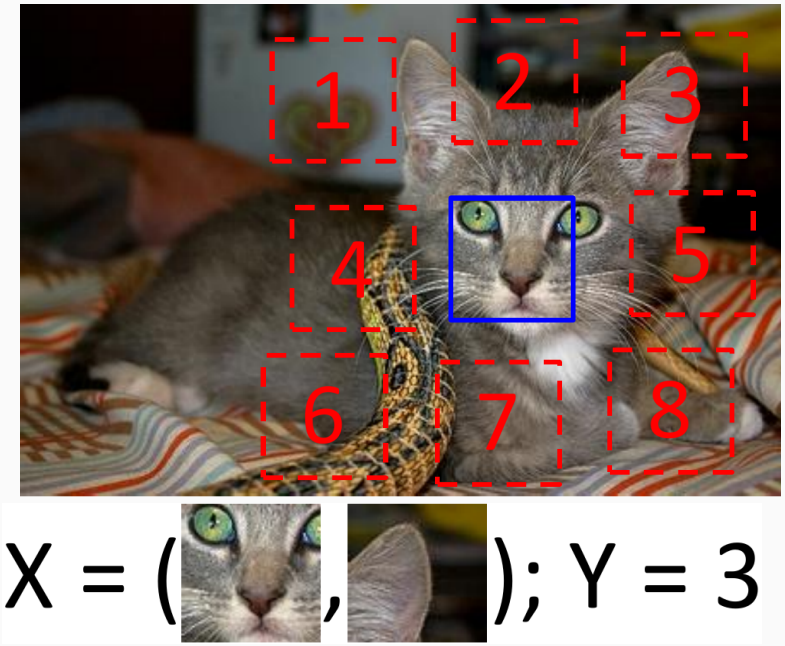
- Learning conditional models $p(y|\mathbf{x})$ from unlabeled data
- Prediction of structural data properties
  - Skip-gram language models (word2vec) [Mikolov et al., 2013]



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]

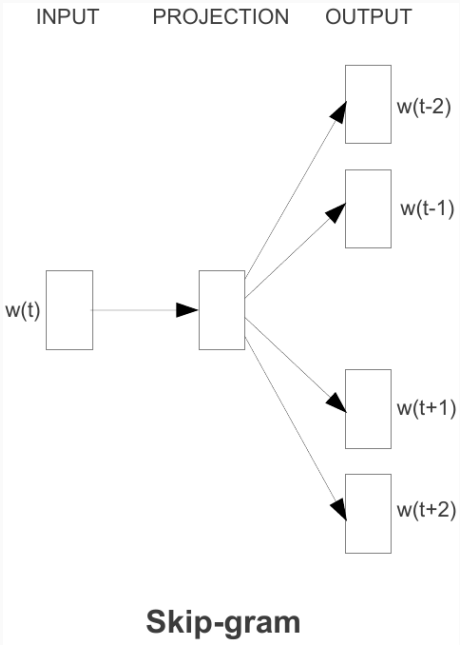


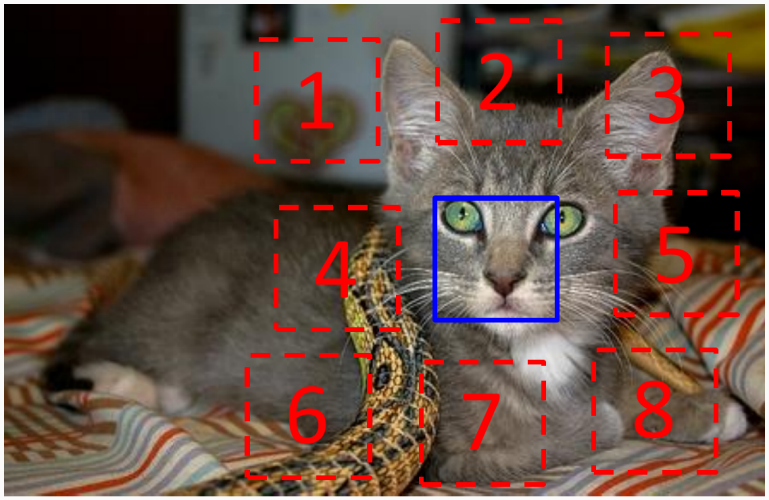


Self-supervised learning

- 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]



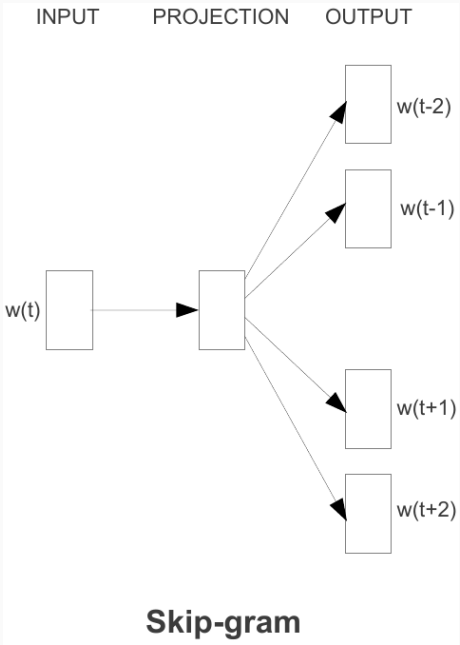


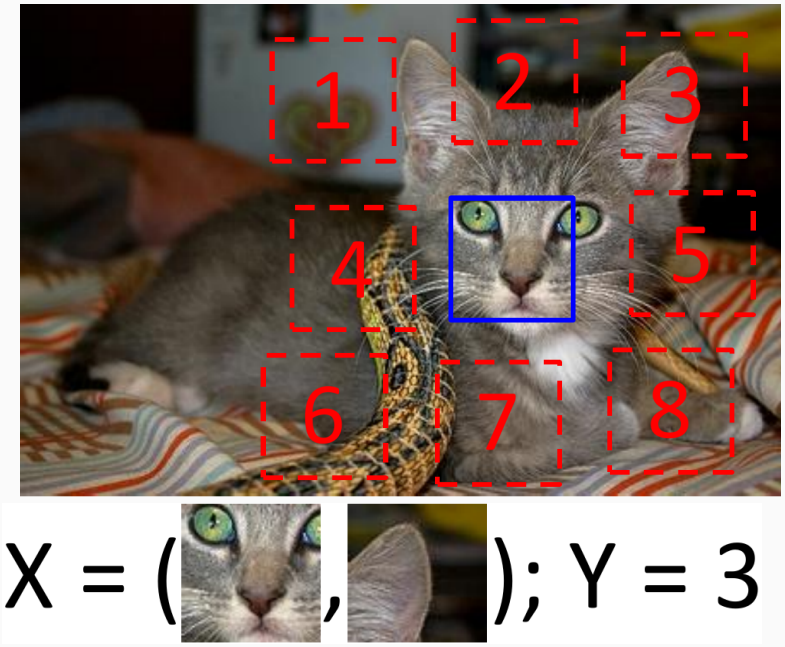
X = (,); Y = 3

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]
  - ...





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
- Unsupervised representation learning

Self-supervised learning to prime supervised learning

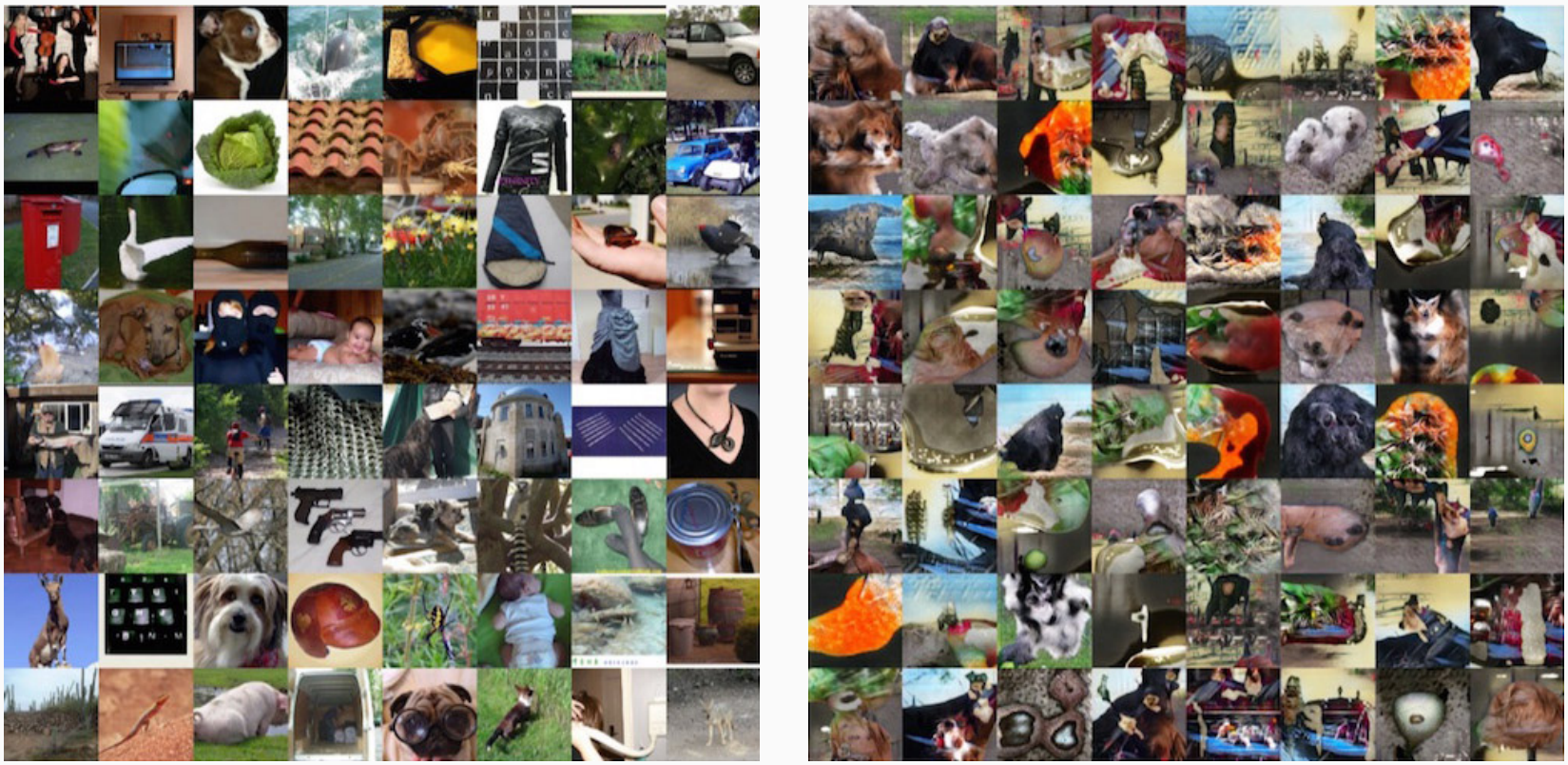
- Supervised pre-training of network on proxy-task
- Fine-tune on final task with limited training data
- Unsupervised representation learning
- Requires a pretext task, does not allow to sample data from model

Part III

Generative models

Generative models

- Unconditional density model $p_\theta(\mathbf{x})$: task is to 'understand' the data
- Parameters estimated from unlabeled data
- Possible to draw samples from model



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

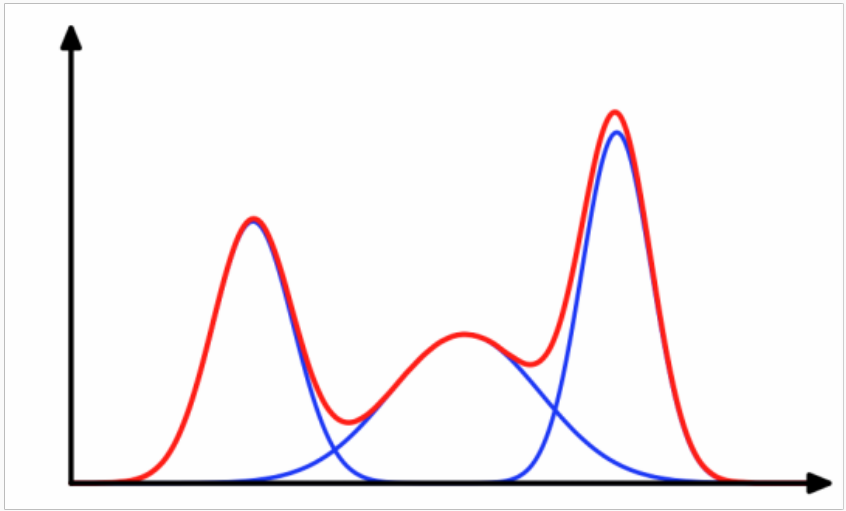
My first generative model

- Gaussian mixture model

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

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

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



My first generative model

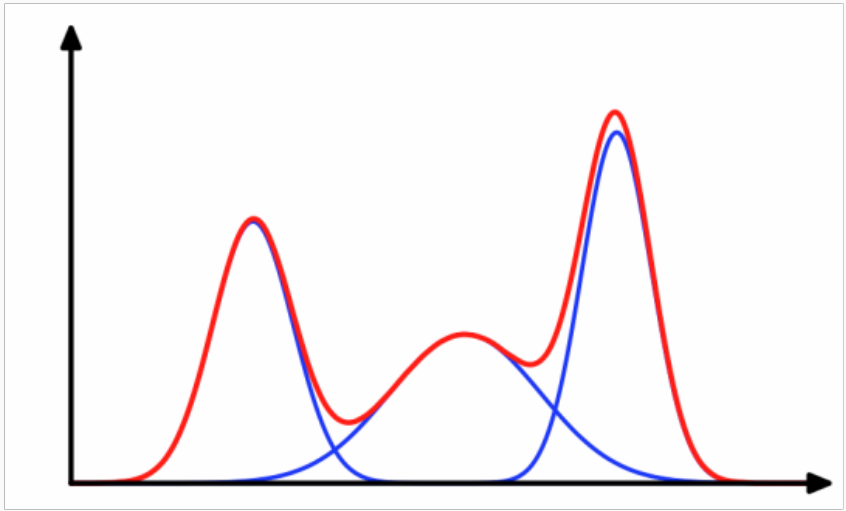
- Gaussian mixture model

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

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

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

- Estimation: Expectation-Maximization (EM) algorithm



My first generative model

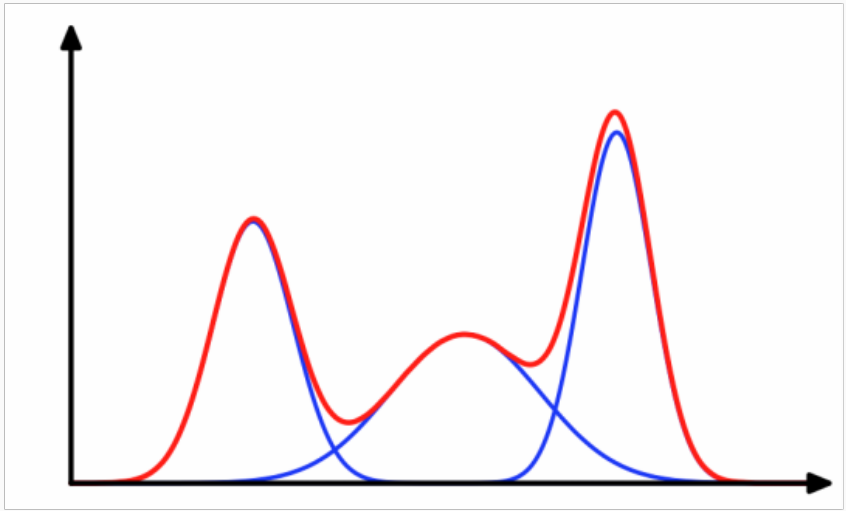
- Gaussian mixture model

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

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

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

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



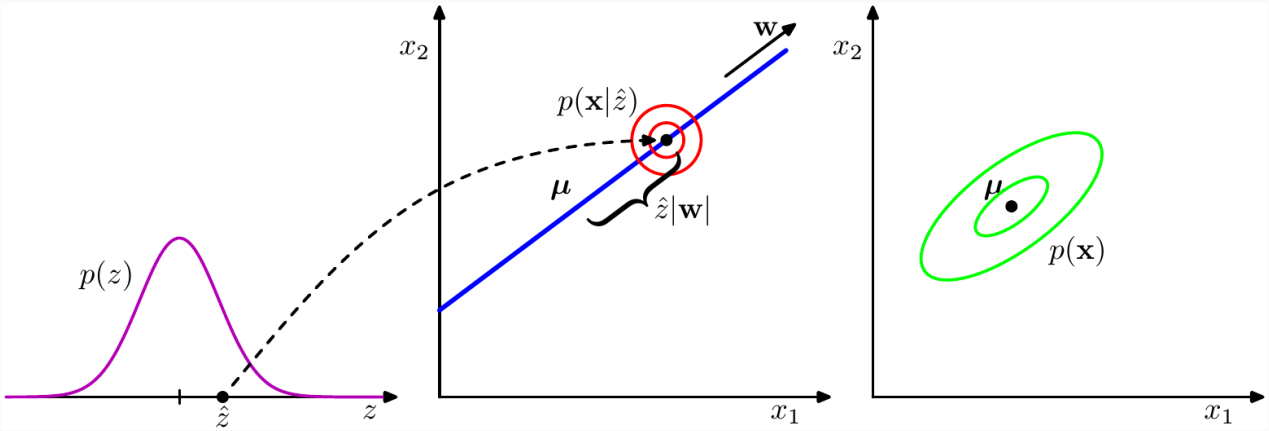
My second generative model

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

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

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

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



My second generative model

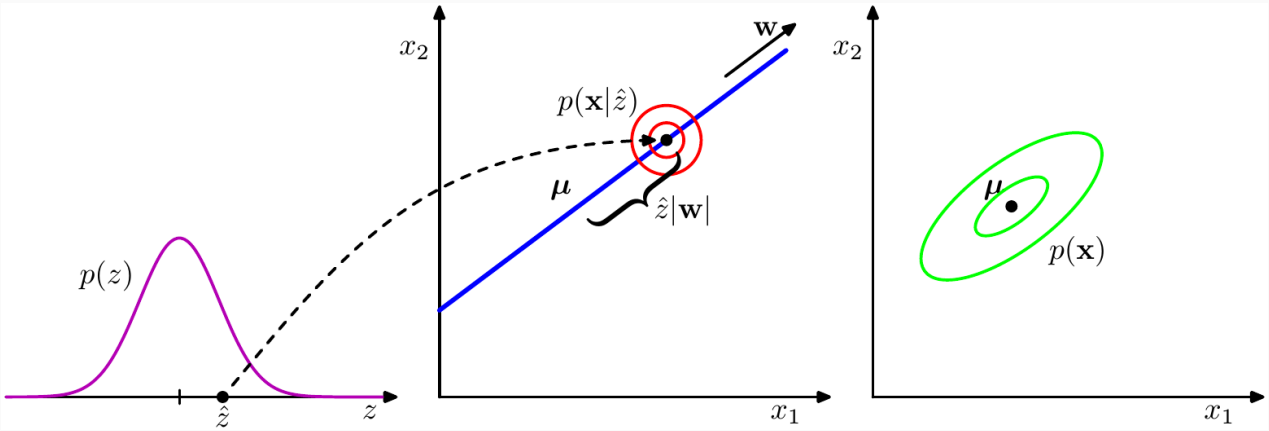
- Probabilistic Principal Component Analysis
  [Roweis, 1997, Tipping and Bishop, 1999]

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

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

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

- Estimation: SVD or EM algorithm



My second generative model

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

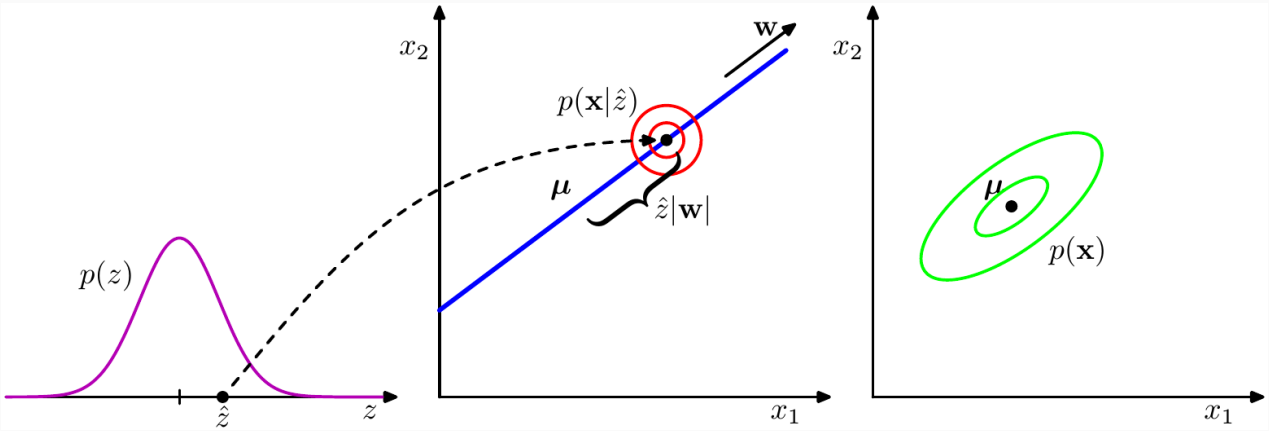
$$p(z) = \mathcal{N}(z; 0, I_d) \tag{6}$$

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

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

- Estimation: SVD or EM algorithm

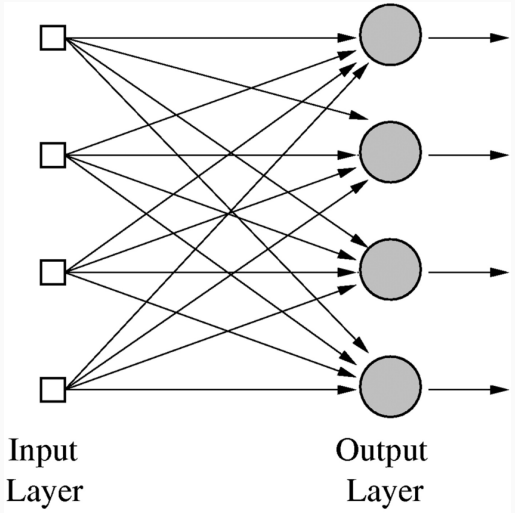
- Sampling: pick point in subspace from prior $p(z)$,
  then draw sample from conditional distribution $p(\mathbf{x}|z)$



Linear latent variable models

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

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



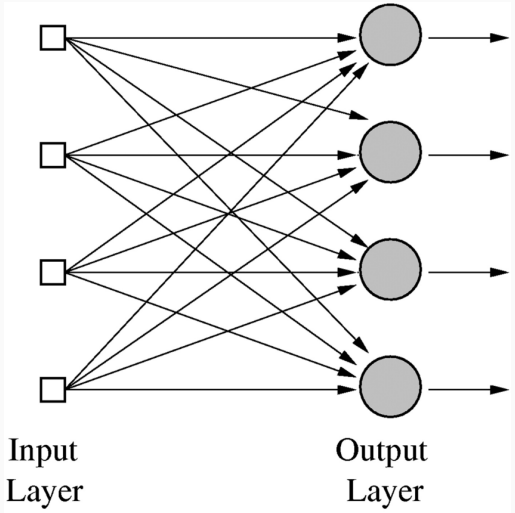
Linear latent variable models

- **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

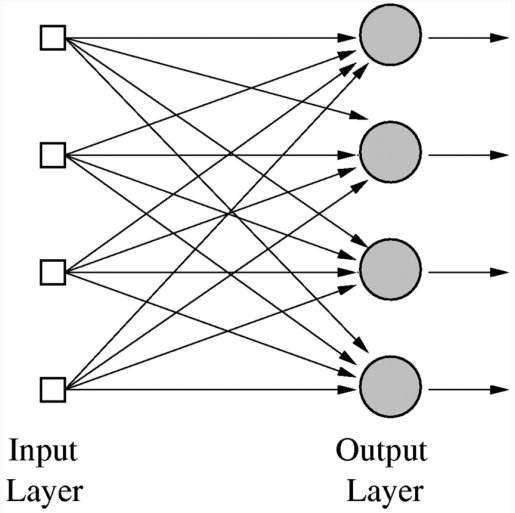
$$\hat{\mathbf{x}} = Wz + \mu \tag{9}$$

- 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

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

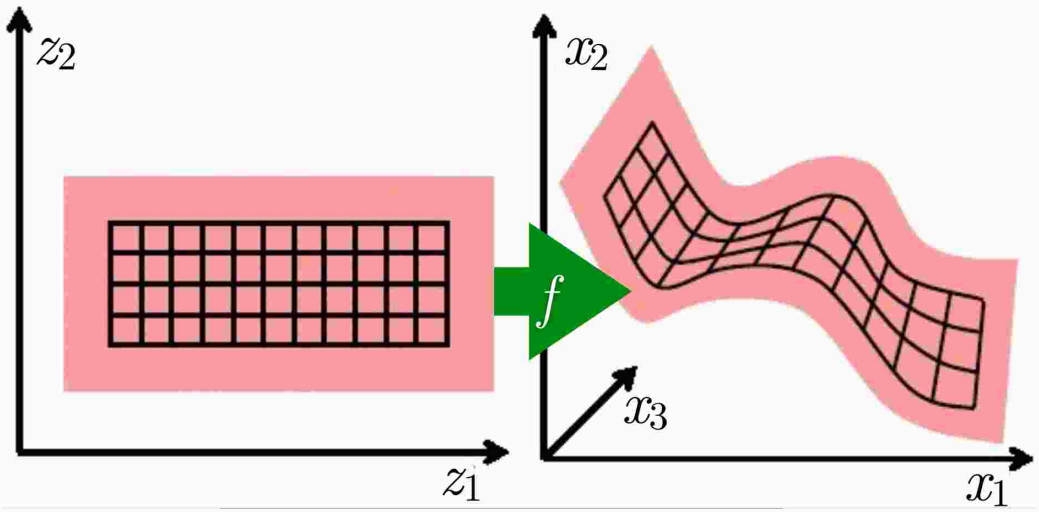


Non-linear latent variable models

- 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



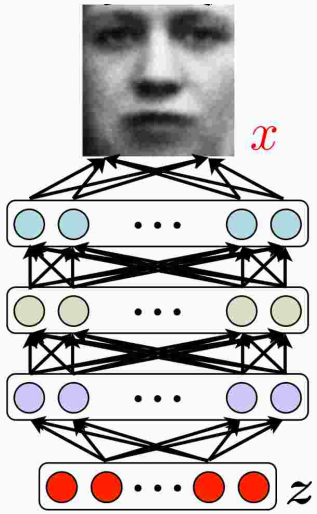
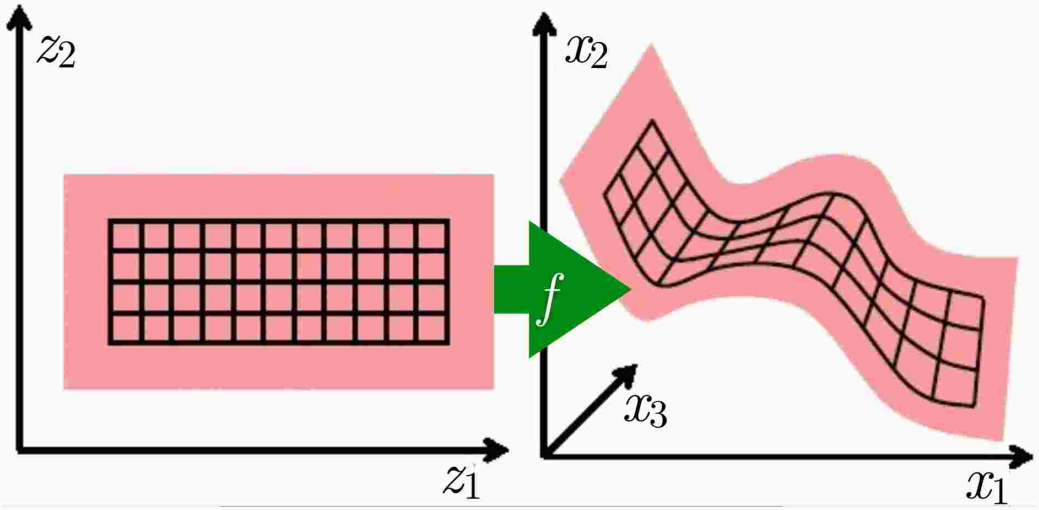


Figure from Aaron Courville

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 $p_\theta(\mathbf{x})$



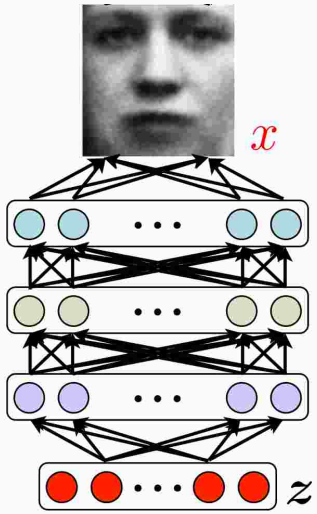


Figure from Aaron Courville

Learning deep latent variable models

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

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.

