Unsupervised & Generative DL

Davide Bacciu

Dipartimento di Informatica
Università di Pisa

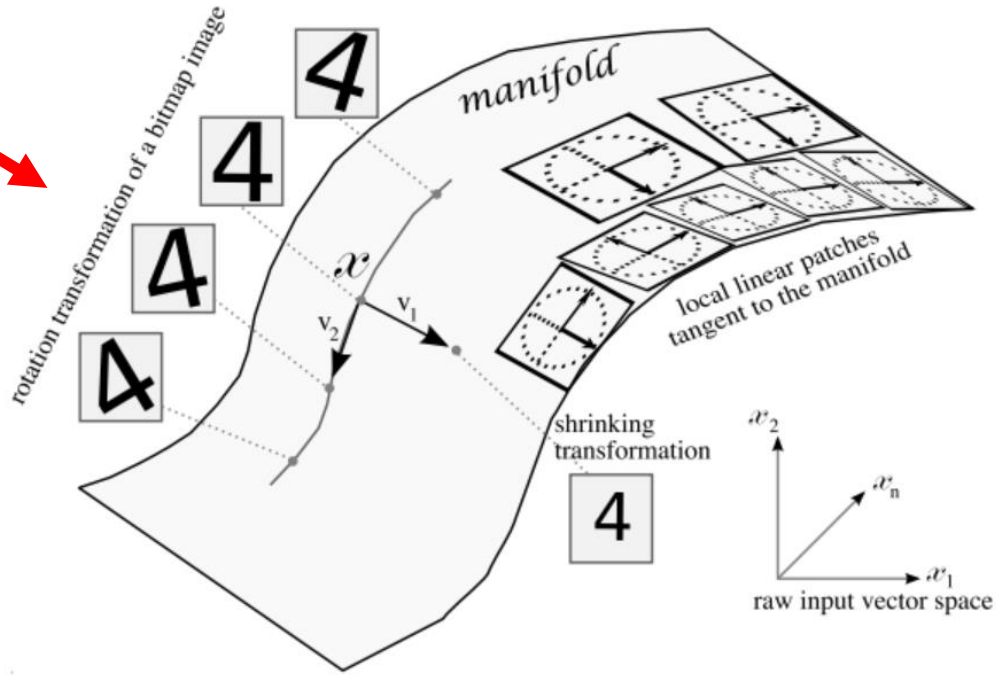
Intelligent Systems for Pattern Recognition (ISPR)

Lecture Outline

- Motivation
  - Why unsupervised?
  - Why generative?
- The DL way to generative learning
  - Learning distributions with fully visible information (RNN)
  - Learning distribution with latent information (VAE)
  - Learning to sample (GAN)
- Applications
  - Generating faces and bedrooms
  - Latent space arithmetic