Learning deep latent variable models

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

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.
- Problem: 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 $\mathbf{z}$

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.
- Problem: 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 $\mathbf{z}$

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.
- Problem: Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$
- i) Approximate integral: Variational autoencoders (VAE)

Learning deep latent variable models

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

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.
- Problem: Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$
- i) Approximate integral: Variational autoencoders (VAE)
- ii) Avoid using the integral: Generative adversarial networks (GAN)

Learning deep latent variable models

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

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.
- Problem: Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$
- i) Approximate integral: Variational autoencoders (VAE)
- ii) Avoid using the integral: Generative adversarial networks (GAN)
- iii) Constrain f_θ so that we can compute $p_\theta(x)$ (e.g. Real-NVP)

Learning deep latent variable models

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

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

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

- Property: Arbitrarily flexible $p_\theta(x)$.
- Problem: Evaluation of $p_\theta(\mathbf{x})$ intractable due to integral involving non-linear deep net $f_\theta(\cdot)$
- i) Approximate integral: Variational autoencoders (VAE)
- ii) Avoid using the integral: Generative adversarial networks (GAN)
- iii) Constrain f_θ so that we can compute $p_\theta(x)$ (e.g. Real-NVP)
- iv) Do not use latent variables (e.g. PixelCNN)

Part IV

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 $p^*(\mathbf{x})$, 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 $p^*(\mathbf{x})$, use classifier D_ϕ
  - $D_\phi(\mathbf{x}) \in [0, 1]$ probability $\mathbf{x}$ is real **vs**. synth. image

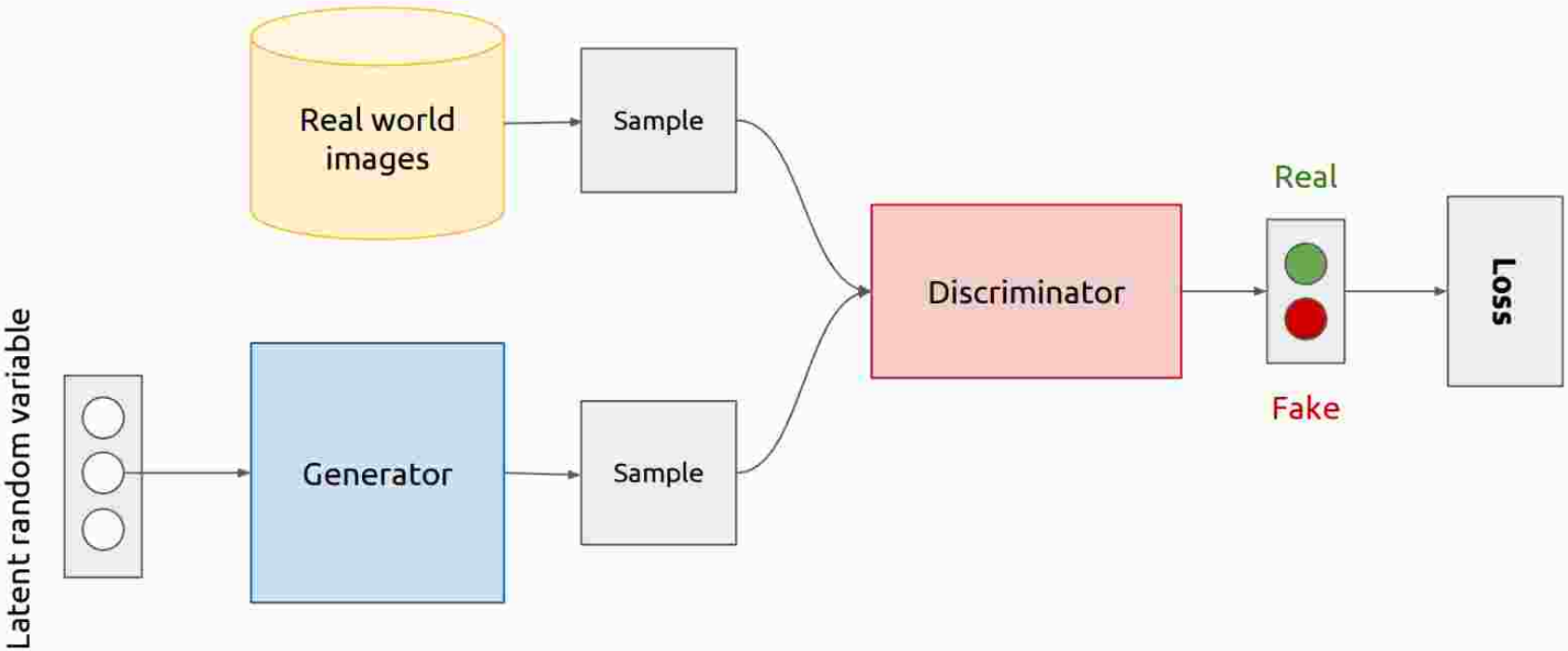


Figure from Kevin McGuinness

Discriminator architecture for images

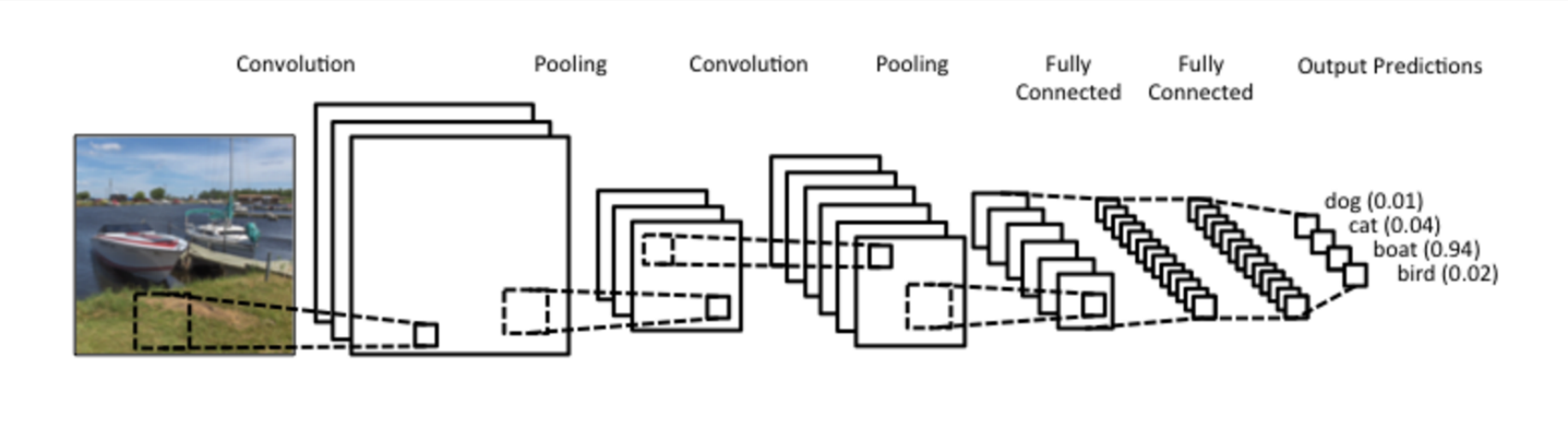


Figure from Kevin McGuinness

- 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)

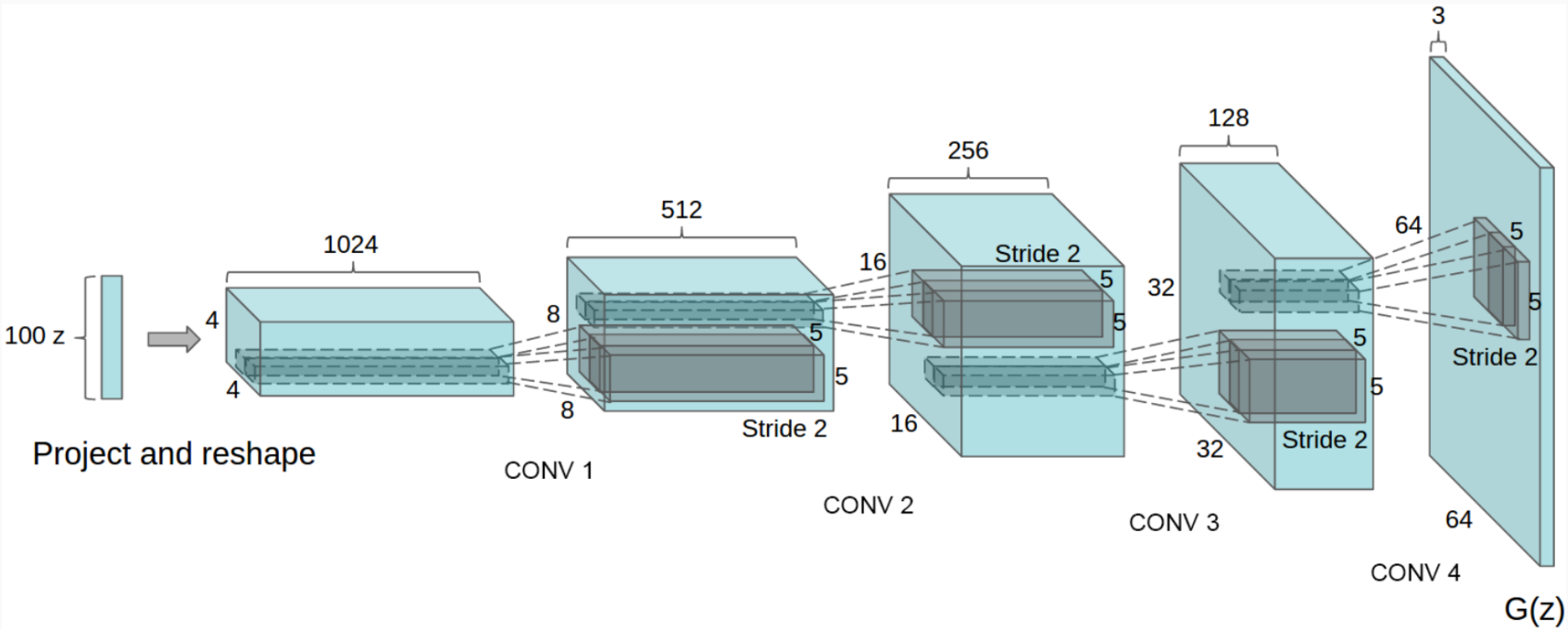


Figure from OpenAI

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

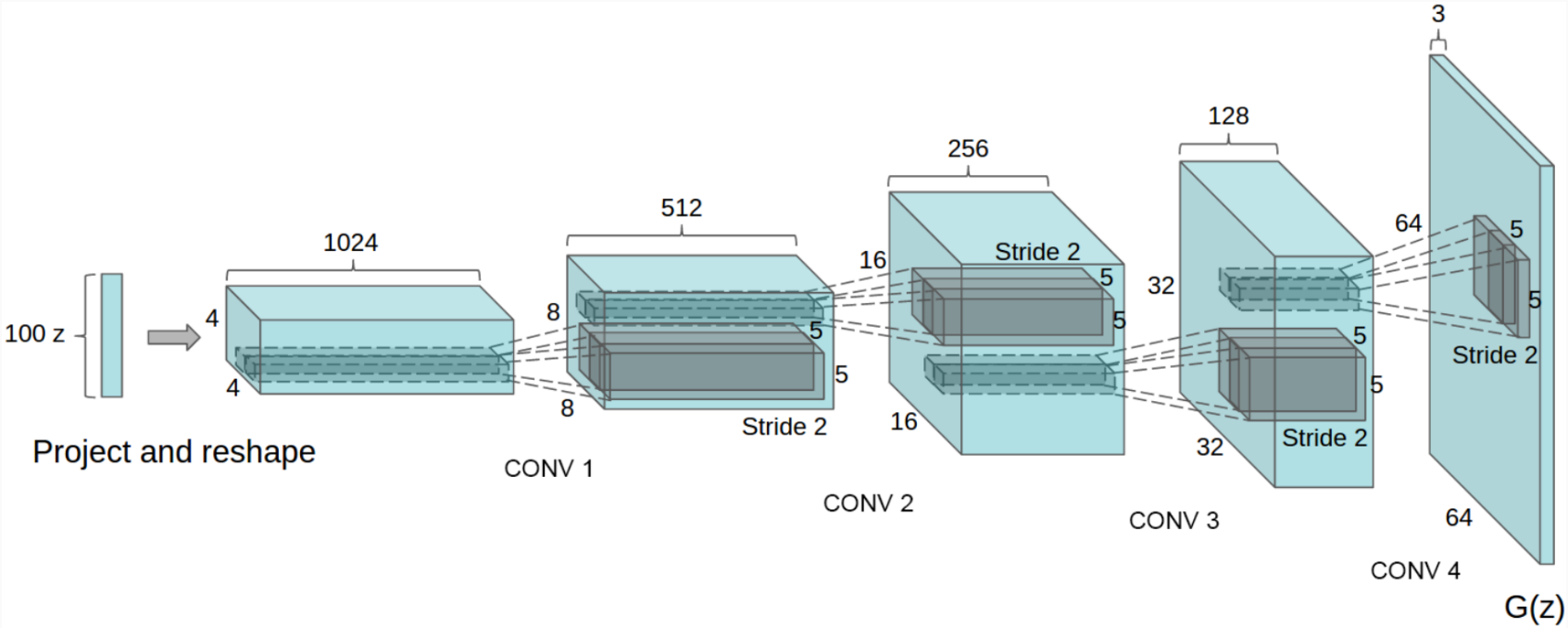
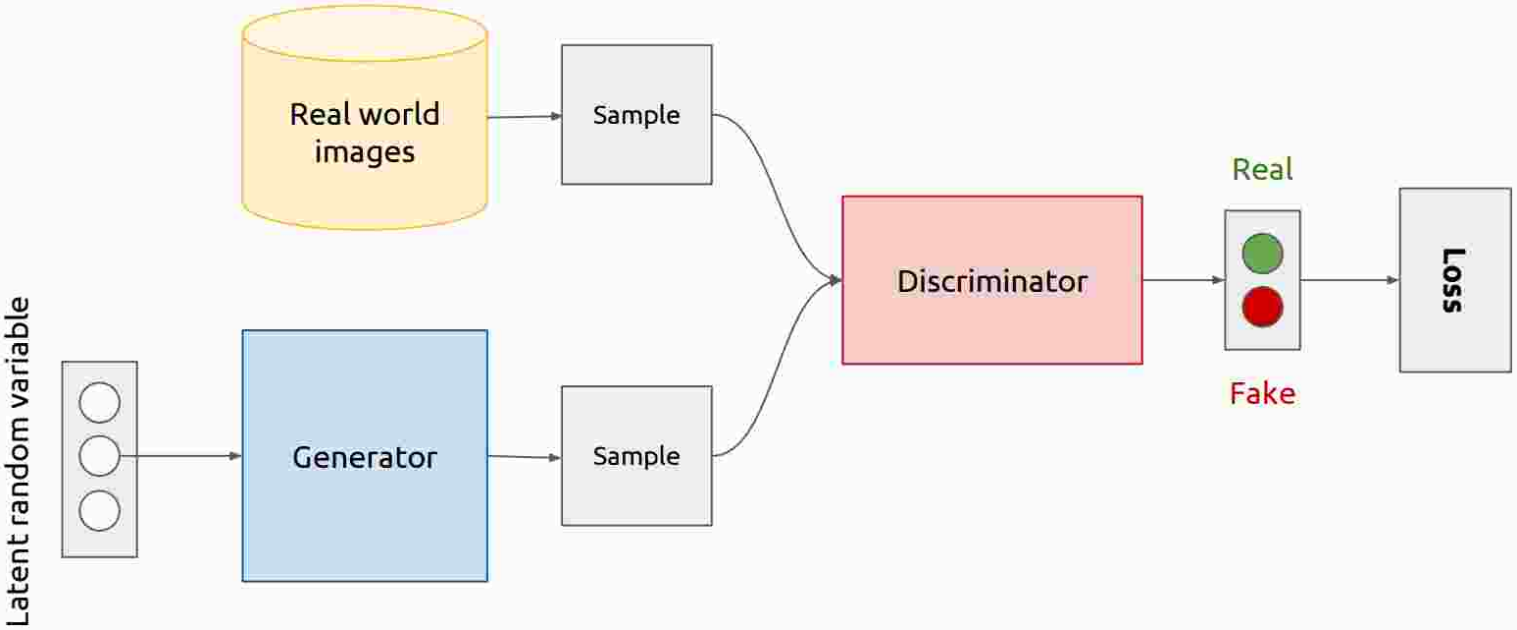


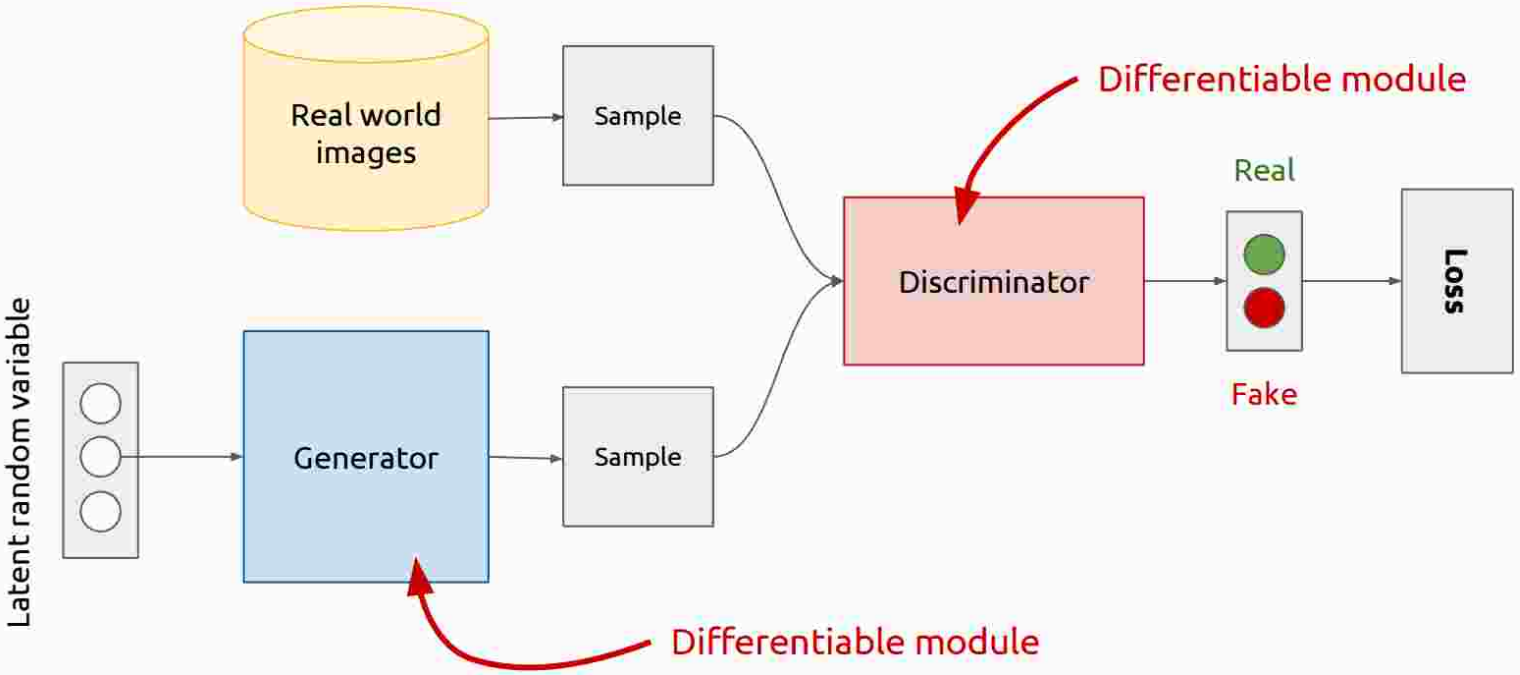
Figure from OpenAI

Training GANs



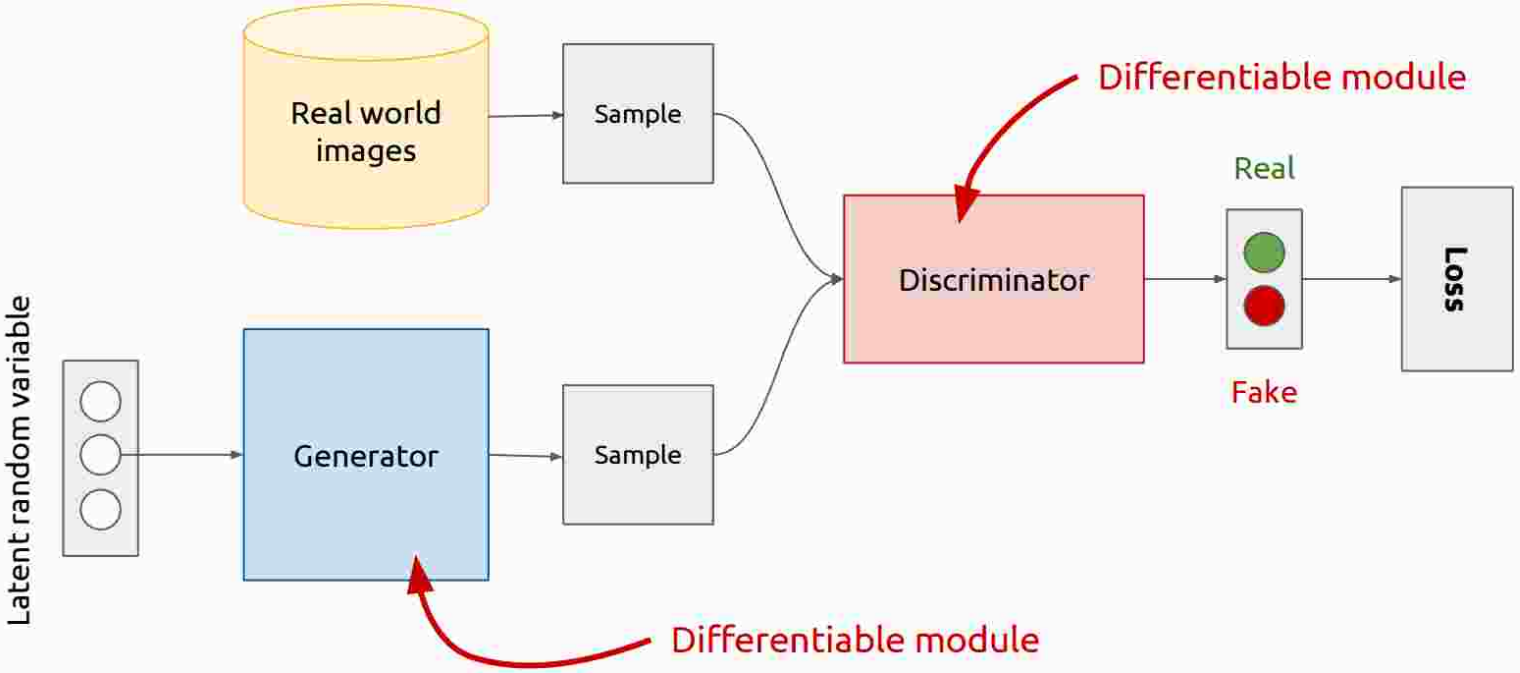
- Discriminator: maximize classification for a given generator
- Generator: degrade classification of a given discriminator

Training GANs



- Discriminator: maximize classification for a given generator
- Generator: degrade classification of a given discriminator
- Samples **z** pass through two differentiable modules

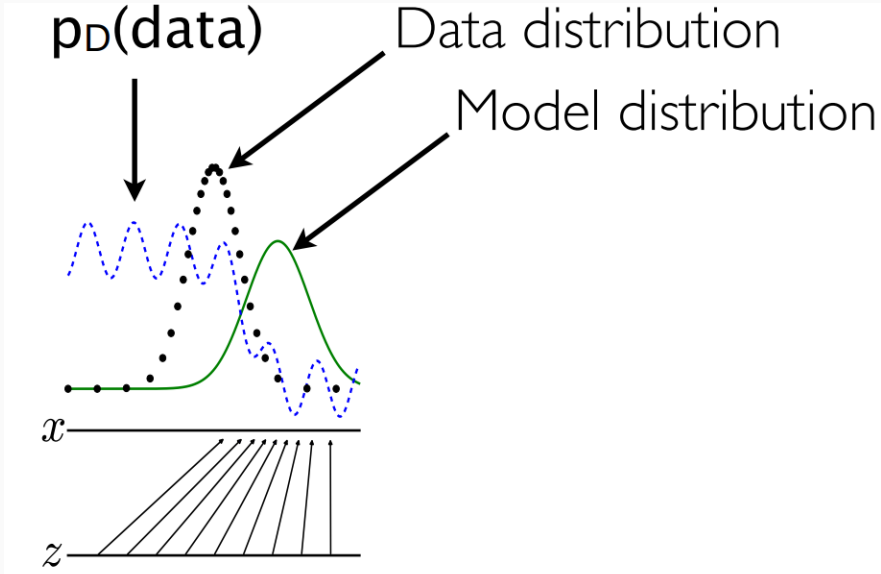
Training GANs



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

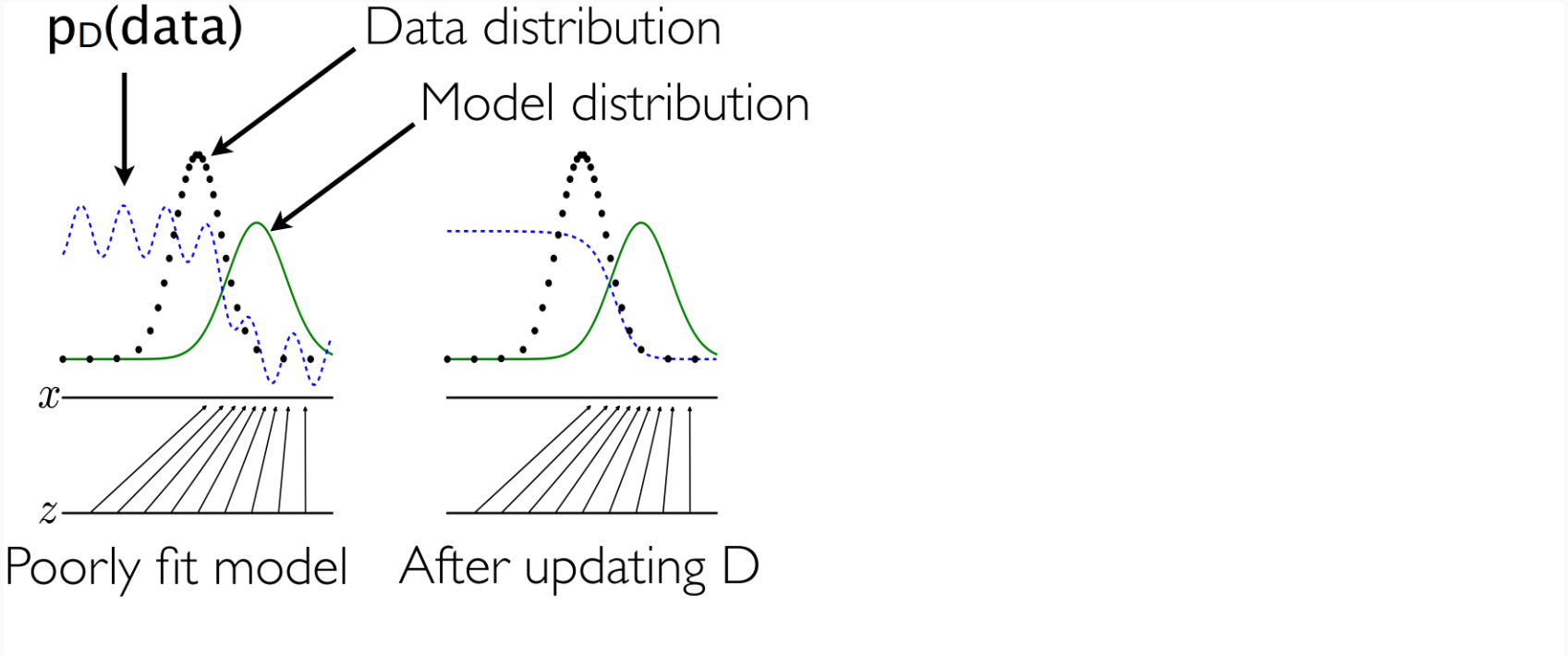
- Samples **z** pass through two differentiable modules
- Discriminator acts as **trainable loss function**

GAN learning process

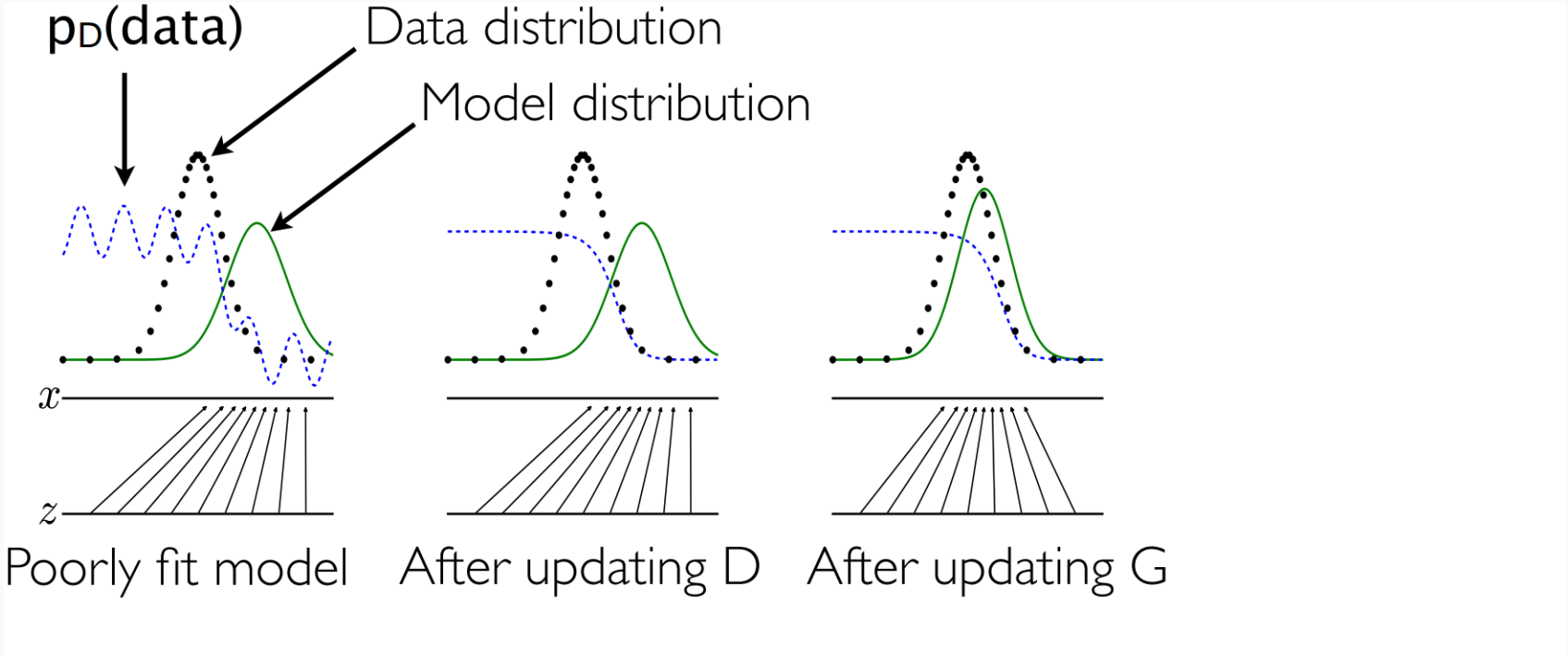


Poorly fit model

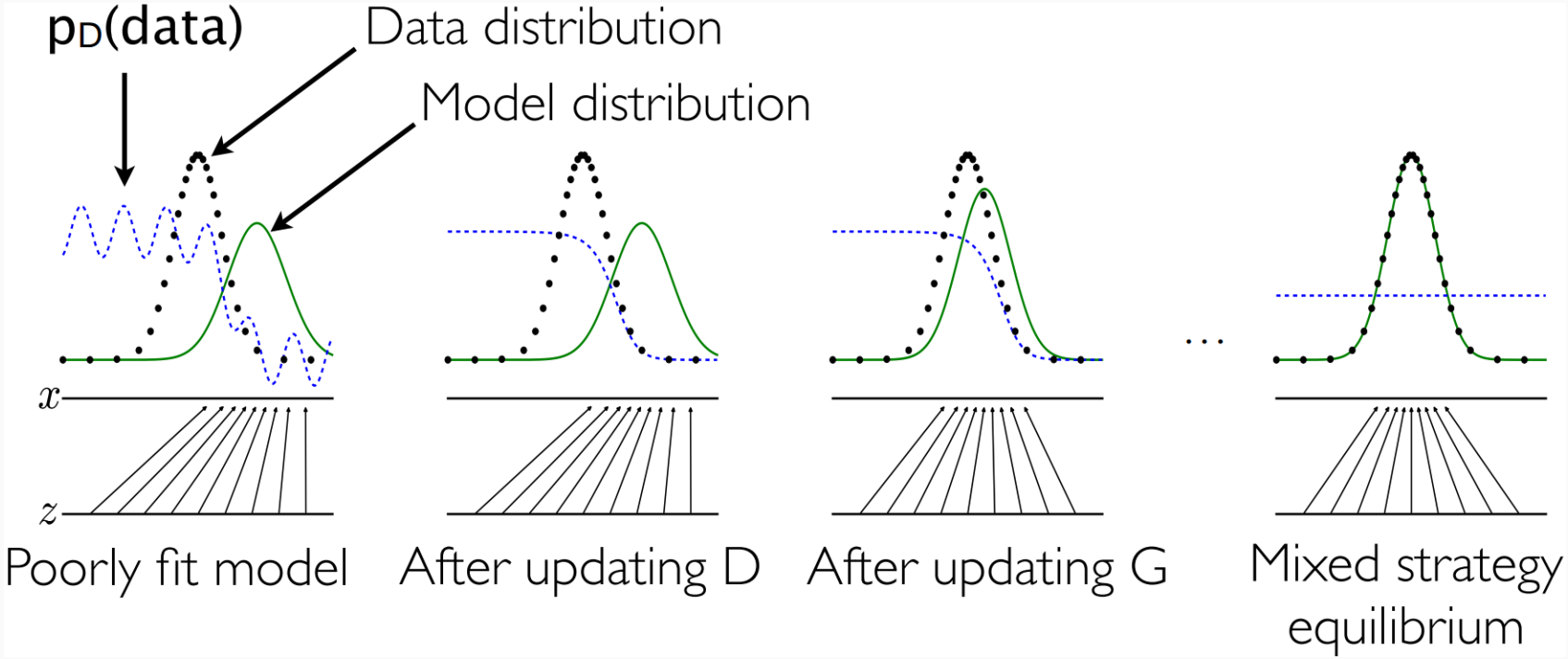
GAN learning process



GAN learning process



GAN learning process



GAN Optimization problem

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

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

GAN Optimization problem

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

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

$$\min_\theta \max_\phi V(\phi, \theta)$$

GAN Optimization problem

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

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

$$\min_\theta \max_\phi V(\phi, \theta)$$

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

GAN Optimization problem

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

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

$$\min_\theta \max_\phi V(\phi, \theta)$$

- 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 $V(\phi, \theta)$: performance of discriminator

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

$$\min_\theta \max_\phi V(\phi, \theta)$$

- 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

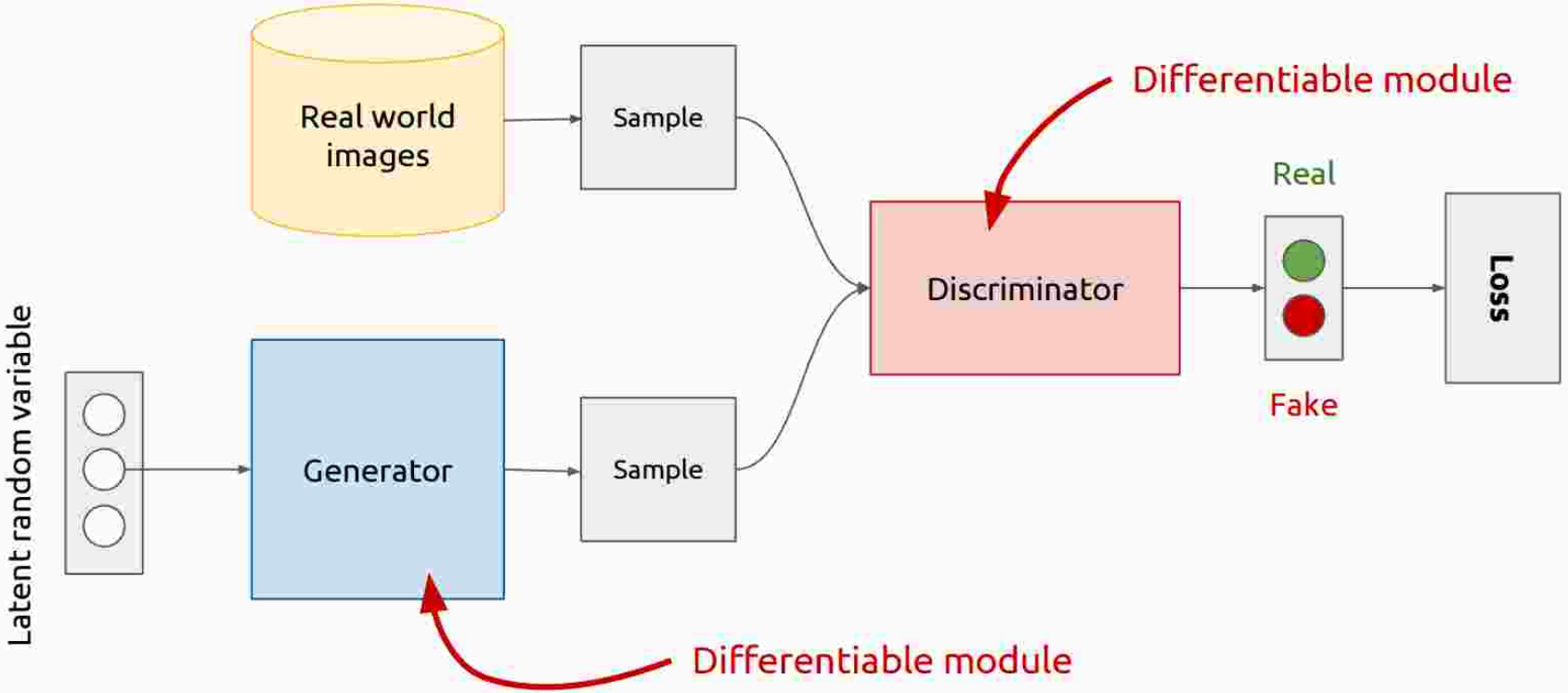
$$\max_D V(D, G) = -\ln 4 + 2D_{JS}(p_{\text{data}}||p_G)$$

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

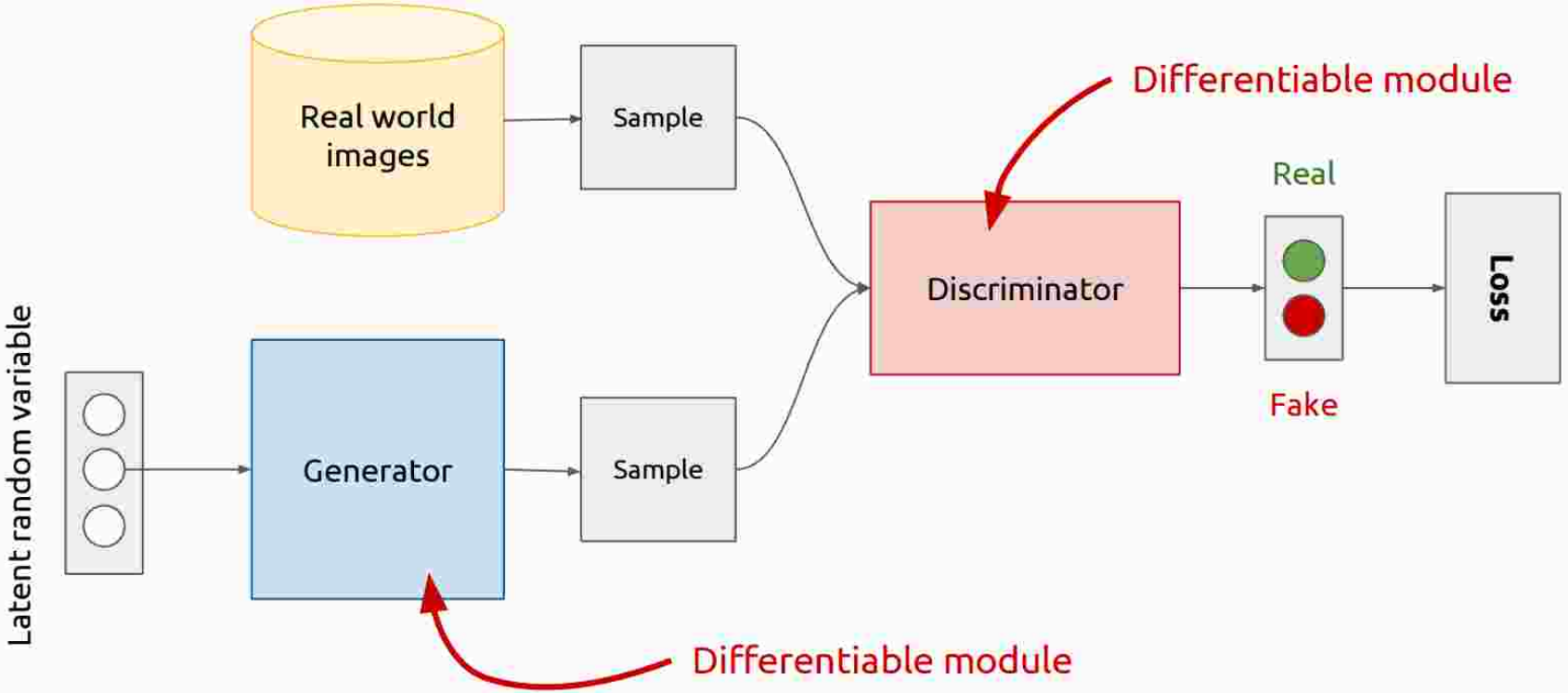
Training GANs in practice



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

- 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]

Modern network on ImageNet, class conditional

Examples taken from Brock et al. 2019

GAN generalizes beyond training data

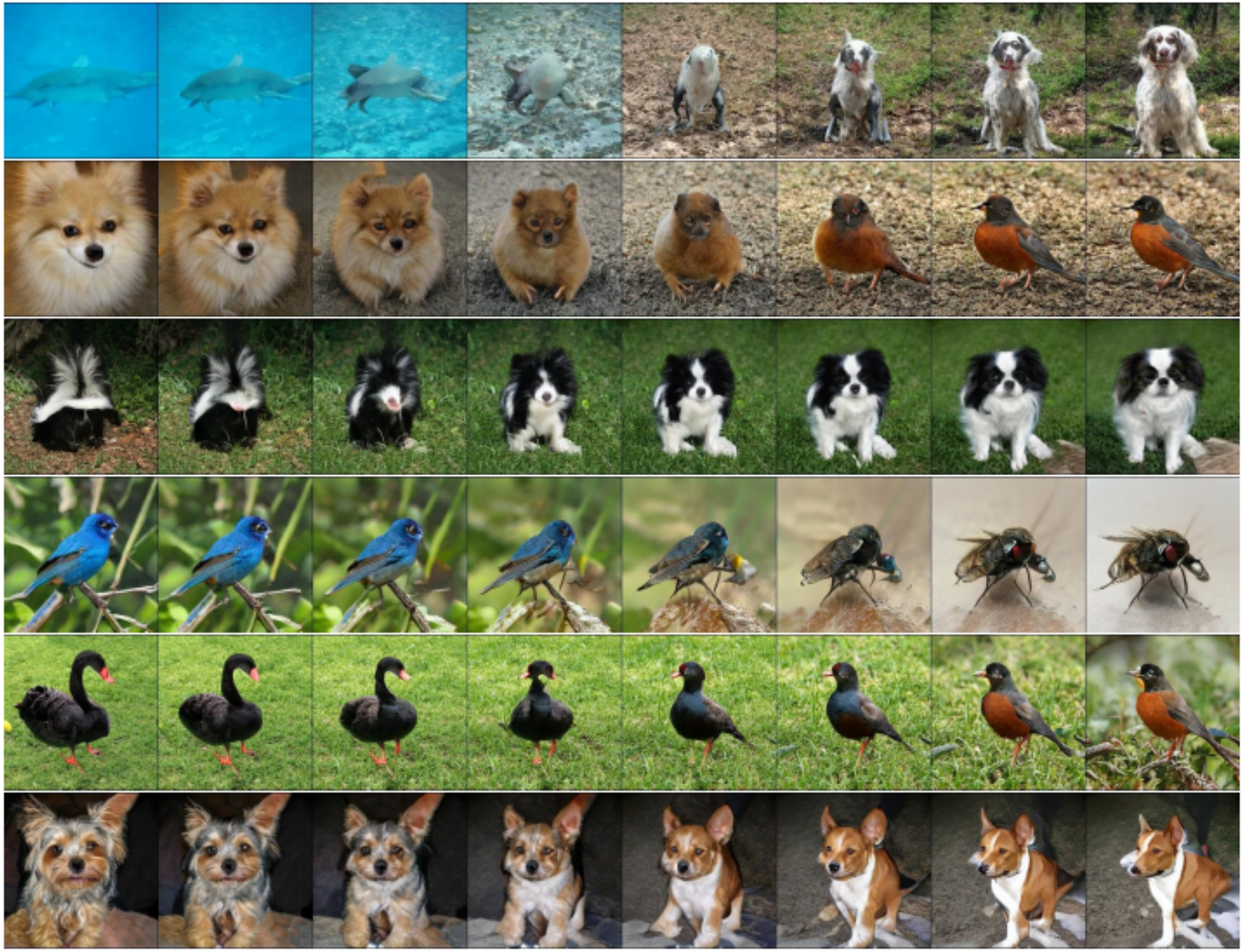
- Sample along linear trajectory in latent space $z_1 \rightarrow z_2$
- Smooth transitions suggest generalization,
  sharp transitions would suggest literal memorization

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

GAN generalizes beyond training data



Vector arithmetic on latent variables

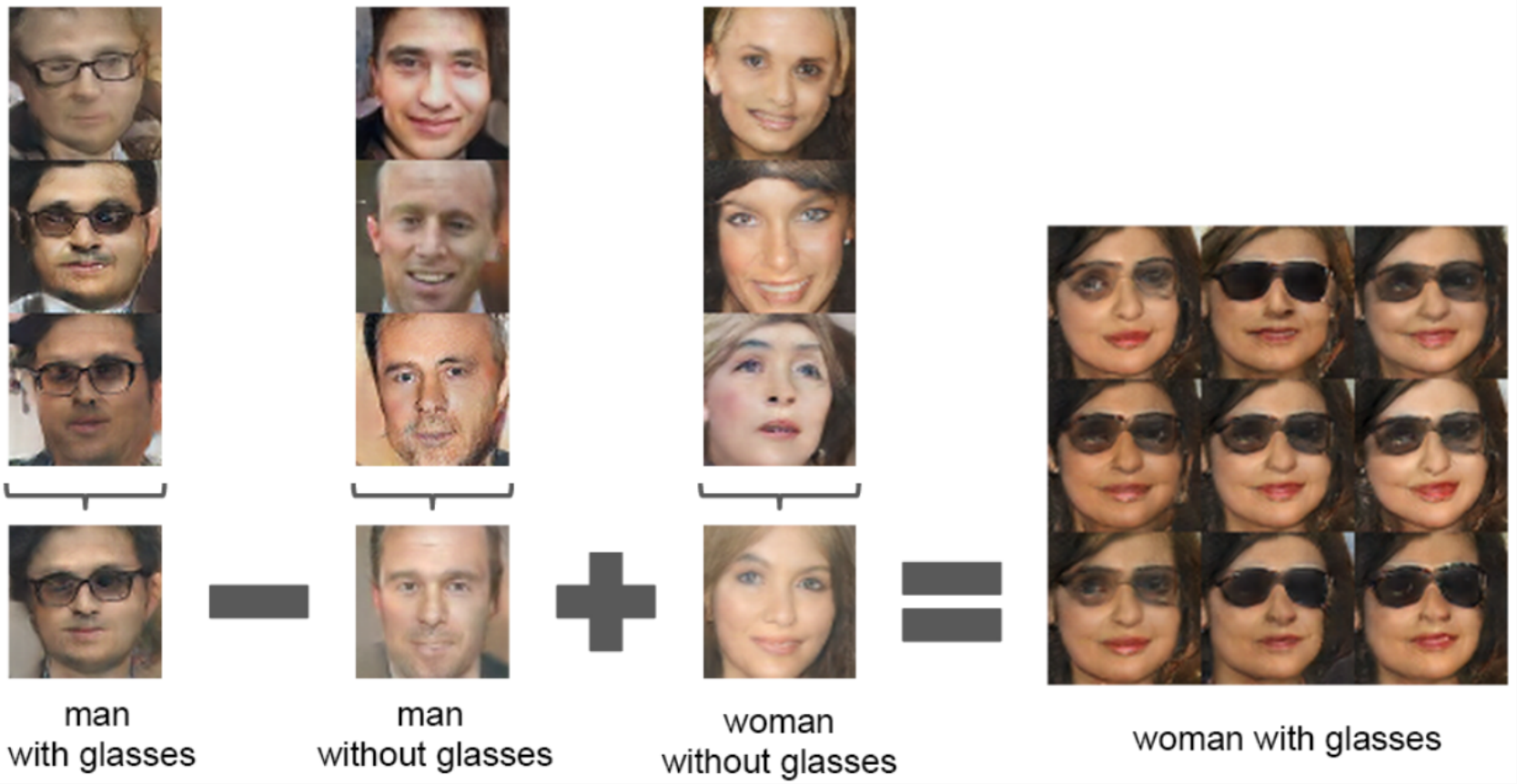
- embedding shows semantic regularities

Vector arithmetic on latent variables

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

Vector arithmetic on latent variables

- embedding shows semantic regularities
- Consider GAN trained on human faces, average **z** vectors over three samples for stability



Quantitative evaluation of adversarial networks

- By construction intractable to compute $p_G(x^*)$, in particular for points in a test set
- Approximate value of $p_G(x^*)$ with Parzen window estimator using samples $x_l \sim p_G(x)$, see [Breuleux et al., 2011]

$$p_{\text{parzen}}(x^*) = \frac{1}{L} \sum_{l=1}^{L} \mathcal{N}(x^*; x_l, \sigma_D I) \tag{15}$$

  Works poorly in high dimension

Quantitative evaluation of adversarial networks

- By construction intractable to compute $p_G(x^*)$, in particular for points in a test set
- Approximate value of $p_G(x^*)$ with Parzen window estimator using samples $x_l \sim p_G(x)$, see [Breuleux et al., 2011]

$$p_{\text{parzen}}(x^*) = \frac{1}{L} \sum_{l=1}^{L} \mathcal{N}(x^*; x_l, \sigma_D I) \tag{15}$$

  Works poorly in high dimension

- This is a sample approximation to a **different** generative model

$$p(z) = \mathcal{N}(z; 0, I) \tag{16}$$

$$p(x|z) = \mathcal{N}(x; G(z), \sigma_D I) \tag{17}$$

Issues training GANs in practice

- GANs known to be difficult to train in practice
  - Formulated as mini-max objective between two networks
  - Optimization can oscillate between solutions
  - Picking "compatible" generator and discriminator architectures
  - Training fails if the discriminator is 'too good'
- Mode collapse: failure to capture part of training data
- Quantitative evaluation not aligned with objective function