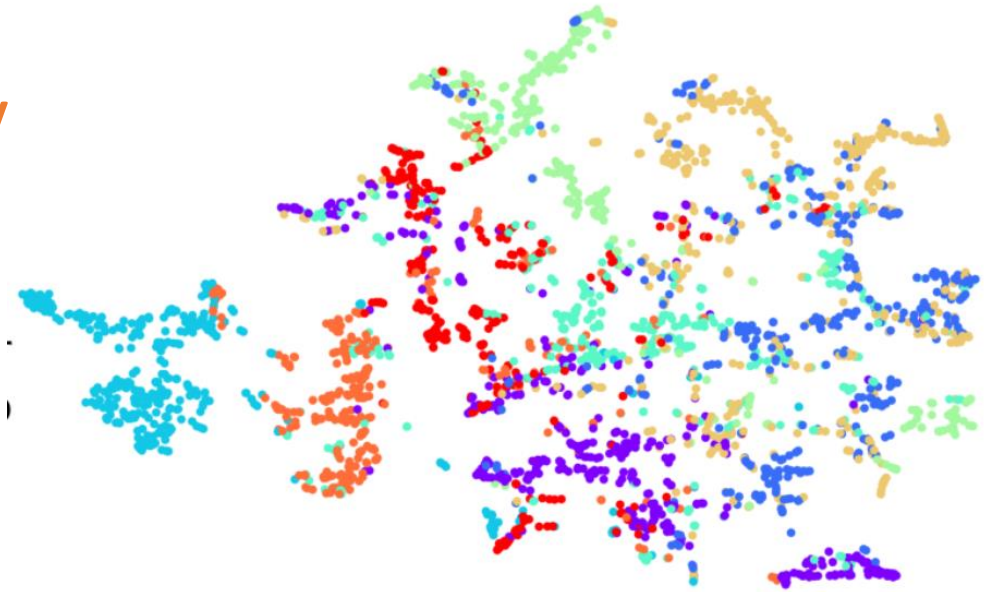
The Problem

- Characterize the data
  - Data distribution
  - Data variances
- To allow
  - Understanding data
  - Generating new observations
  - ..and ultimately reasoning



Why Unsupervised?

Labelled data is costly and difficult to obtain

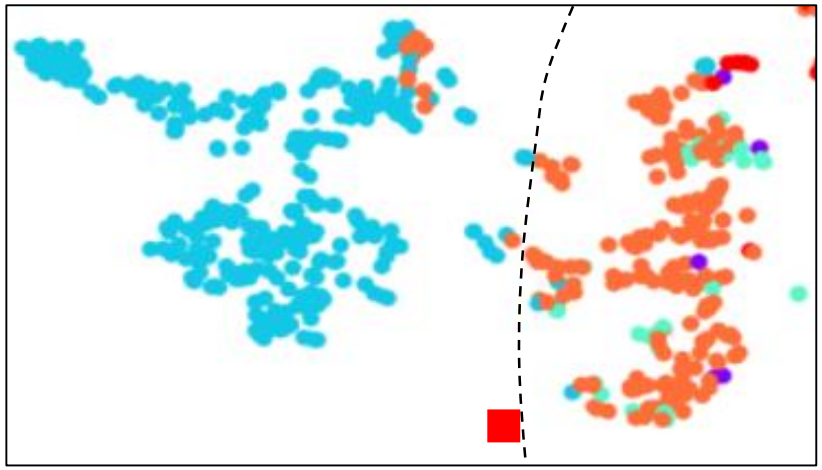


A sustainable future for deep learning

- Learning the latent structure of data
- Discover important features
- Learn task independent representations
- Introduce (if any) supervision only on few samples

Why Generative?

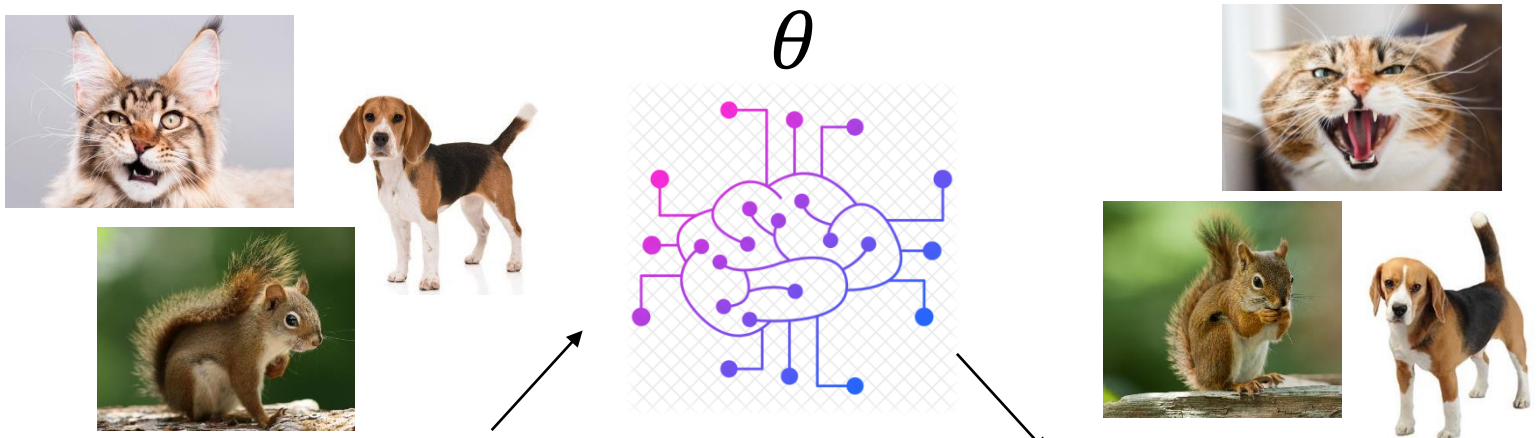
- Focusing too much on discrimination rather than on characterizing data can cause issues
  - Reduced interpretability
  - Adversarial examples