Part V

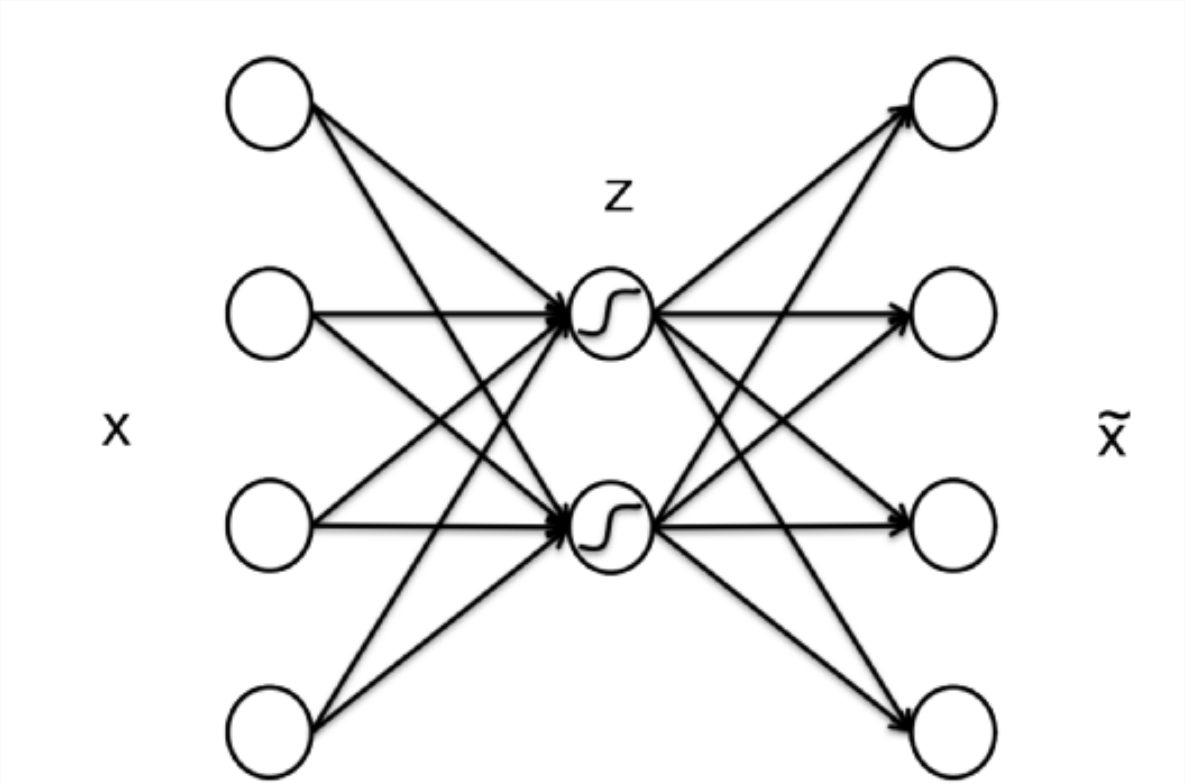
Variational Autoencoders

Autoencoders

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

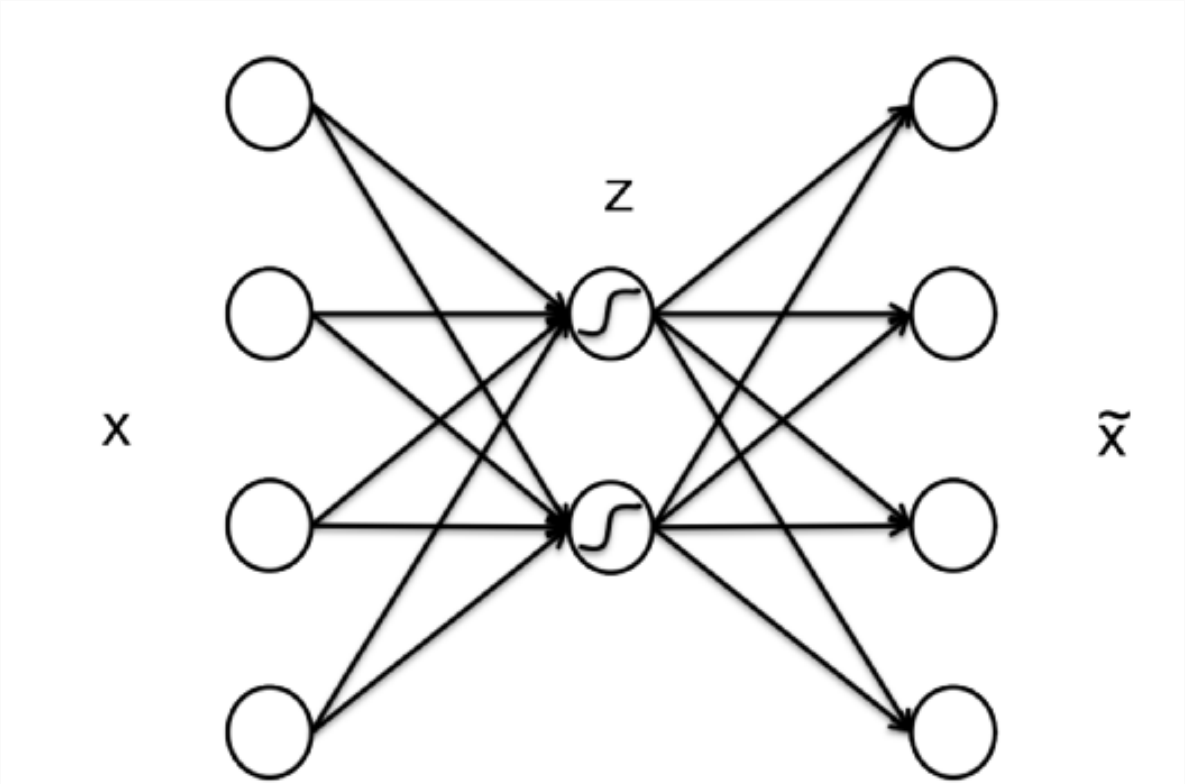
Autoencoders

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



Autoencoders

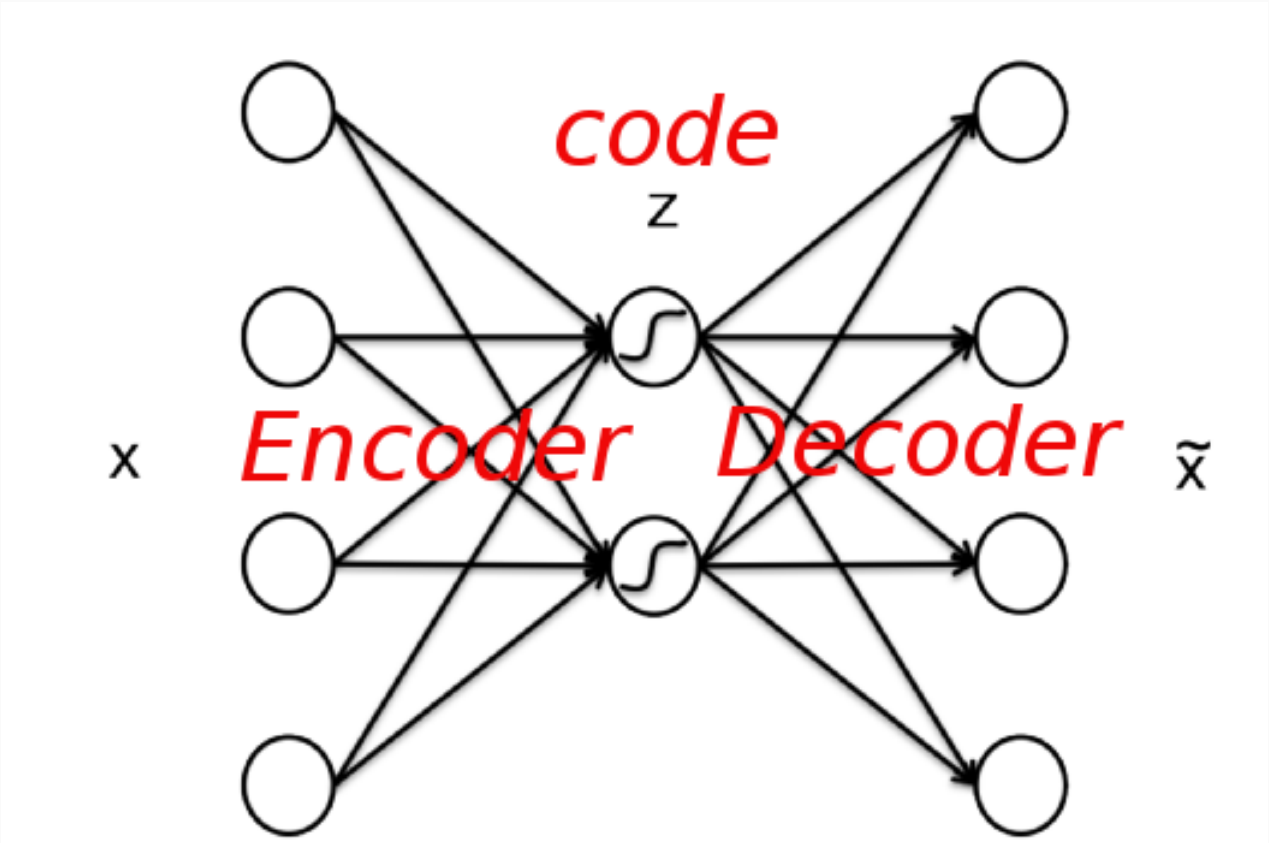
- Learn latent representation **z** via reconstruction of data **x**
- Neural network where output $\sim$ input
  - Encoder: maps data **x** to latent code **z**
  - Decoder: maps latent code **z** to reconstruction $\tilde{\mathbf{x}}$
- Loss minimizes discrepancy between **x** and $\tilde{\mathbf{x}}$



Relation autoencoders and PCA [Baldi and Hornik, 1989]

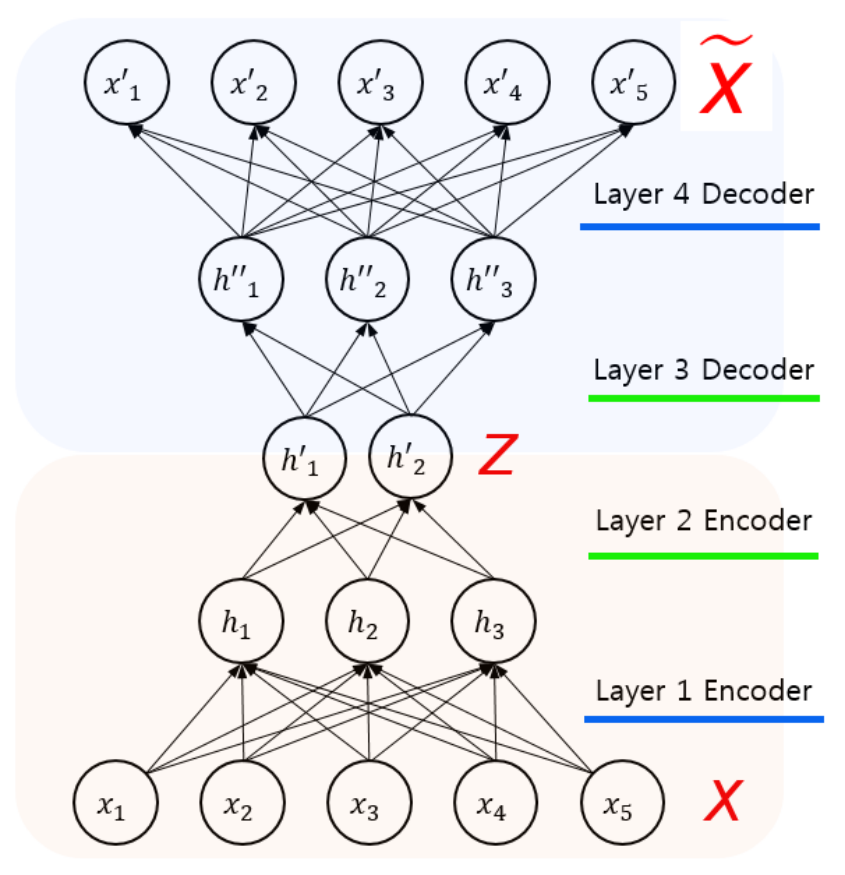
- Autoencoder recovers PCA if
  1. Encoder and decoder are both linear
  2. Optimizing ℓ_2 reconstruction loss

$$\min_{V,W} \frac{1}{2N} \sum_{n=1}^{N} ||x_n - VWx_n||^2 \tag{18}$$



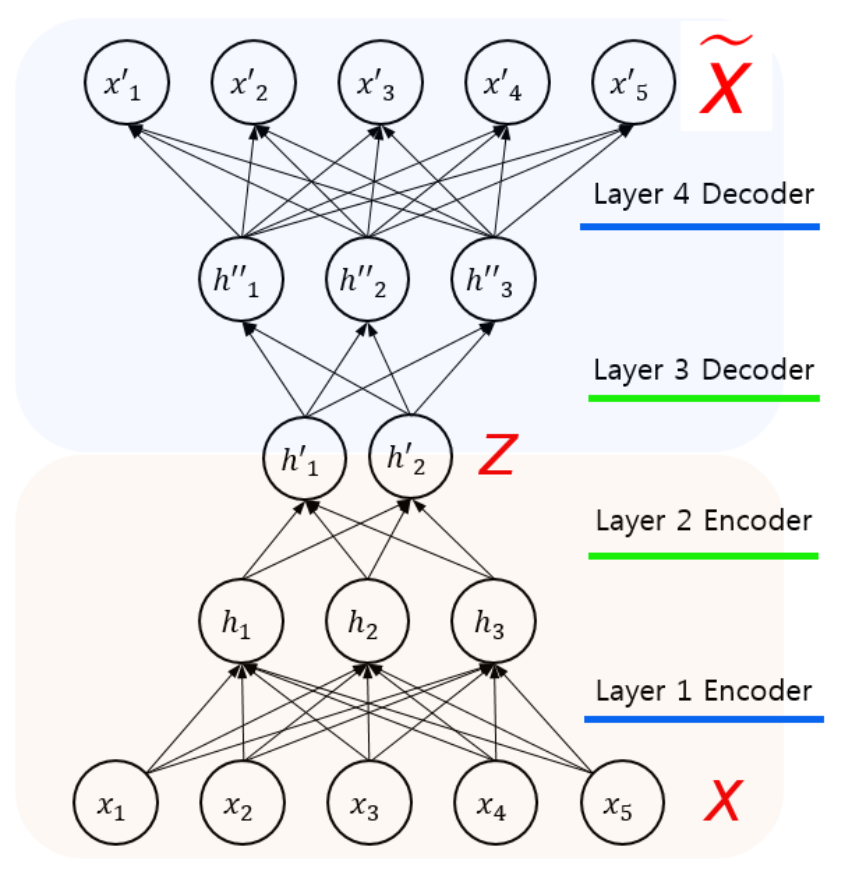
Deep non-linear autoencoders

- Stack many non-linear layers in encoder and decoder



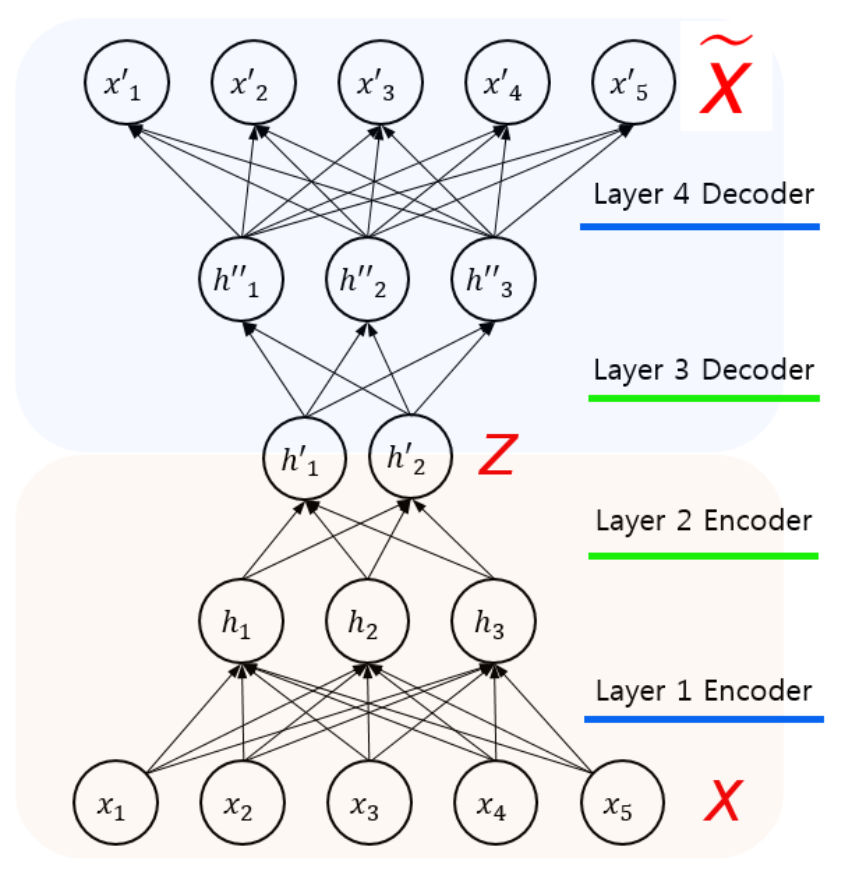
Deep non-linear autoencoders

- 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 $\mathbf{x}$

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta^\mu(\mathbf{z}), f_\theta^\sigma(\mathbf{z})) \tag{19}$$

Autoencoding variational Bayes [Kingma and Welling, 2014]

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

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta^\mu(\mathbf{z}), f_\theta^\sigma(\mathbf{z})) \tag{19}$$

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

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; g_\phi^\mu(\mathbf{x}), g_\phi^\sigma(\mathbf{x})) \tag{20}$$

Autoencoding variational Bayes [Kingma and Welling, 2014]

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

$$p_\theta(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta^\mu(\mathbf{z}), f_\theta^\sigma(\mathbf{z})) \tag{19}$$

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

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; g_\phi^\mu(\mathbf{x}), g_\phi^\sigma(\mathbf{x})) \tag{20}$$

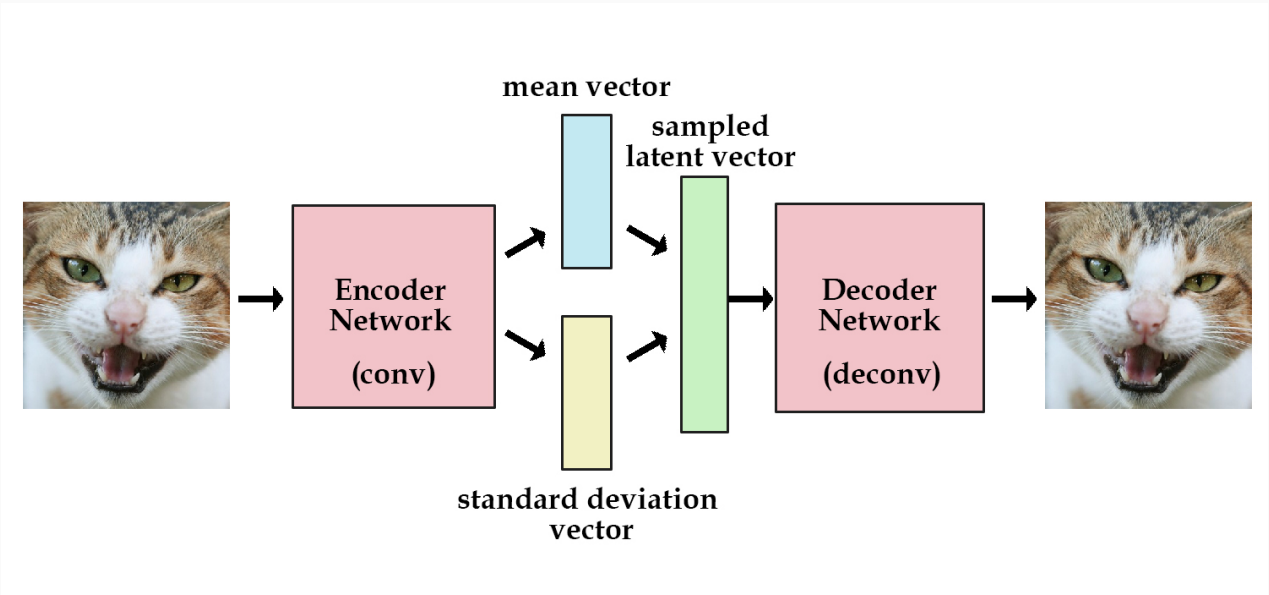


Figure from kvfrans@github

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{21}$$

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{21}$$

- Idea 0: Monte-Carlo estimation. Problem: high dimensional

Objective function: Evidence lower bound (ELBO)

- Quantity of interest: marginal likelihood or "evidence"

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \tag{21}$$

- Idea 0: Monte-Carlo estimation. Problem: high dimensional
- Idea 1: Weighted sampling

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

Objective function: Evidence lower bound (ELBO)

- Idea 2: Efficient estimation with the ELBO

(23)

(24)

(25)

Objective function: Evidence lower bound (ELBO)

- Idea 2: Efficient estimation with the ELBO

$$\ln(p_\theta(\mathbf{x})) = \ln\left(\int_{\mathbf{z}} q_\phi(z|x) p_\theta(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q_\phi(z|x)} dz\right) \tag{23}$$

(24)

(25)

Objective function: Evidence lower bound (ELBO)

- Idea 2: Efficient estimation with the ELBO

$$\ln(p_\theta(\mathbf{x})) = \ln\left(\int_{\mathbf{z}} q_\phi(z|x) p_\theta(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q_\phi(z|x)} dz\right) \tag{23}$$

$$\geq \int_{\mathbf{z}} q_\phi(z|x) \ln\left(p_\theta(x|z) \frac{p(\mathbf{z})}{q_\phi(z|x)}\right) d\mathbf{z} \tag{24}$$

$$\tag{25}$$

Objective function: Evidence lower bound (ELBO)

- Idea 2: Efficient estimation with the ELBO

$$\ln(p_\theta(\mathbf{x})) = \ln\left(\int_{\mathbf{z}} q_\phi(z|x) p_\theta(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q_\phi(z|x)} dz\right) \tag{23}$$

$$\geq \int_{\mathbf{z}} q_\phi(z|x) \ln\left(p_\theta(x|z) \frac{p(\mathbf{z})}{q_\phi(z|x)}\right) d\mathbf{z} \tag{24}$$

$$= \mathbb{E}_{q_\phi(z|x)}[\ln(p_\theta(x|z))] - D_{KL}(q_\phi(z|x)||p(z)) \tag{25}$$

Objective function: Evidence lower bound (ELBO)

- Idea 2: Efficient estimation with the ELBO

$$\ln(p_\theta(\mathbf{x})) = \ln\left(\int_{\mathbf{z}} q_\phi(z|x) p_\theta(\mathbf{x}|\mathbf{z}) \frac{p(\mathbf{z})}{q_\phi(z|x)} dz\right) \tag{23}$$

$$\geq \int_{\mathbf{z}} q_\phi(z|x) \ln\left(p_\theta(x|z) \frac{p(\mathbf{z})}{q_\phi(z|x)}\right) d\mathbf{z} \tag{24}$$

$$= \mathbb{E}_{q_\phi(z|x)}[\ln(p_\theta(x|z))] - D_{KL}(q_\phi(z|x)||p(z)) \tag{25}$$

- ELBO becomes function of inference net and generative net

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \tag{26}$$

Objective function: Evidence lower bound (ELBO)

Comments on the ELBO:

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{27}$$

Objective function: Evidence lower bound (ELBO)

Comments on the ELBO:

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{27}$$

- Has an auto-encoder interpretation.

Objective function: Evidence lower bound (ELBO)

Comments on the ELBO:

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{27}$$

- Has an auto-encoder interpretation.
- Efficient computations, at the cost of approximation.

Objective function: Evidence lower bound (ELBO)

Comments on the ELBO:

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi(z|x)}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big)}_{\text{Regularization}} \tag{27}$$

- Has an auto-encoder interpretation.
- Efficient computations, at the cost of approximation.
- KL divergence: non-negative, and zero if and only if $q = p$. Balance between both terms

Objective function: Evidence lower bound (ELBO)

Re-writing the ELBO:

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

(31)

Objective function: Evidence lower bound (ELBO)

Re-writing the ELBO:

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

Using Bayes rule yields $p(z) = \frac{p(x)p(z|x)}{p(x|z)}$ and:

(31)

Objective function: Evidence lower bound (ELBO)

Re-writing the ELBO:

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

Using Bayes rule yields $p(z) = \frac{p(x)p(z|x)}{p(x|z)}$ and:

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x}|\mathbf{z})}{p(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})} \right) \tag{29}$$

(31)

Objective function: Evidence lower bound (ELBO)

Re-writing the ELBO:

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

Using Bayes rule yields $p(z) = \frac{p(x)p(z|x)}{p(x|z)}$ and:

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x}|\mathbf{z})}{p(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})} \right) \tag{29}$$

$$= \mathbb{E}_{q_\phi}\left[\frac{\ln p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{x})}{\ln p_\theta(\mathbf{x}|\mathbf{z})} \right] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\big) \tag{30}$$

$$\tag{31}$$

Objective function: Evidence lower bound (ELBO)

Re-writing the ELBO:

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

Using Bayes rule yields $p(z) = \frac{p(x)p(z|x)}{p(x|z)}$ and:

$$\begin{aligned} F(\theta, \phi) &= \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - \int_z q_\phi(\mathbf{z}|\mathbf{x}) \log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x}|\mathbf{z})}{p(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})} \right) && (29) \\ &= \mathbb{E}_{q_\phi}\left[\frac{\ln p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{x})}{\ln p_\theta(\mathbf{x}|\mathbf{z})} \right] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\big) && (30) \\ &= \ln p_\theta(\mathbf{x}) - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})\big) && (31) \end{aligned}$$

Objective function: Evidence lower bound (ELBO)

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \tag{32}$$

$$= \ln p_\theta(\mathbf{x}) - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{33}$$

Comments:

Objective function: Evidence lower bound (ELBO)

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \tag{32}$$

$$= \ln p_\theta(\mathbf{x}) - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{33}$$

Comments:

- Second form intractable

Objective function: Evidence lower bound (ELBO)

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \tag{32}$$

$$= \ln p_\theta(\mathbf{x}) - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{33}$$

Comments:

- Second form intractable
- Second form clearly a lower bound

Objective function: Evidence lower bound (ELBO)

$$F(\theta, \phi) = \mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\big) \tag{32}$$

$$= \ln p_\theta(\mathbf{x}) - D_{KL}\big(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\big) \tag{33}$$

Comments:

- Second form intractable
- Second form clearly a lower bound
- Second bound is tight if and only if $q_\phi(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{z}|\mathbf{x})$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)}_{\textbf{Regularization}} \tag{34}$$

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

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)}_{\textbf{Regularization}} \tag{34}$$

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

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right) = \frac{1}{2}\left[1 + \ln g_\phi^\sigma(\mathbf{x}) - g_\phi^\mu(\mathbf{x}) - g_\phi^\sigma(\mathbf{x})\right] \tag{35}$$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)}_{\textbf{Regularization}} \tag{34}$$

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

$$D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right) = \frac{1}{2}\left[1 + \ln g_\phi^\sigma(\mathbf{x}) - g_\phi^\mu(\mathbf{x}) - g_\phi^\sigma(\mathbf{x})\right] \tag{35}$$

- Differentiable function of inference net parameters

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\textbf{Reconstruction}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)}_{\text{Regularization}} \tag{36}$$

- **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

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\textbf{Reconstruction}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)}_{\text{Regularization}} \tag{36}$$

- **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 $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x})$

$$\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] \approx \frac{1}{S}\sum_{s=1}^{S} \ln p_\theta(\mathbf{x}|\mathbf{z_s}) \tag{37}$$

Computation ELBO for variational autoencoder

$$F(\theta, \phi) = \underbrace{\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})]}_{\textbf{Reconstruction}} - \underbrace{D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})\right)}_{\text{Regularization}} \tag{36}$$

- **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
  $\mathbf{z_s} \sim q_\phi(\mathbf{z}|\mathbf{x})$

$$\mathbb{E}_{q_\phi}[\ln p_\theta(\mathbf{x}|\mathbf{z})] \approx \frac{1}{S}\sum_{s=1}^{S} \ln p_\theta(\mathbf{x}|\mathbf{z_s}) \tag{37}$$

- Estimator is non-differentiable due to sampling operator

Re-parametrization trick

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

Re-parametrization trick

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

$$\mathbf{z}_s = g_\phi^\mu(\mathbf{x}) + g_\phi^\sigma(\mathbf{x}) \odot \epsilon_s, \qquad \epsilon_s \sim \mathcal{N}(\epsilon_s; 0, I) \tag{38}$$

Re-parametrization trick

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

$$\mathbf{z}_s = g_\phi^\mu(\mathbf{x}) + g_\phi^\sigma(\mathbf{x}) \odot \epsilon_s, \qquad \epsilon_s \sim \mathcal{N}(\epsilon_s; 0, I) \tag{38}$$

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

Re-parametrization trick

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

$$\mathbf{z}_s = g_\phi^\mu(\mathbf{x}) + g_\phi^\sigma(\mathbf{x}) \odot \epsilon_s, \qquad \epsilon_s \sim \mathcal{N}(\epsilon_s; 0, I) \tag{38}$$

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

$$F(\theta, \phi) \approx \frac{1}{S} \sum_{s=1}^{S} \ln p_\theta\left(\mathbf{x}|g_\phi^\mu(\mathbf{x}) + g_\phi^\sigma(\mathbf{x}) \odot \epsilon_s\right) \tag{39}$$

$$-\frac{1}{2}\left[1 + \ln g_\phi^\sigma(\mathbf{x}) - g_\phi^\mu(\mathbf{x}) - g_\phi^\sigma(\mathbf{x})\right] \tag{40}$$

Re-parametrization trick in a cartoon

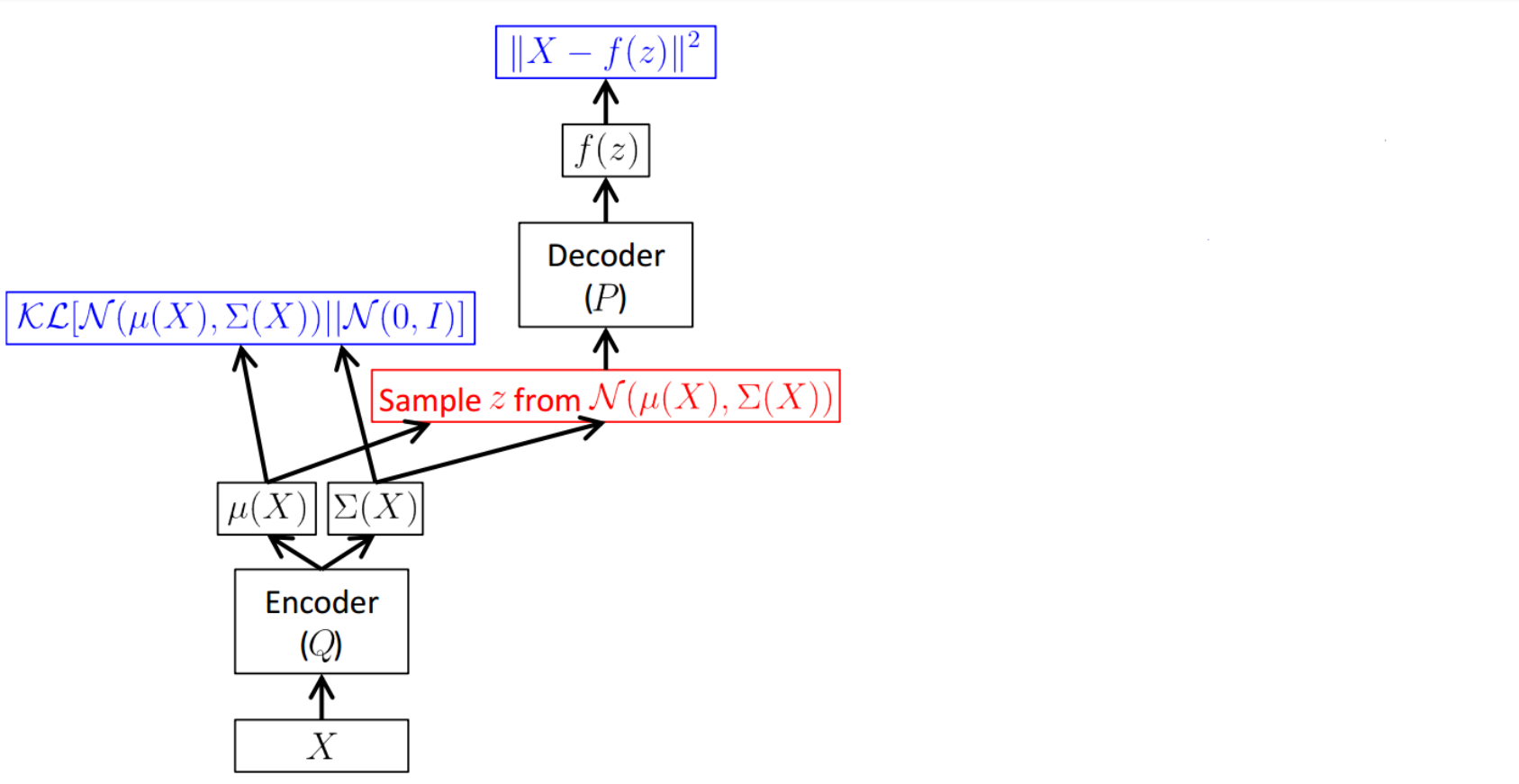


Figure from [Doersch, 2016]

Re-parametrization trick in a cartoon

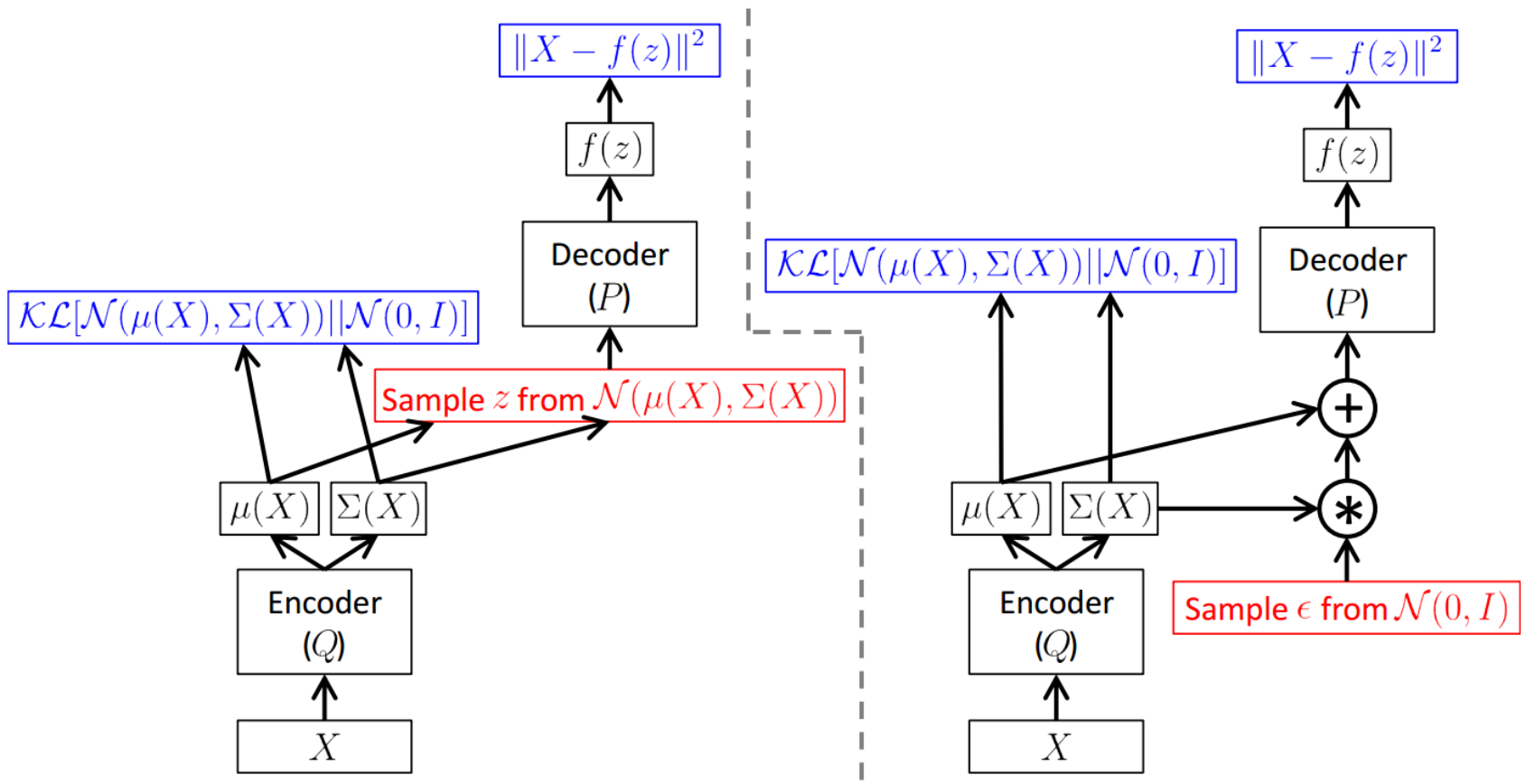


Figure from [Doersch, 2016]

Autoencoding variational Bayes training algorithm

- For each data point **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

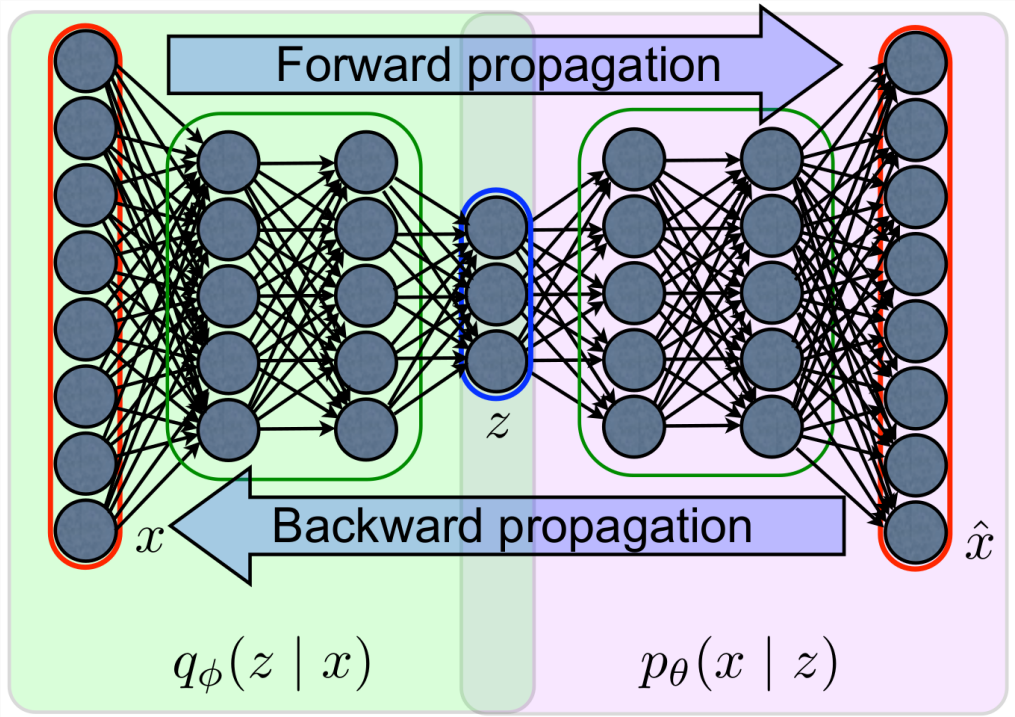


Figure from Aaron Courville

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]

Part VI

Deep invertible transformations

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

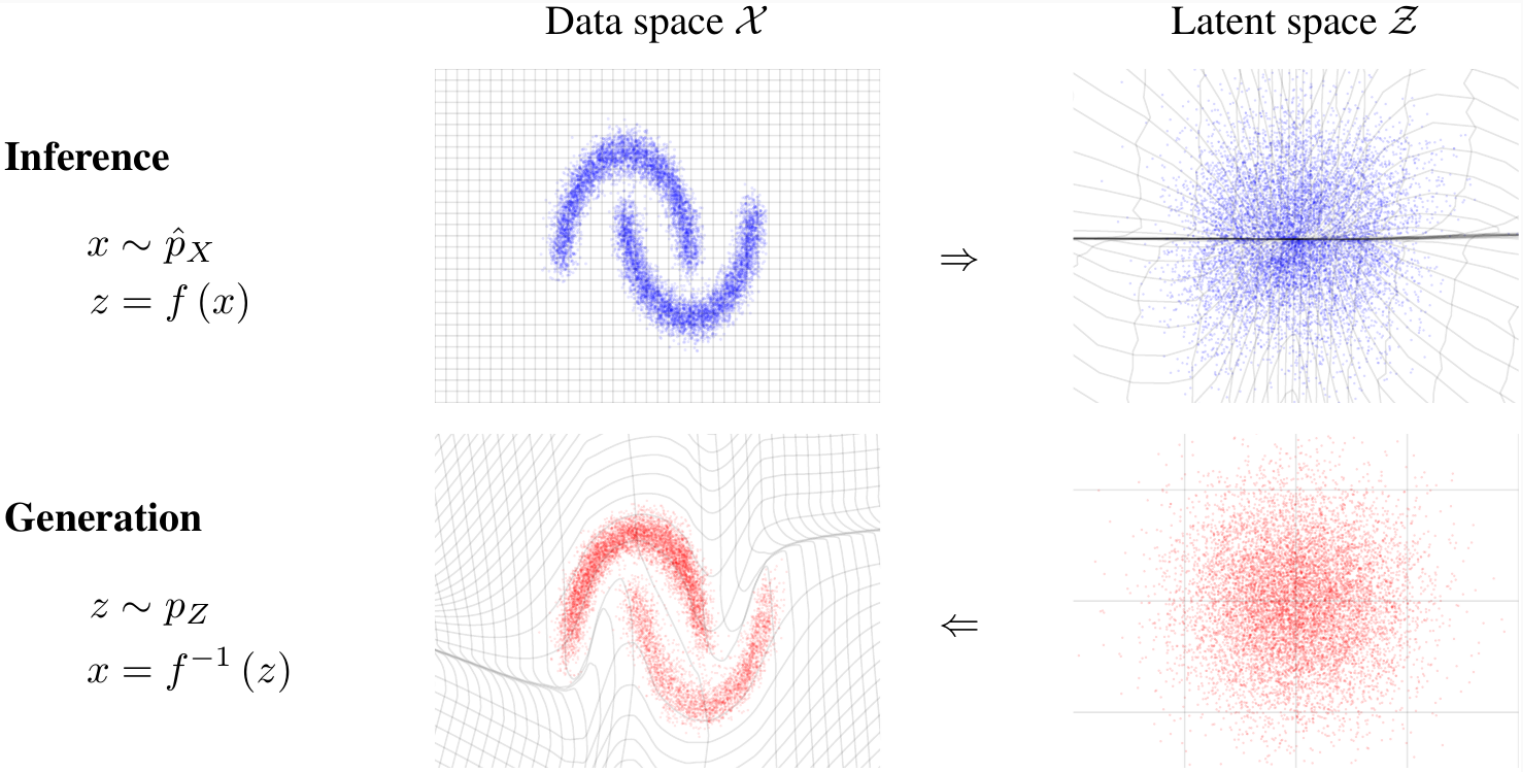
- Learn invertible function from latent to data space

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

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

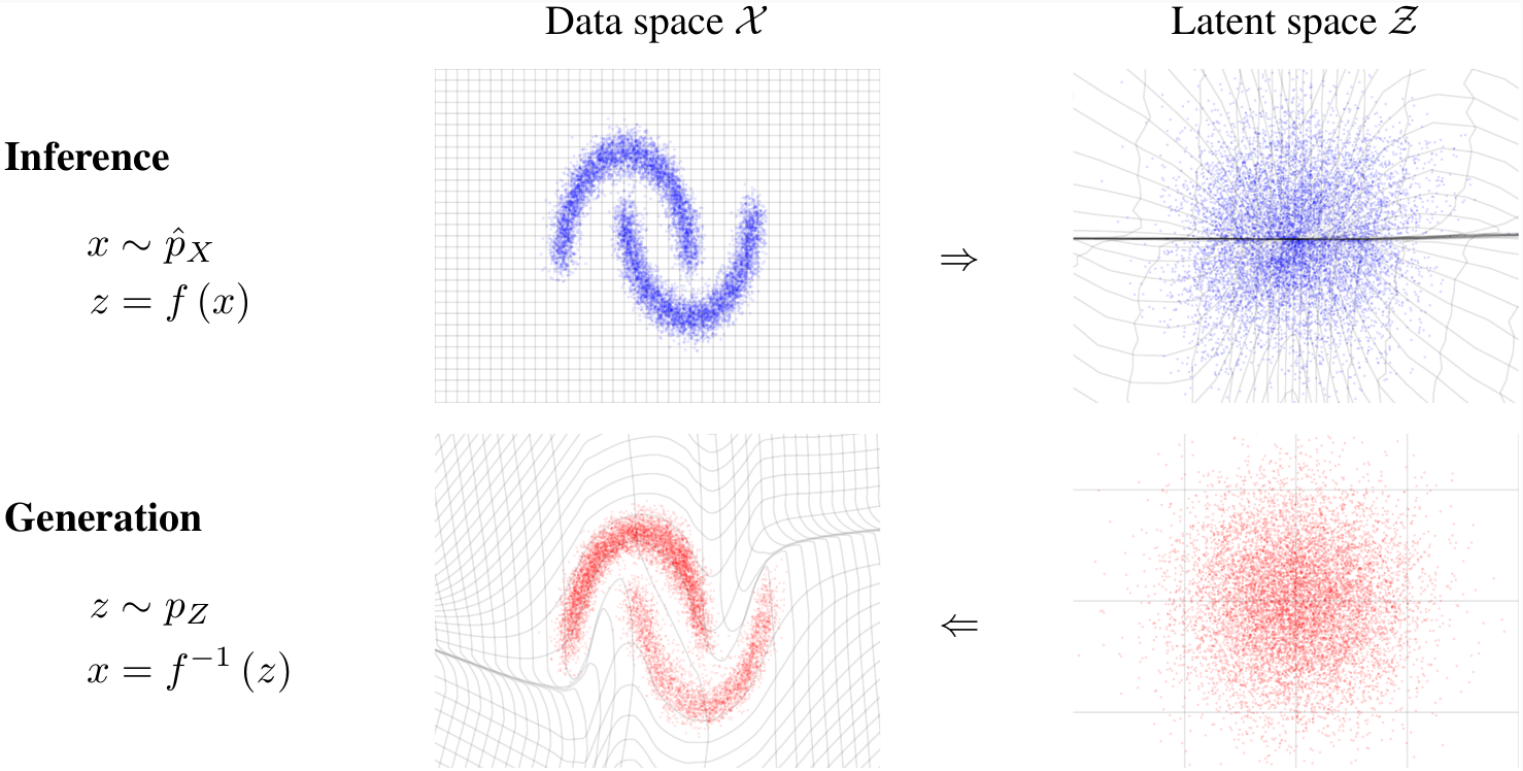
Non-volume preserving (NVP) transformation [Dinh et al., 2017]

- 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

- Using the change of variable formula:

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

- Need to ensure efficient computation of (i) $\mathbf{y} = f(\mathbf{x})$ and (ii) determinant

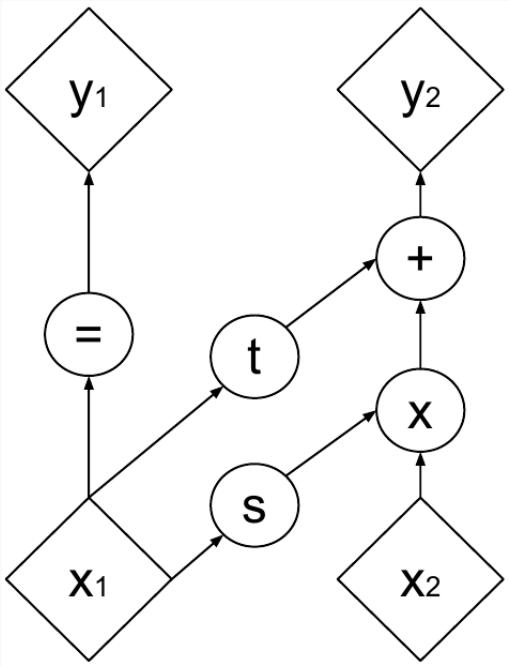
Change of variable formula for invertible function

- Using the change of variable formula:

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

- Need to ensure efficient computation of (i) $\mathbf{y} = f(\mathbf{x})$ and (ii) determinant

1. Partition variables in two groups



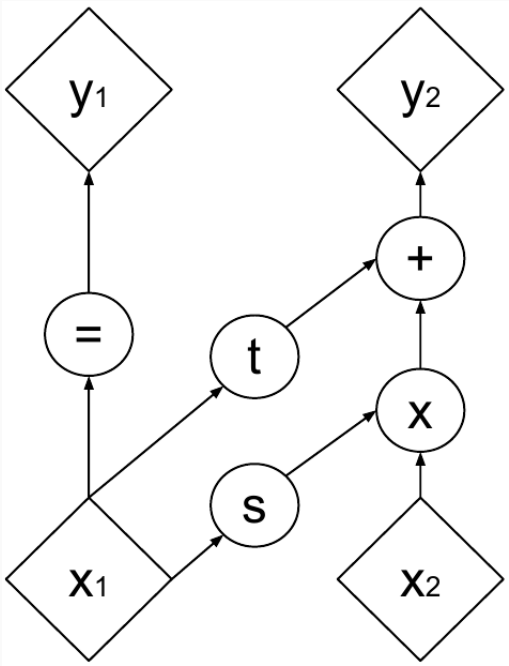
Change of variable formula for invertible function

- Using the change of variable formula:

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

- Need to ensure efficient computation of (i) $\mathbf{y} = f(\mathbf{x})$ and (ii) determinant

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



Change of variable formula for invertible function

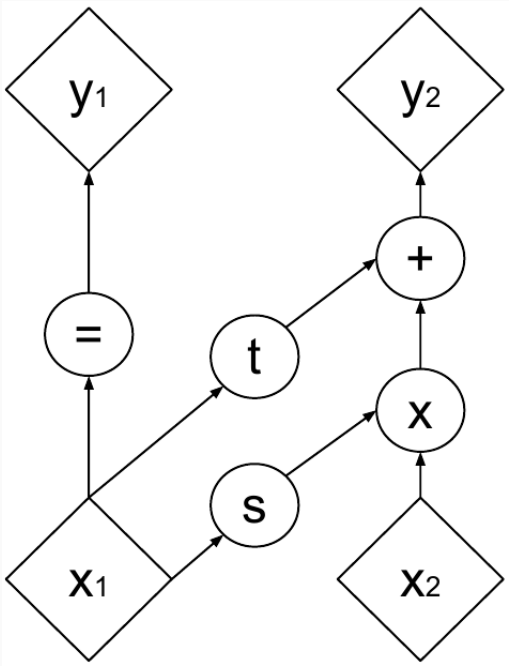
- Using the change of variable formula:

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

- Need to ensure efficient computation of (i) $\mathbf{y} = f(\mathbf{x})$ and (ii) determinant

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

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

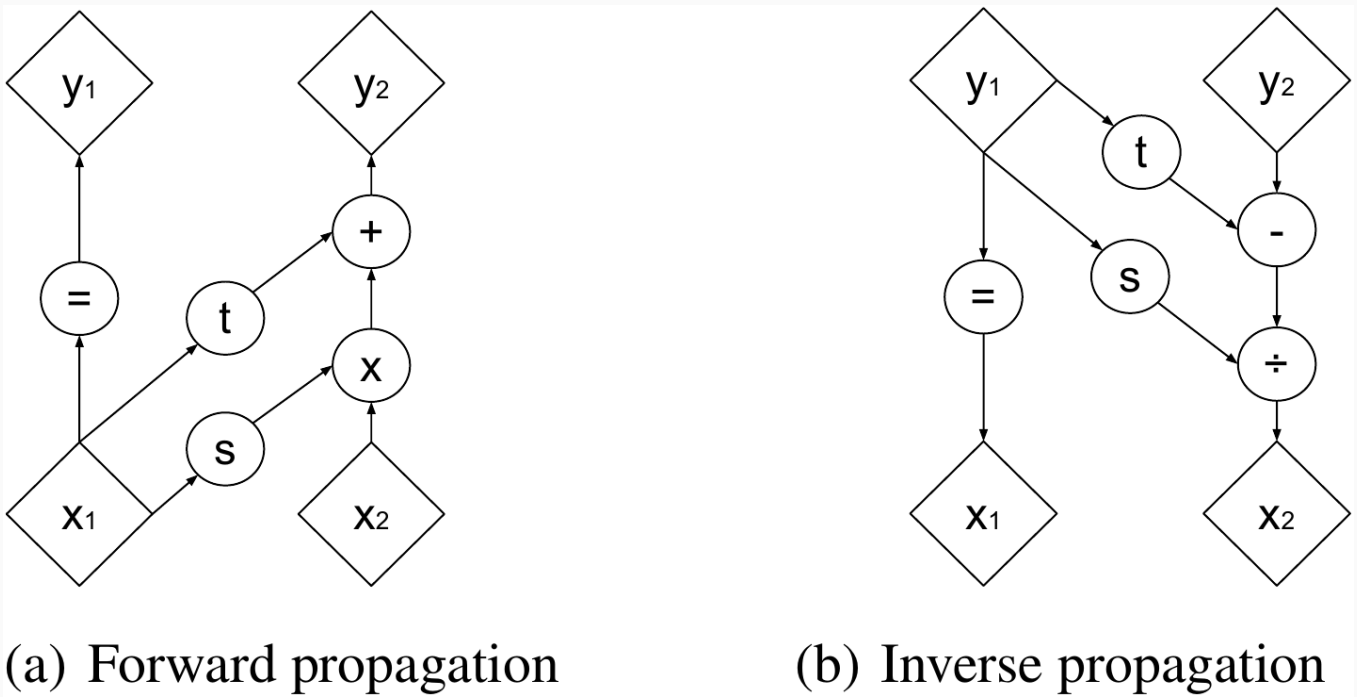


Properties: Efficient inversion

- Inverse transformation

$$\mathbf{x}_1 = \mathbf{y}_1 \tag{41}$$

$$\mathbf{x}_2 = (\mathbf{y}_2 - t(\mathbf{x}_1)) \odot \exp(-s(\mathbf{x}_1)) \tag{42}$$