  - Generative models (try to) characterize data distribution
    - Understand the data $\Longrightarrow$ Understand the world
    - Understand data variances $\Longrightarrow$ Learn to steer them
    - Understand normality $\Longrightarrow$ Detect anomalies

Approaching the Problem from a DL Perspective

Given training data, learn a (deep) neural network that can generate new samples from (an approximation of) the data distribution



Training data $\sim P(x)$

Generated data $\sim P_\theta(x)$

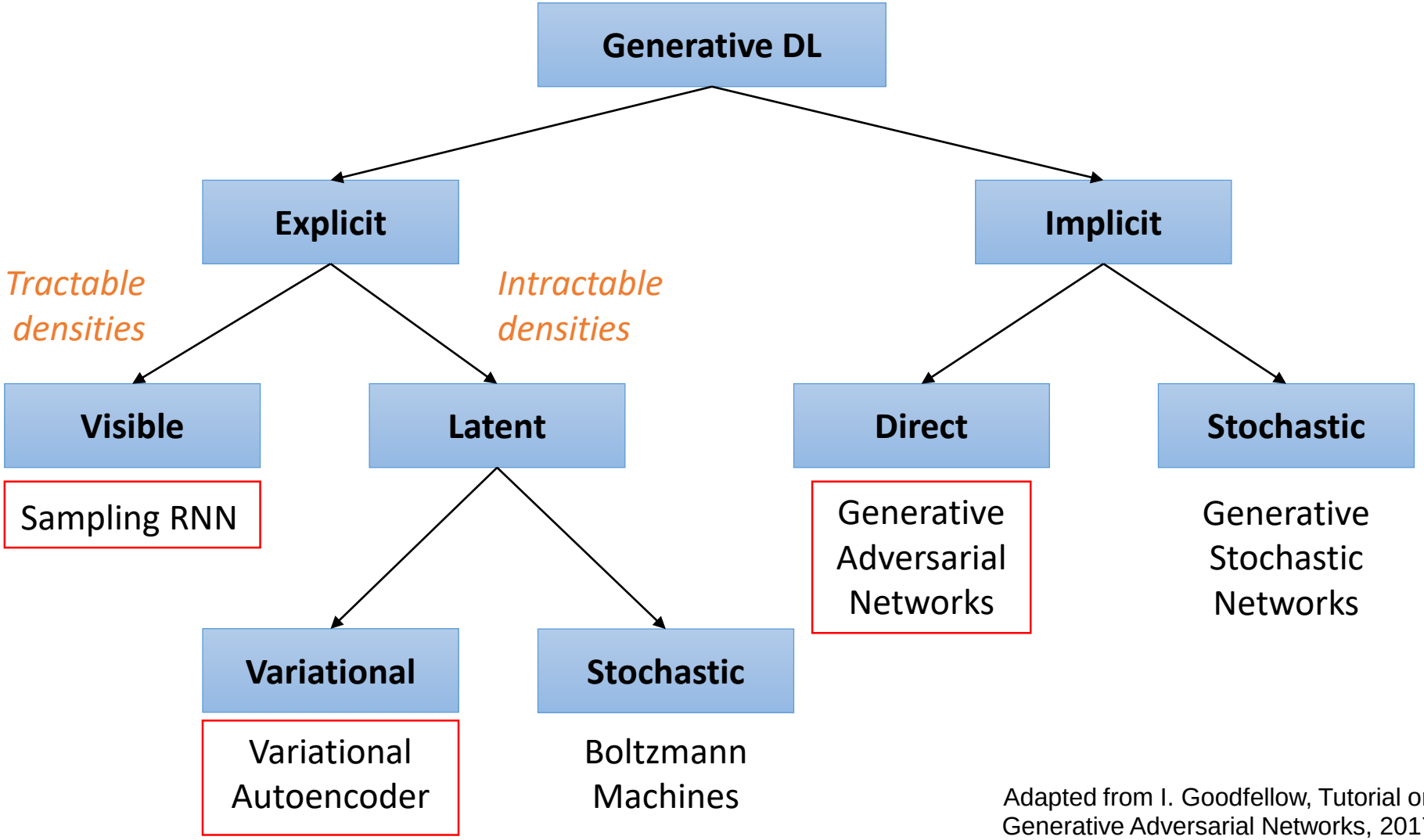
Approaching the Problem from a DL Perspective