(a) Forward propagation (b) Inverse propagation

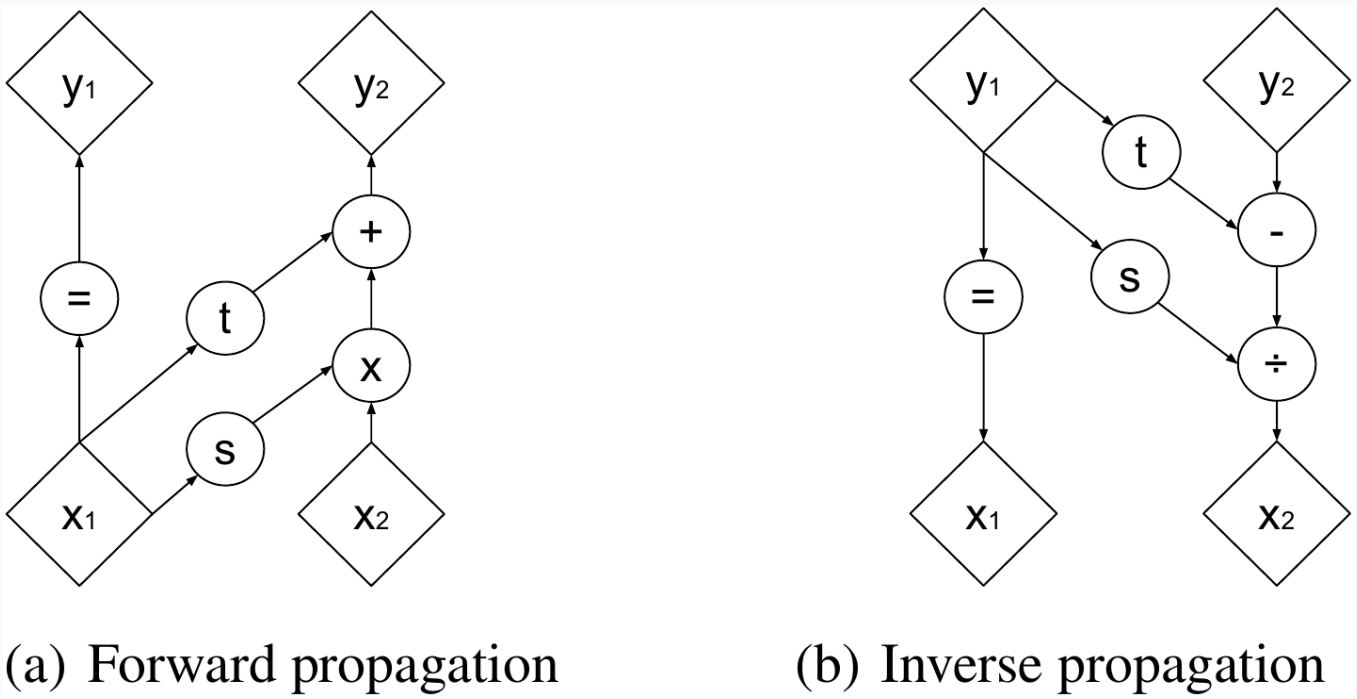
Properties: Efficient inversion

- Inverse transformation

$$\mathbf{x}_1 = \mathbf{y}_1 \tag{41}$$

$$\mathbf{x}_2 = (\mathbf{y}_2 - t(\mathbf{x}_1)) \odot \exp(-s(\mathbf{x}_1)) \tag{42}$$

- 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

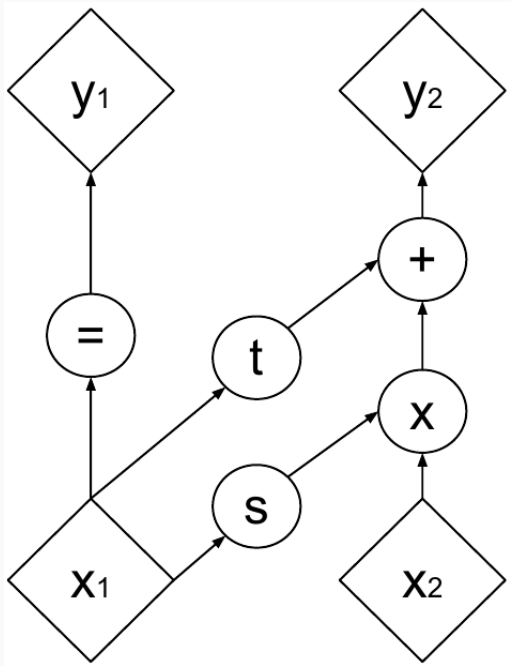
Properties: Efficient determinant computation

- 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

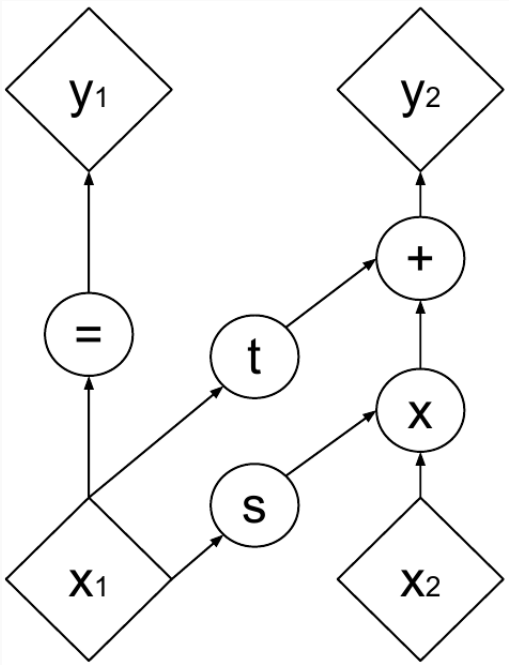
$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}^\top} = \begin{bmatrix} I_d & 0 \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1^\top} & \text{diag}(\exp(s(\mathbf{x}_1))) \end{bmatrix}$$

- Determinant given by product of Jacobian's diagonal terms

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

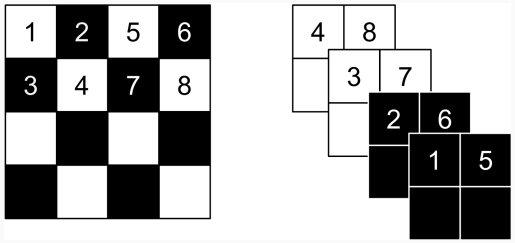
- Log-likelihood easily computed, optimize using stochastic gradient decent

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



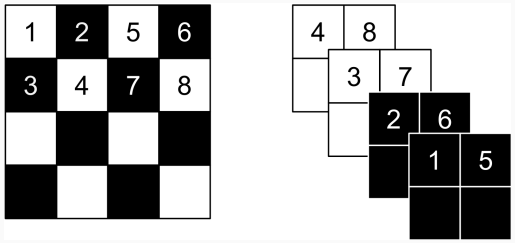
Implementation

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



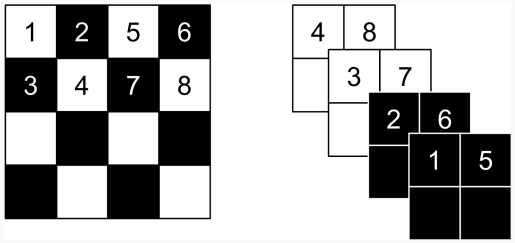
Implementation

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

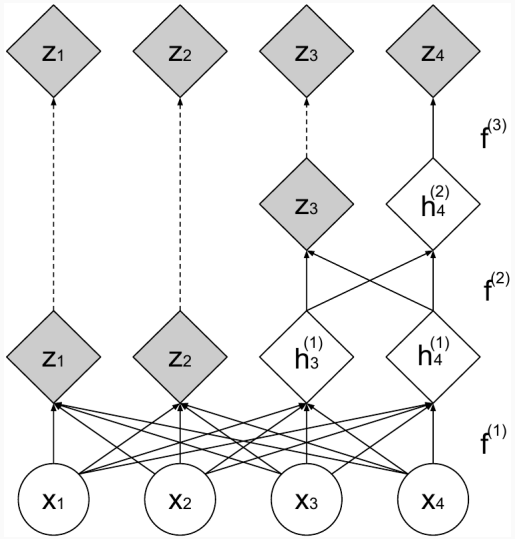


Implementation

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

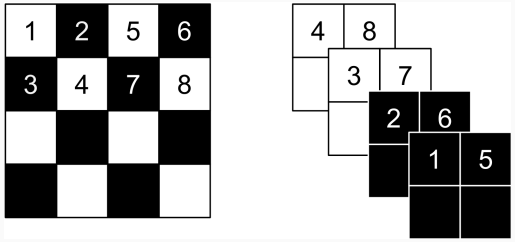


- Stack multiple layers of transformations

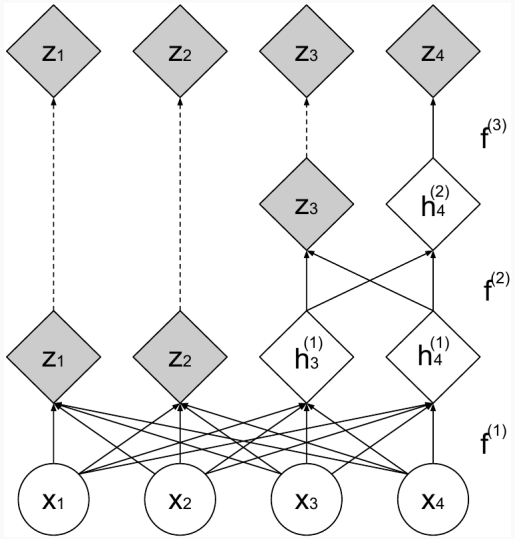


Implementation

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

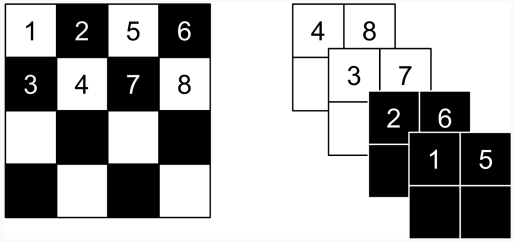


- Stack multiple layers of transformations
  - Alternating the masking patterns

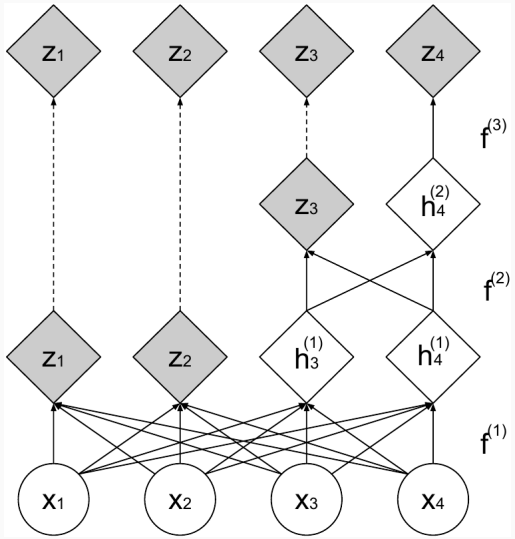


Implementation

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

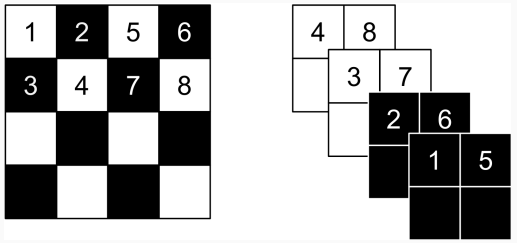


- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants



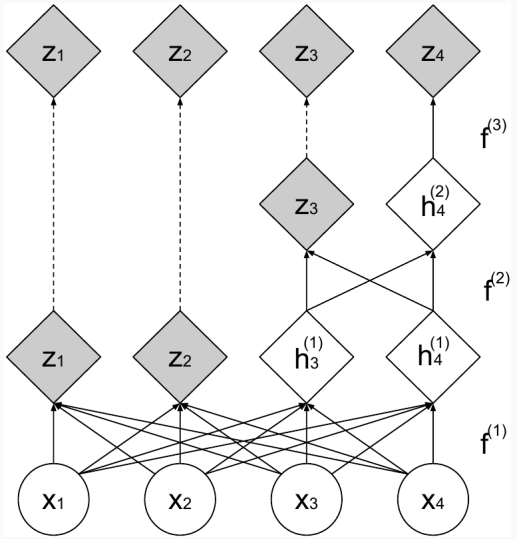
Implementation

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



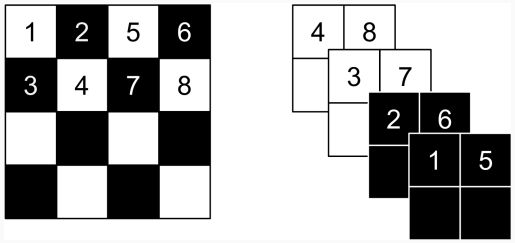
- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants

- Feature abstraction hierarchy



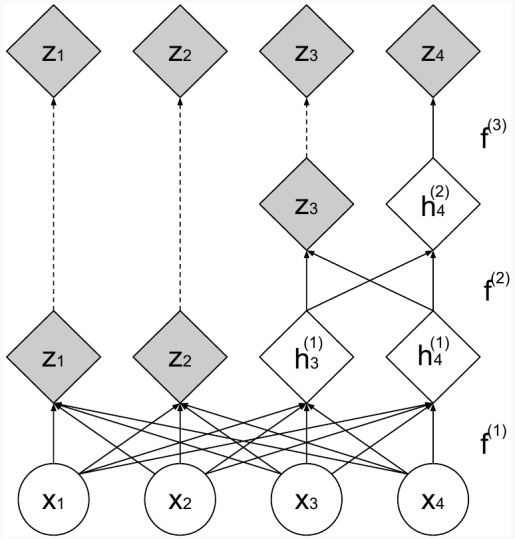
Implementation

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



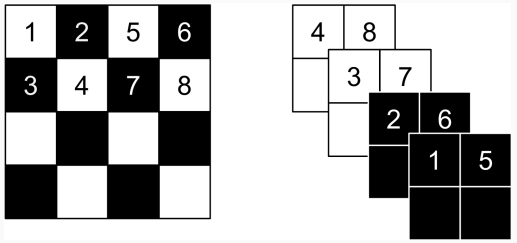
- Stack multiple layers of transformations
  - Alternating the masking patterns
  - Composing data transformations, summing log determinants

- 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

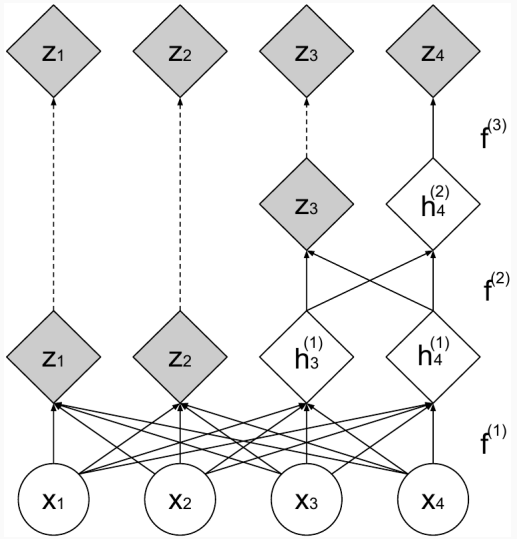


Illustration multi-scale feature hierarchy

- Images obtained after re-sampling part of latent variables

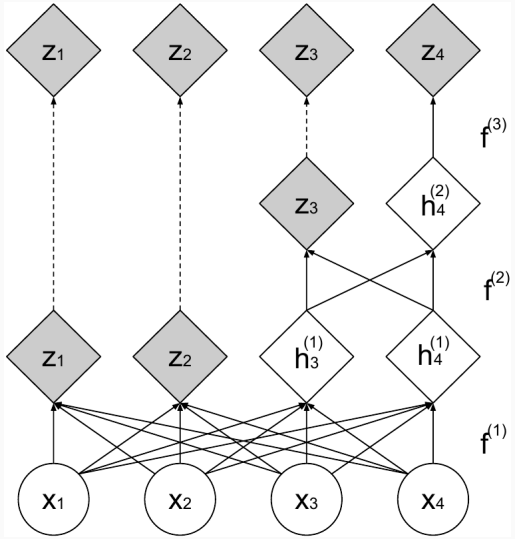


Illustration multi-scale feature hierarchy

- 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

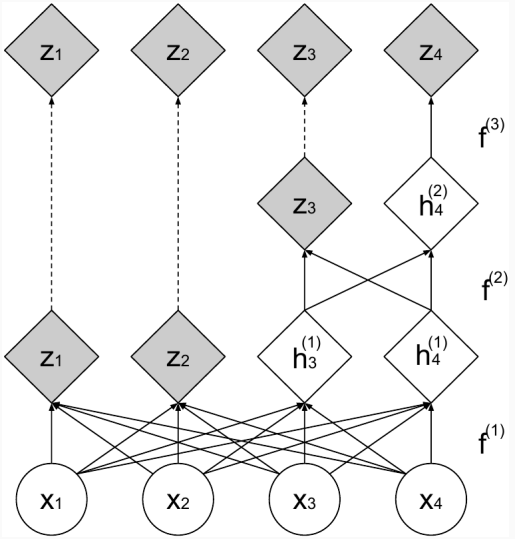
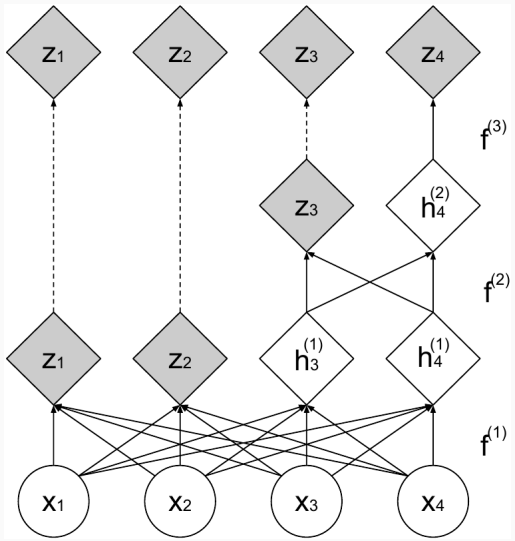


Illustration multi-scale feature hierarchy

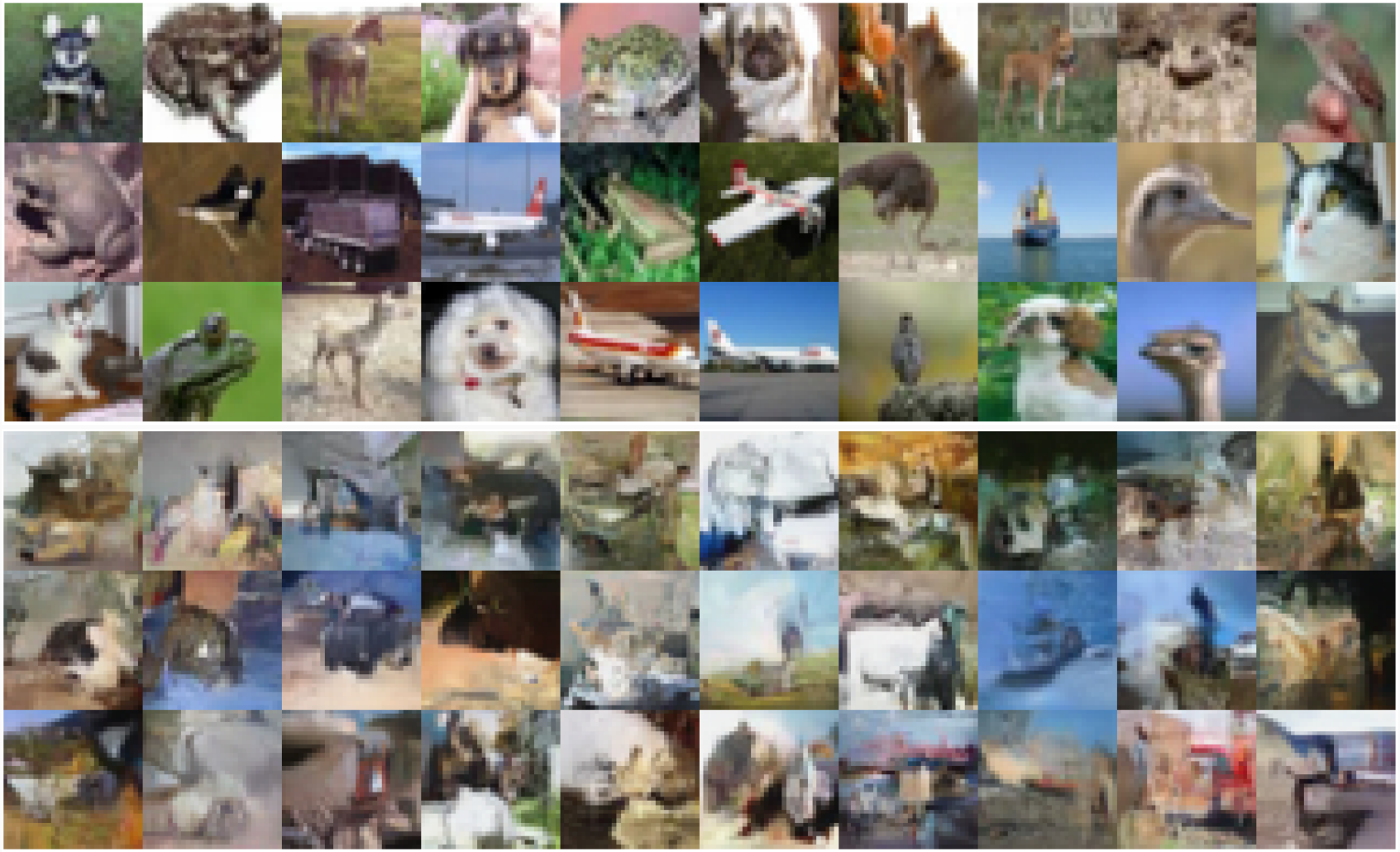
- Images obtained after re-sampling part of latent variables
- From left to right: original, keeping $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$

ImageNet 64 × 64

CelebA 64 × 64



Images & Samples NVP: CIFAR10 Dataset 32×32



Part VII

Autoregressive density estimation

Autoregressive modeling

- Consider generic factorization of joint probability

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{43}$$

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

Autoregressive modeling

- Consider generic factorization of joint probability

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{43}$$

  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

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{43}$$

  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

$$p(\mathbf{x}_{1:D}) = p(x_1) \prod_{i=2}^{D} p(x_i|\mathbf{x}_{<i}) \tag{43}$$

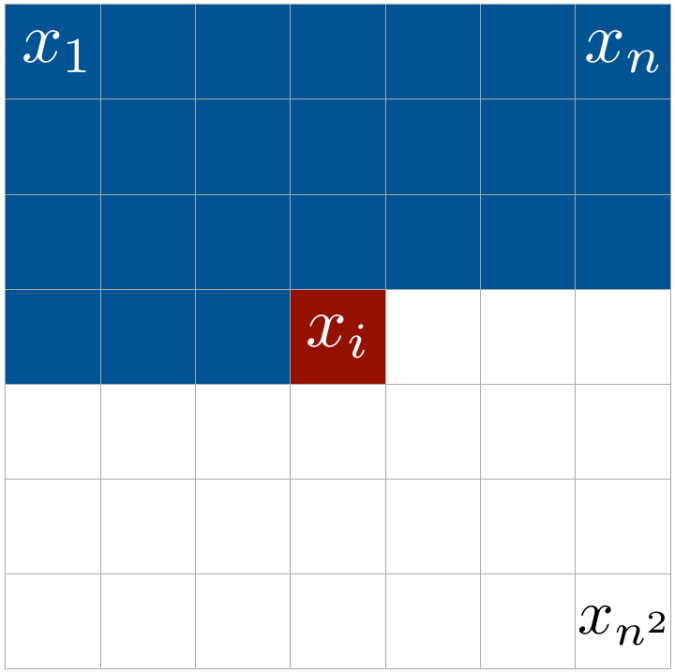
  with $\mathbf{x}_{<i} = \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$

- 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

Pixel Recurrent/Convolutional Neural Networks

[Oord et al., 2016b]

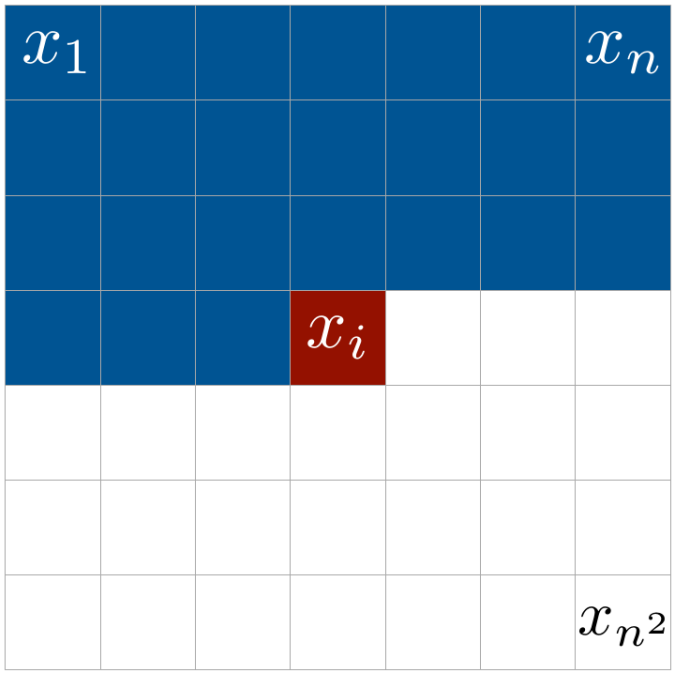
- Predict pixels one-by-one in row-major ordering



Pixel Recurrent/Convolutional Neural Networks

[Oord et al., 2016b]

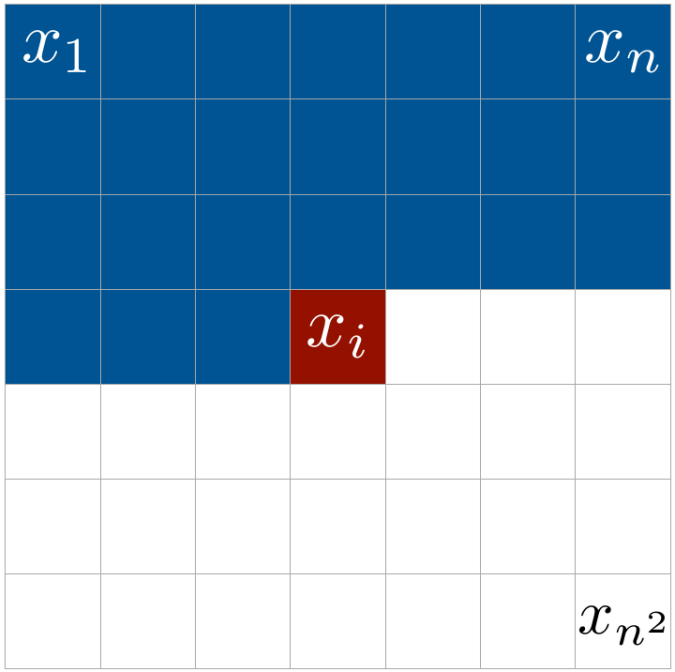
- Predict pixels one-by-one in row-major ordering
- Translation invariant definition of conditionals $p(x_i|\mathbf{x}_{<i})$



Pixel Recurrent/Convolutional Neural Networks

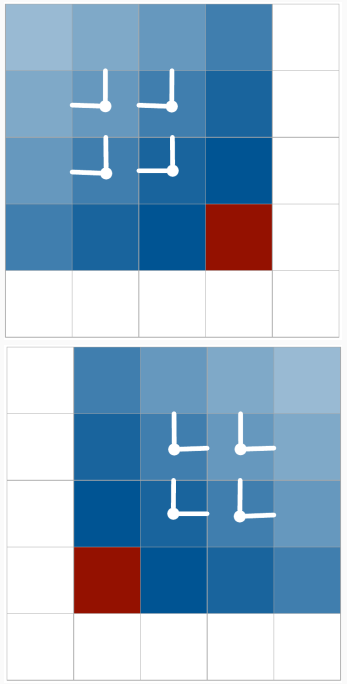
[Oord et al., 2016b]

- 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



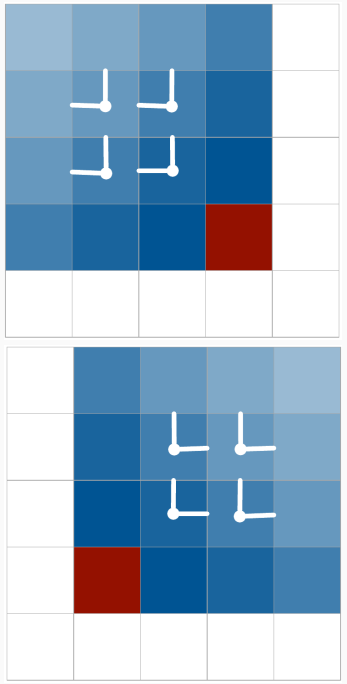
Pixel RNN: Bi-directional LSTM

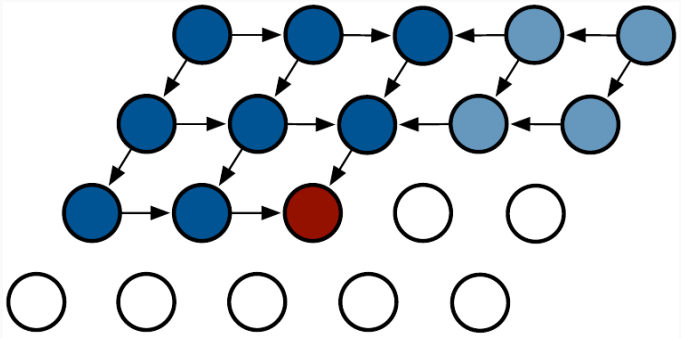
- Two sets of LSTM units, working down-right and down-left
  - Input up and left/right state
  - Input up and left/right pixels



Pixel RNN: Bi-directional LSTM

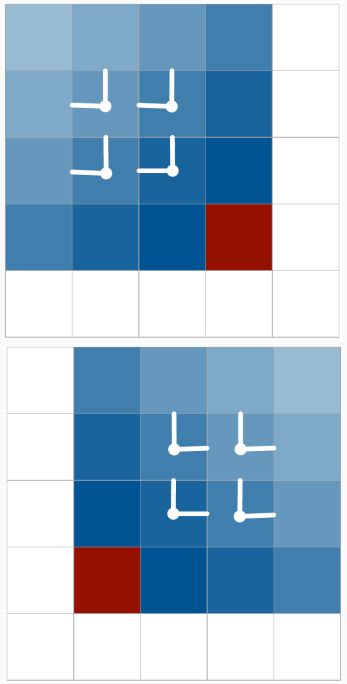
- Two sets of LSTM units, working down-right and down-left
  - Input up and left/right state
  - Input up and left/right pixels
- Receptive field
  - In each stream: all pixels above and to the right/left
  - Combined: all previous pixels

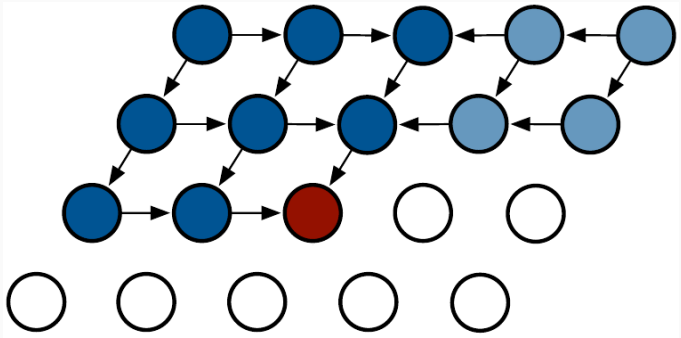




Pixel RNN: Bi-directional LSTM

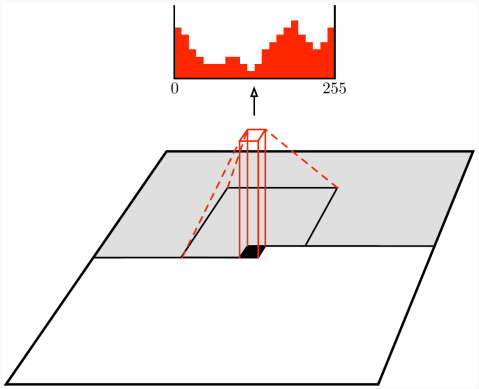
- Two sets of LSTM units, working down-right and down-left
  - Input up and left/right state
  - Input up and left/right pixels
- Receptive field
  - In each stream: all pixels above and to the right/left
  - Combined: all previous pixels
- Slow sequential training process
  - Due to sequential state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





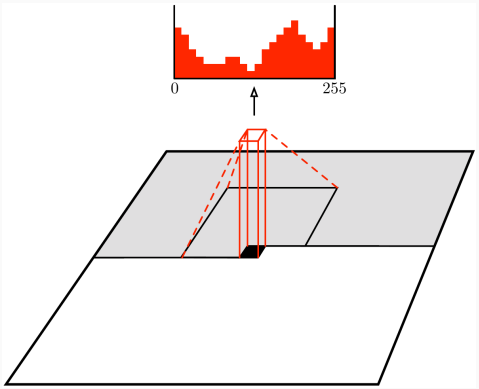
Pixel Convolutional Neural Networks

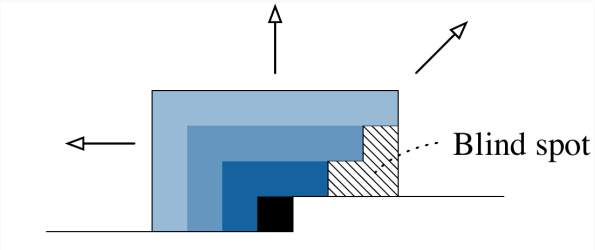
- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property



Pixel Convolutional Neural Networks

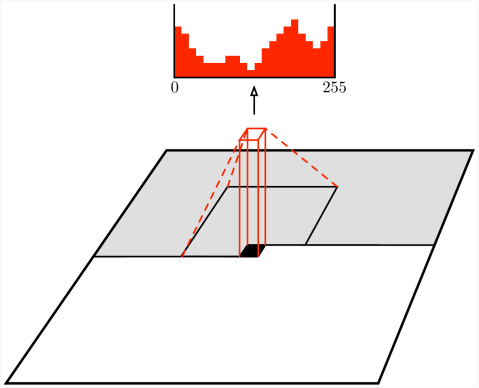
- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property
  - Layers increase receptive field

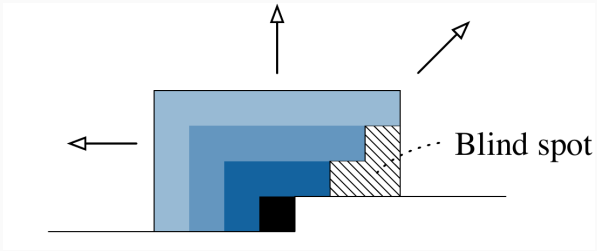


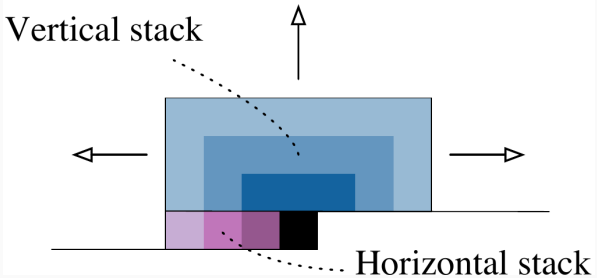


Pixel Convolutional Neural Networks

- 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

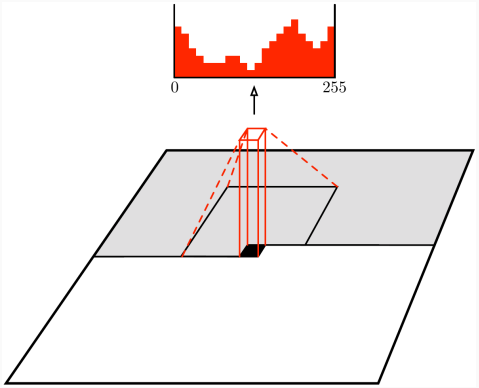


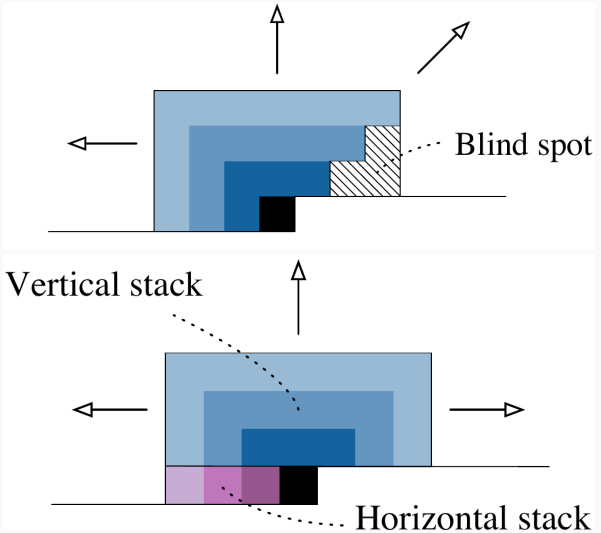




Pixel Convolutional Neural Networks

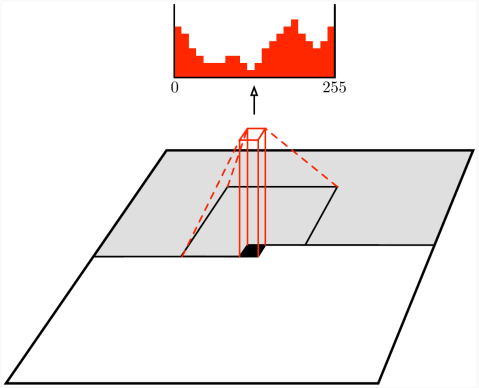
- Use limited context via CNN layers
  - Only local dependencies per layer
- Masked convolutions to ensure autoregressive property
  - Layers increase receptive field
  - Two stacks to fill blind spot: horizontal stack reads from vertical stack, not vice-versa
- Efficient parallel training, but sampling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

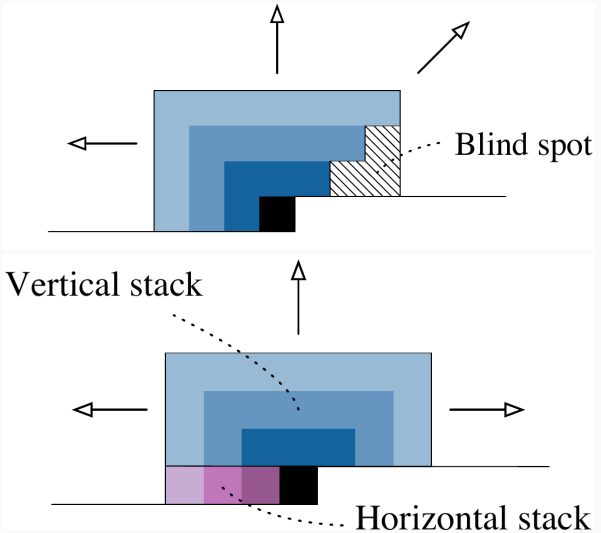




Pixel Convolutional Neural Networks

- 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]

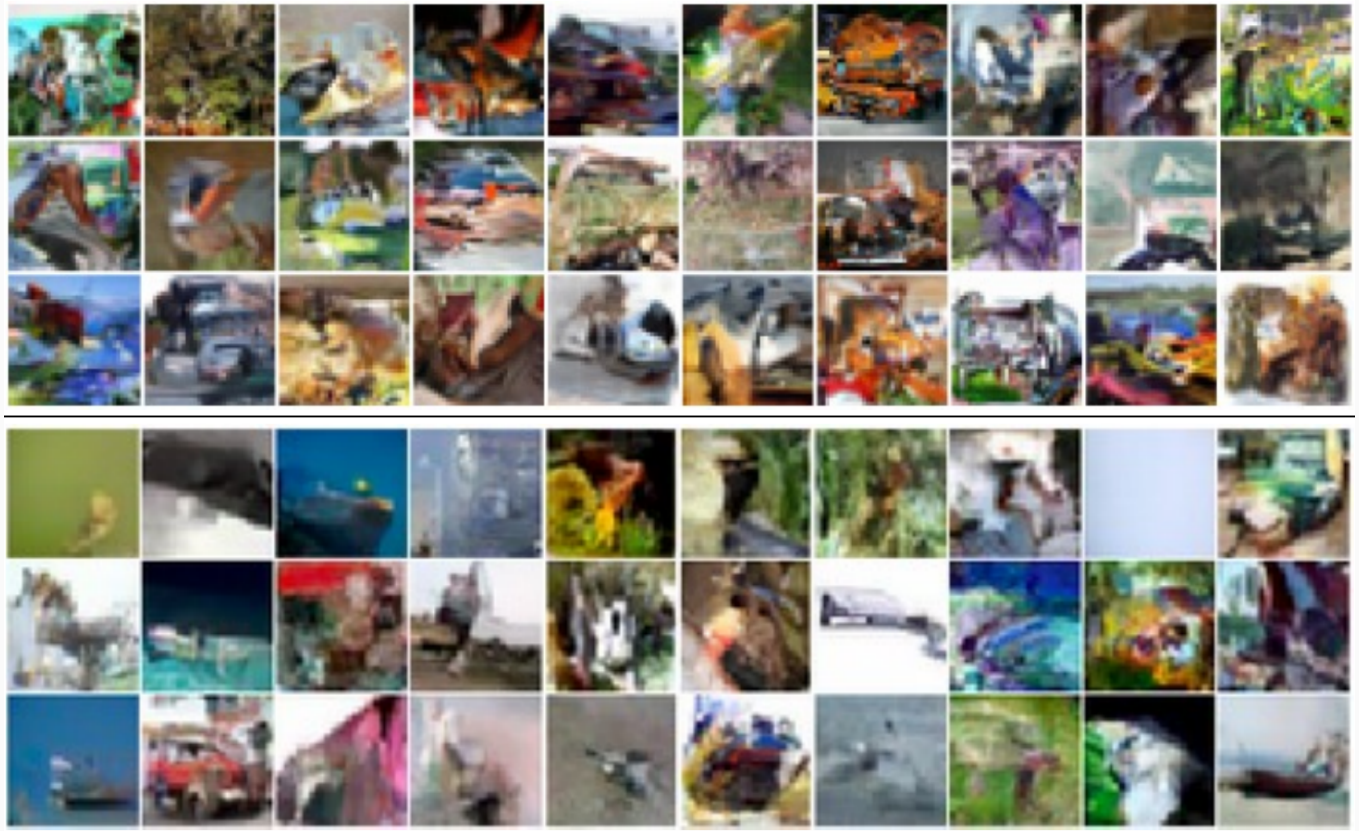




Class-conditional pixelCNN [Oord et al., 2016c]

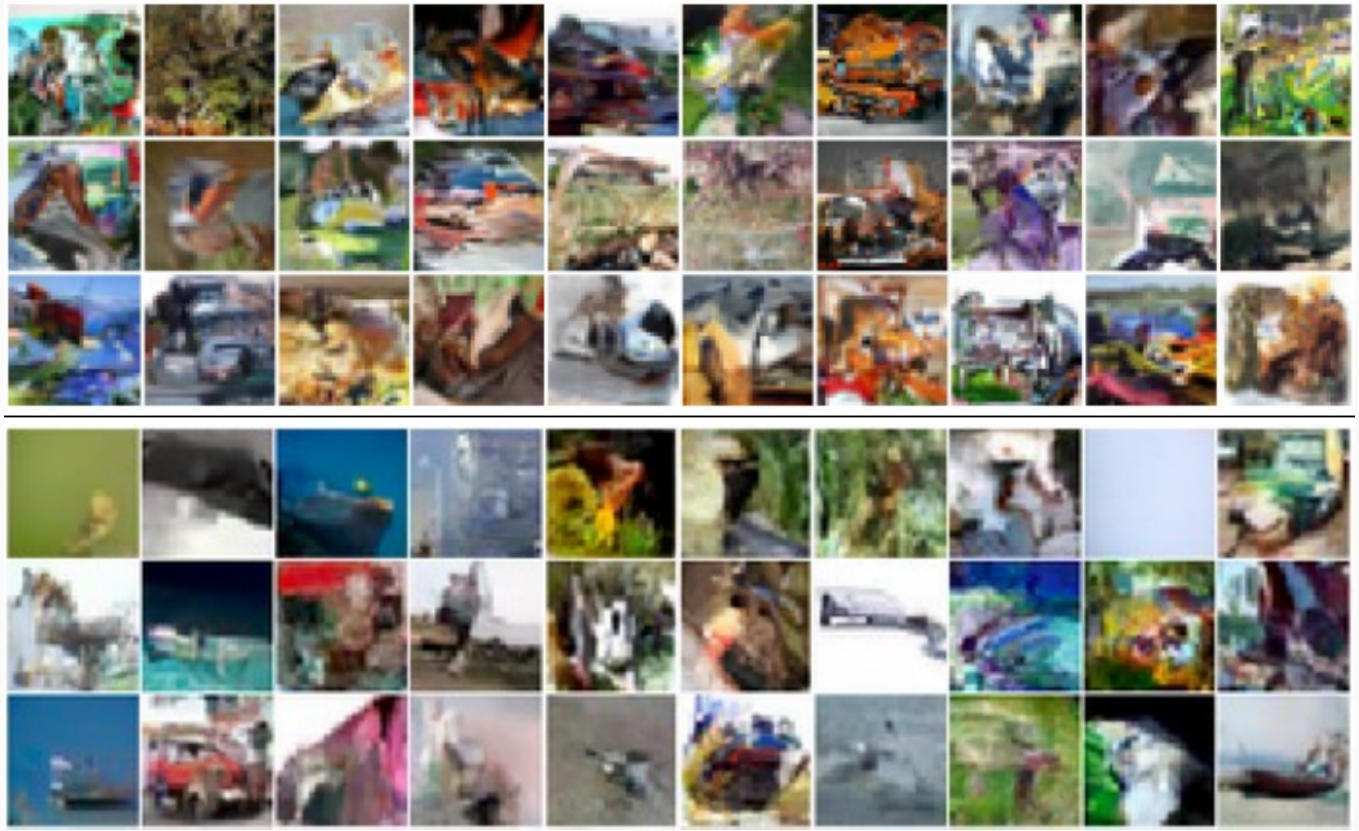
- Samples single model trained across 1,000 ImageNet classes

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

Parallel multiscale autoregressive density estimation

[Reed et al., 2017]

- Address the inherently limited sampling efficiency of autoregressive models

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

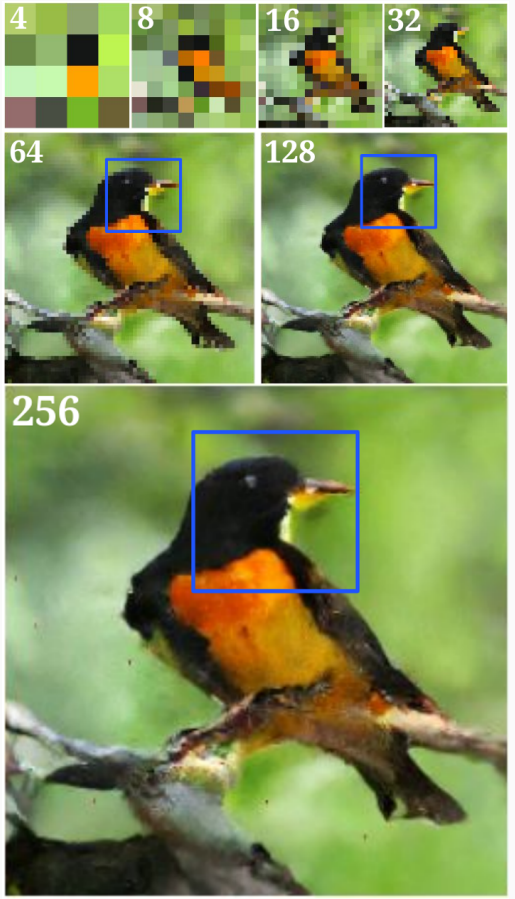
Parallel multiscale autoregressive density estimation

[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

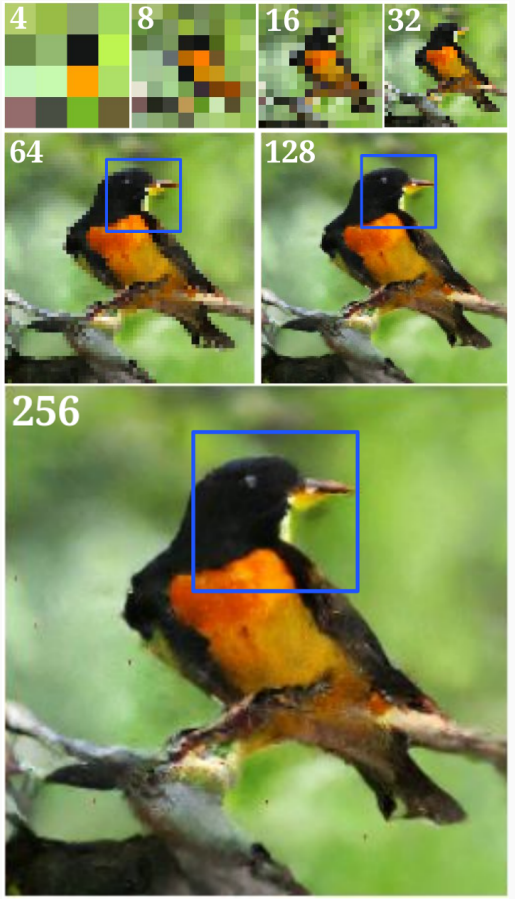


Parallel multiscale autoregressive density estimation

[Reed et al., 2017]

- Address the inherently limited sampling efficiency of autoregressive models

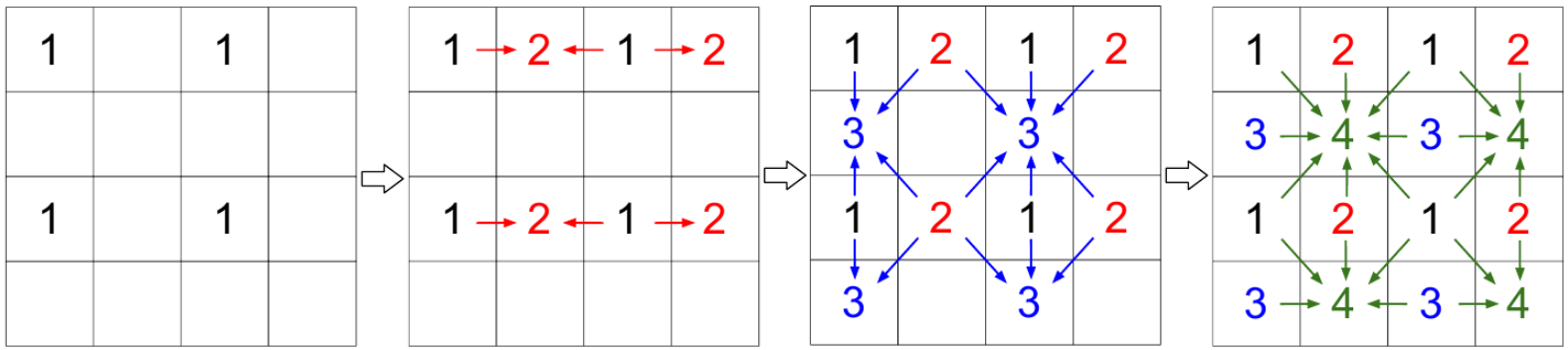
$$p(\mathbf{x}_{1:N}) = \prod_{i=1}^{N} p(x_i|\mathbf{x}_{<i})$$



- 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

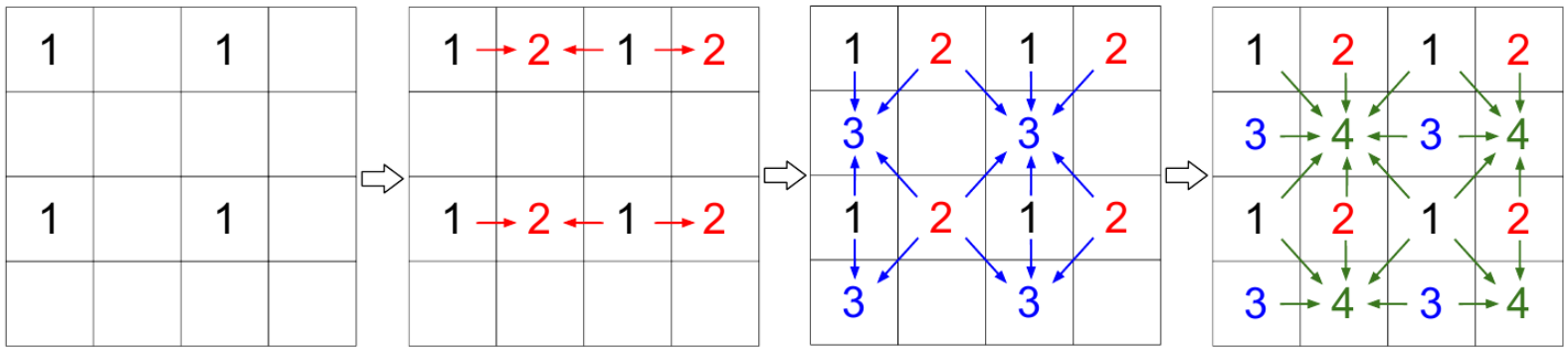
Sampling pixels in groups

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

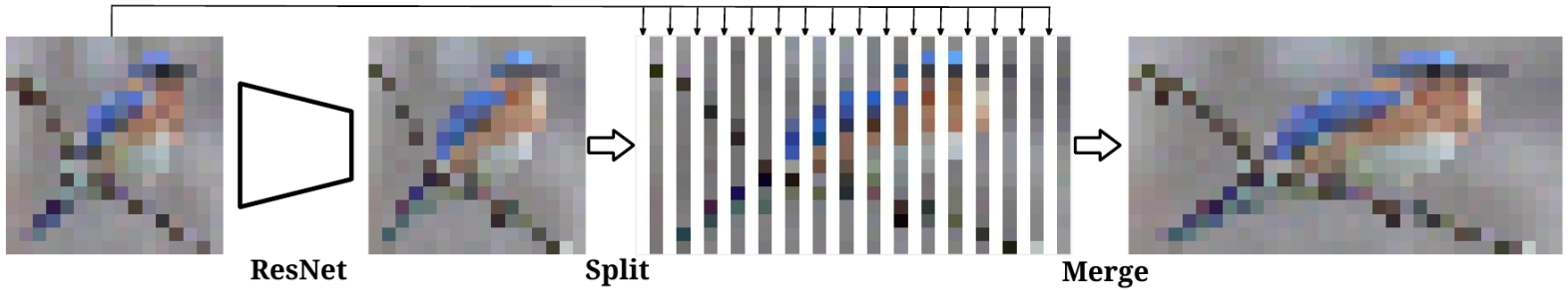


Sampling pixels in groups

- 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

References i


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


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


Breuleux, O., Bengio, Y., and Vincent, P. (2011).
Quickly generating representative samples from an RBM-derived process.
Neural Computation, 23(8):2053–2073.