Given training data, learn a (deep) neural network that can generate new samples from (an approximation of) the data distribution

Two approaches

- Explicit $\Longrightarrow$ Learn a model density $P_\theta(x)$
- Implicit $\Longrightarrow$ Learn a process that samples data from $P_\theta(x) \approx P(x)$

A Taxonomy



Adapted from I. Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Learning with Fully Visible Information

If all information is fully visible the joint distribution can be computed from the chain rule factorization

Bayesian Networks → $P(\boldsymbol{x}) = \prod_{i}^{N} P(x_i | x_1, \ldots, x_{i-1})$

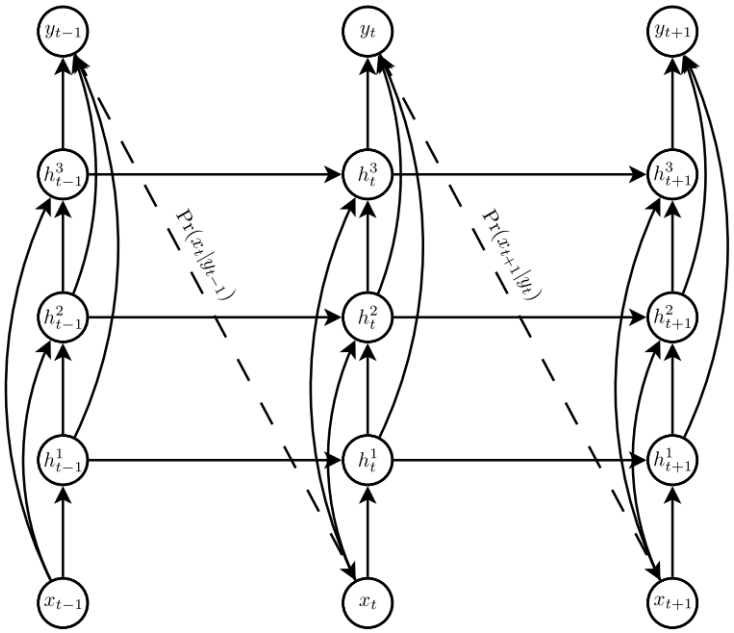
Probability of a pixel having a certain intensity value, given the known intensity of its predecessor

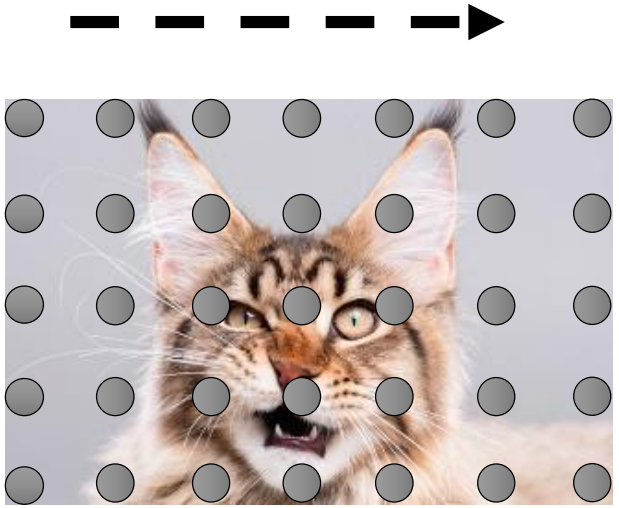
Need to be able to define a sensible ordering for the chain rule