References ii


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*.


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


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


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

References iii


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


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


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

References iv


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


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


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


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

References v


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


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

References vi


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


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


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*.


Roweis, S. (1997).
EM Algorithms for PCA and SPCA.
In *NeurIPS*.


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

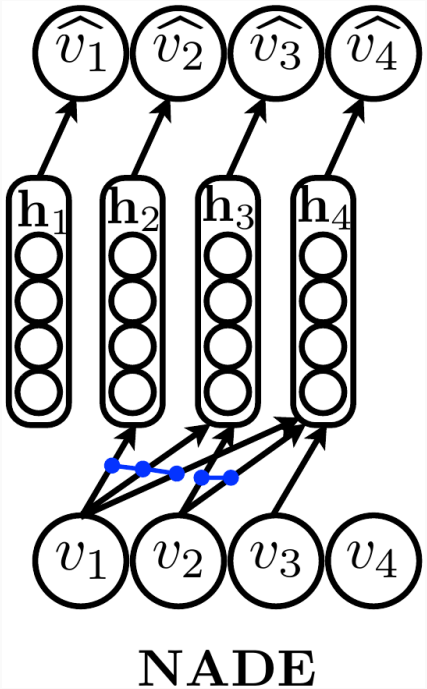

Salimans, T., Karpathy, A., Chen, X., and Kingma, D. (2017).
PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications.
In *ICLR*.


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

Neural Autoregressive Distribution Estimator (NADE) [?]

Neural Autoregressive Distribution Estimator (NADE) [?]

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

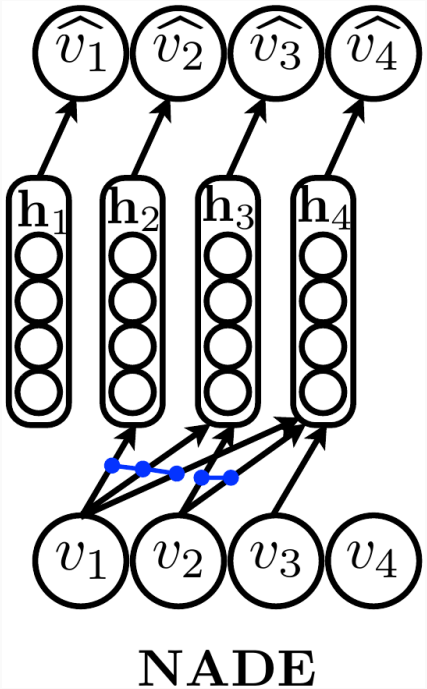


Neural Autoregressive Distribution Estimator (NADE) [?]

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

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

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



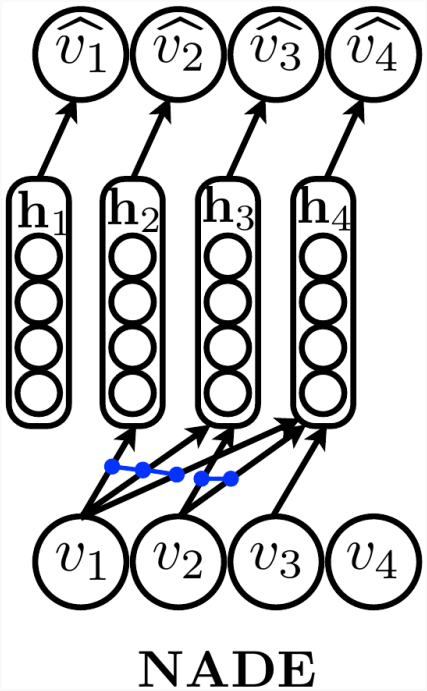
Neural Autoregressive Distribution Estimator (NADE) [?]

- 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 $\mathbf{h}_i$



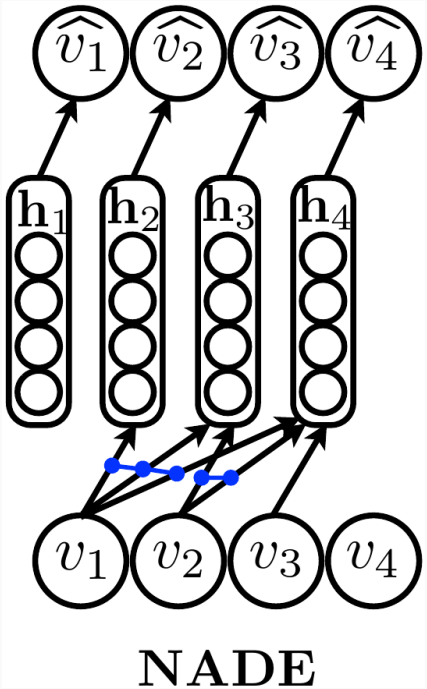
Neural Autoregressive Distribution Estimator (NADE) [?]

- 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 $\mathbf{h}_i$
- Input-to-hidden parameters $\mathbf{w}_j$ shared



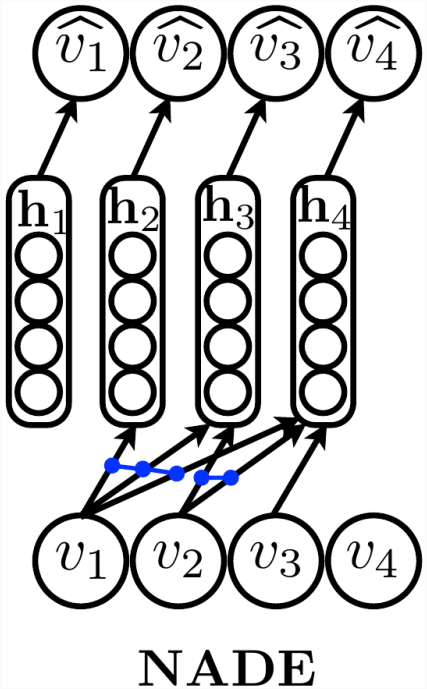
Neural Autoregressive Distribution Estimator (NADE) [?]

- 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 $\mathbf{h}_i$
- 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 $O(DH)$

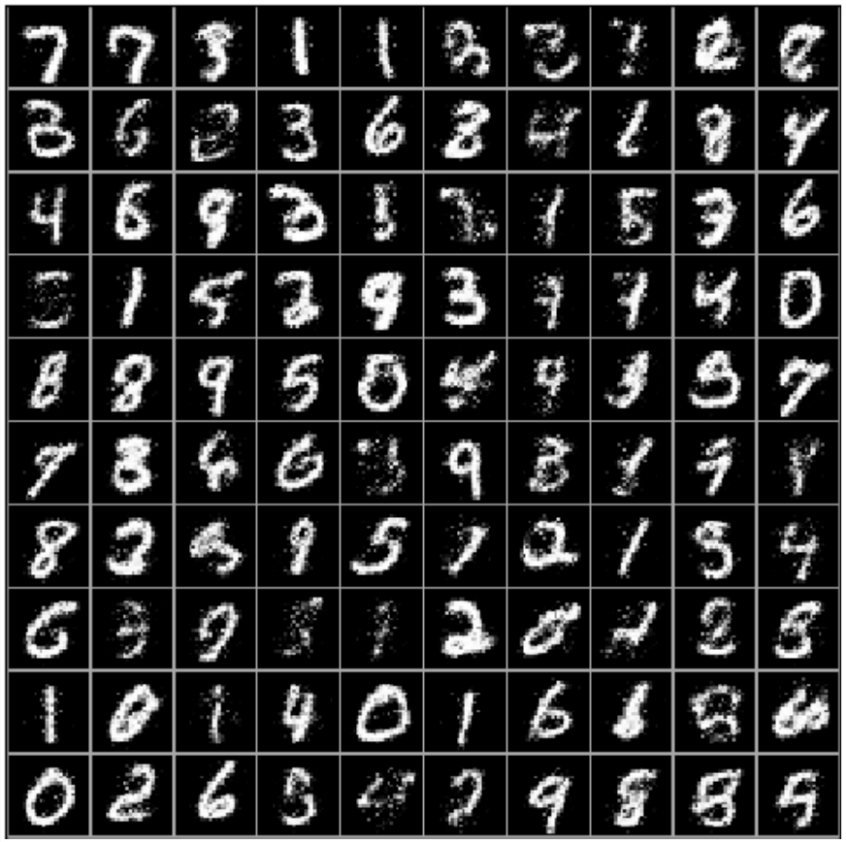
- Nr. of parameters linear in data and hidden dimension
- Computation of $\ln p(\mathbf{v})$ and gradient also linear

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

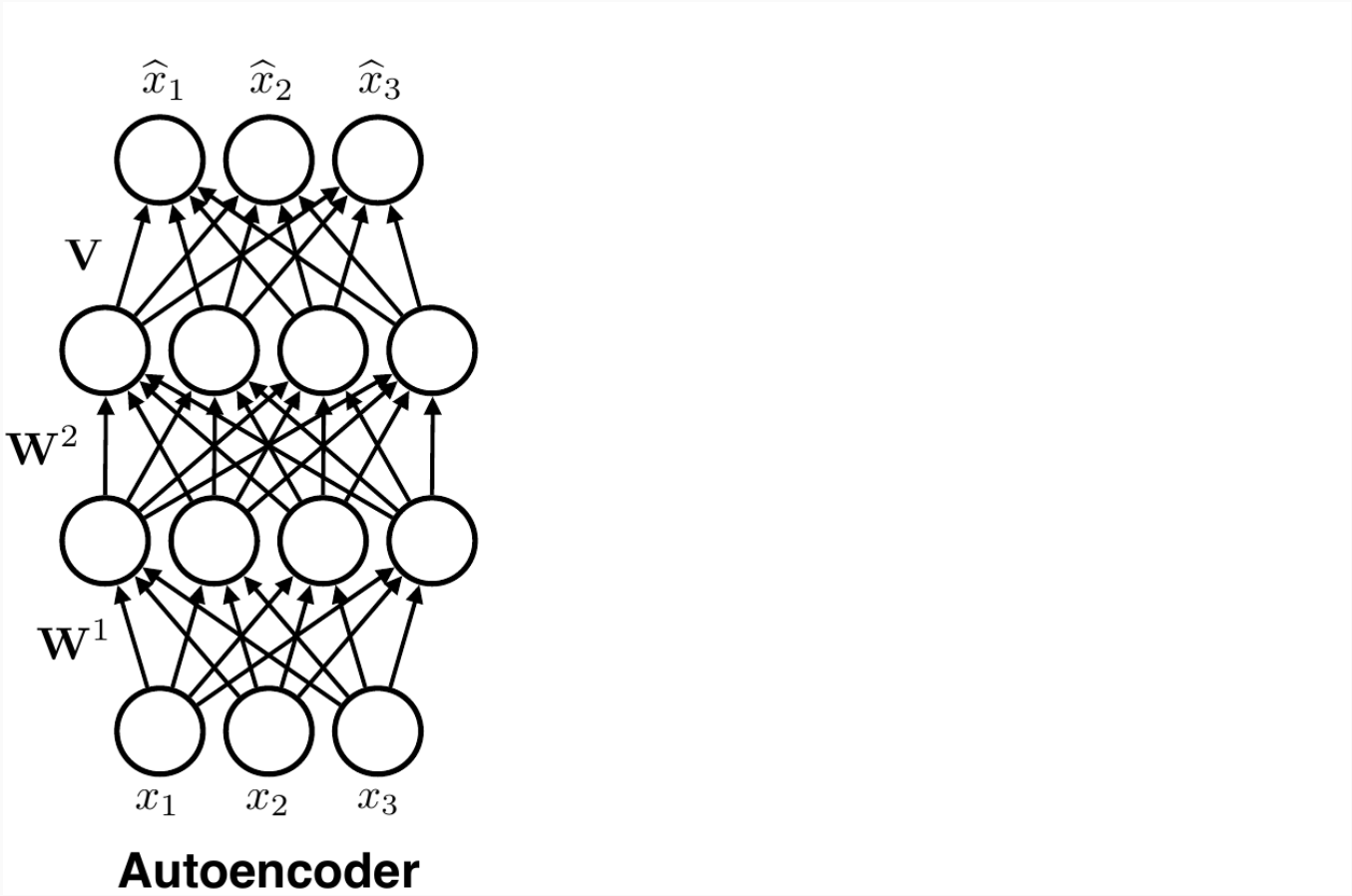


Samples from NADE trained on MNIST digits

Masked Autoencoder for Distribution Estimation (MADE) [?]

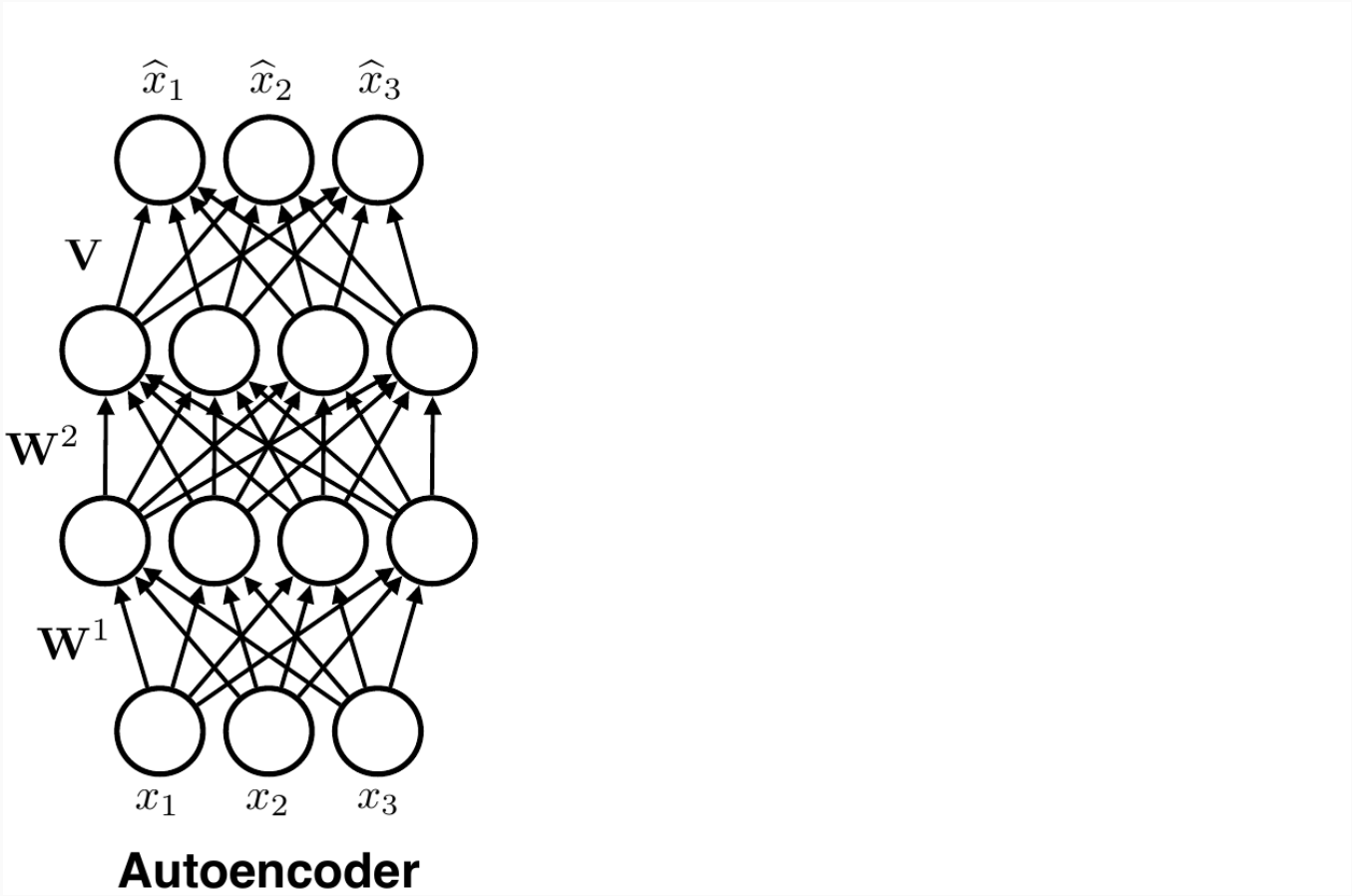
Masked Autoencoder for Distribution Estimation [?]

- Adapt autoencoders for distribution estimation



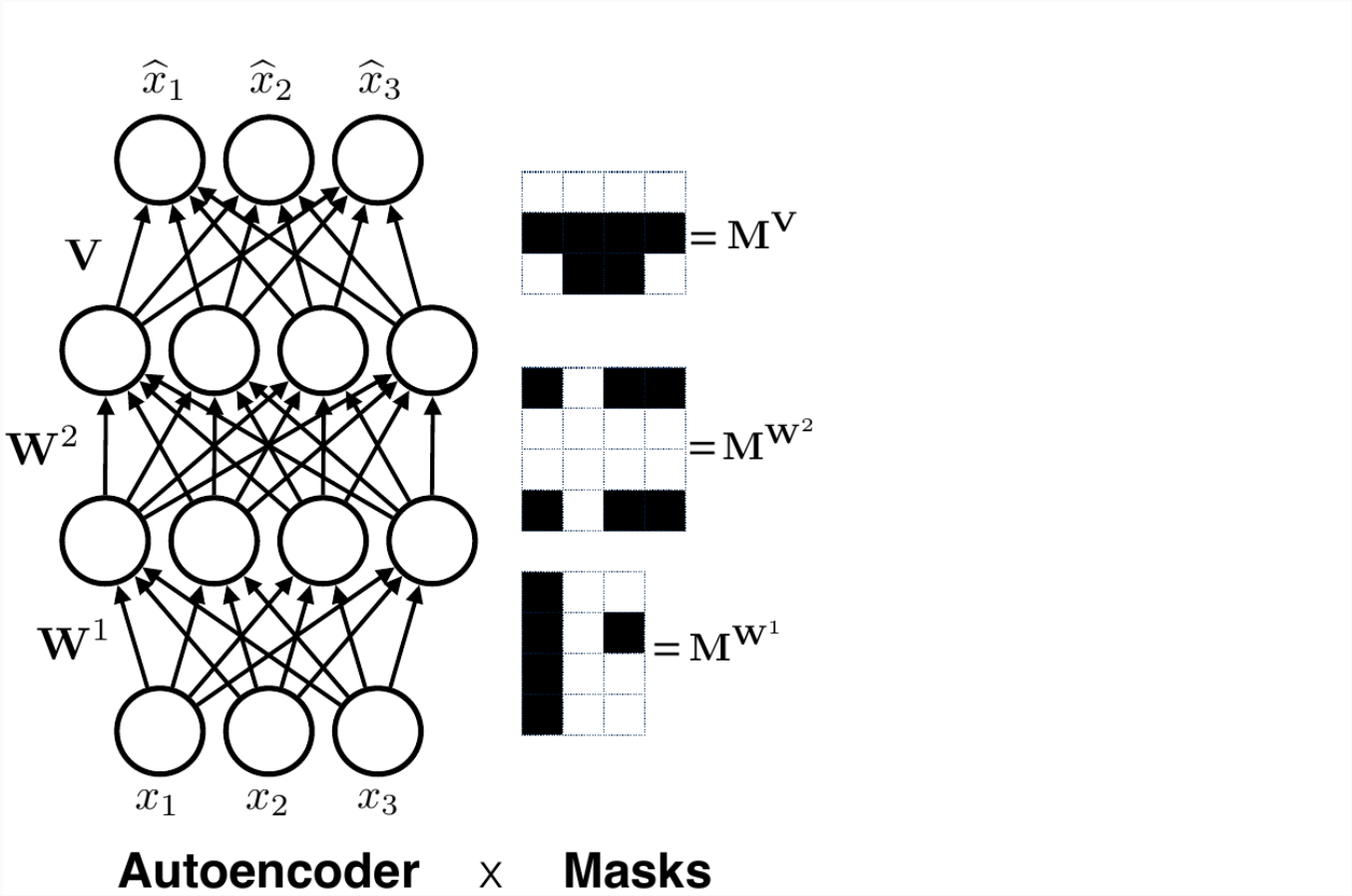
Masked Autoencoder for Distribution Estimation (MADE) [?]

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



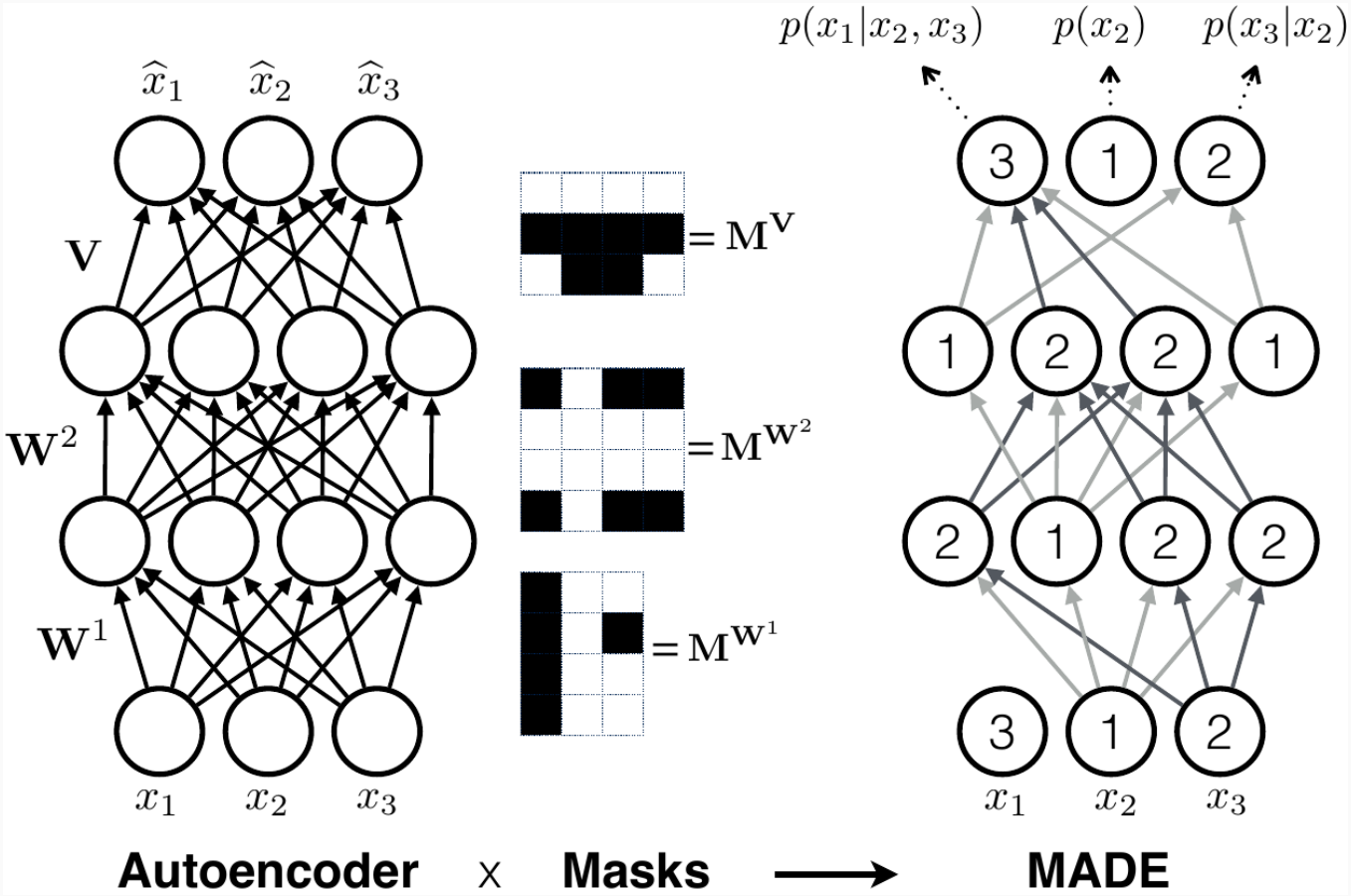
Masked Autoencoder for Distribution Estimation [?]

- 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