Conditional distribution difficult to compute

Approximating the Conditional Probability

Using a deep learning approach that we have already encountered

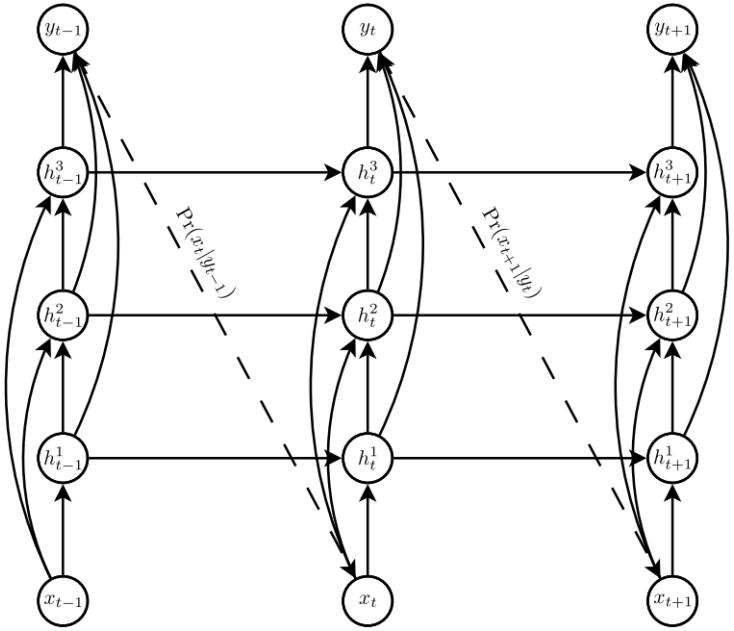


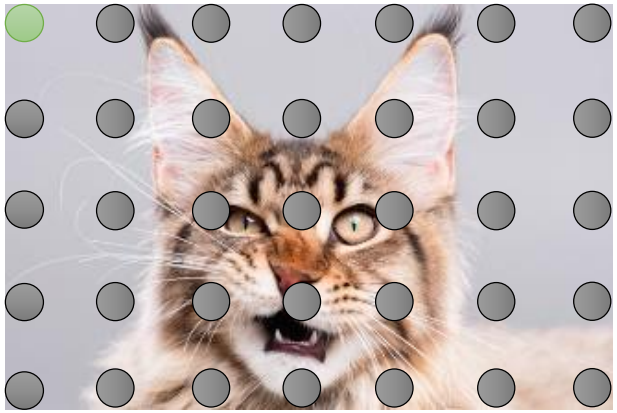


Scan the image according to a schedule and encode the dependency from previous pixels in the states of an RNN

Approximating the Conditional Probability

Using a deep learning approach that we have already encountered

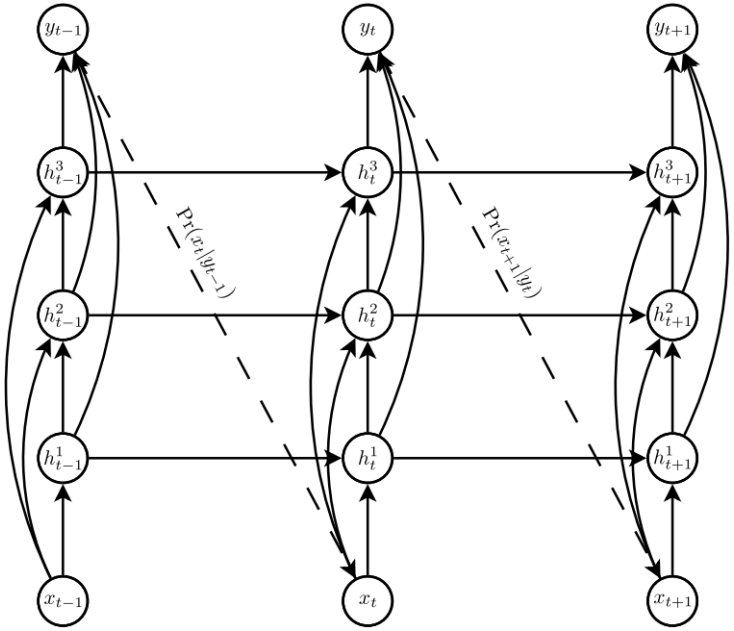


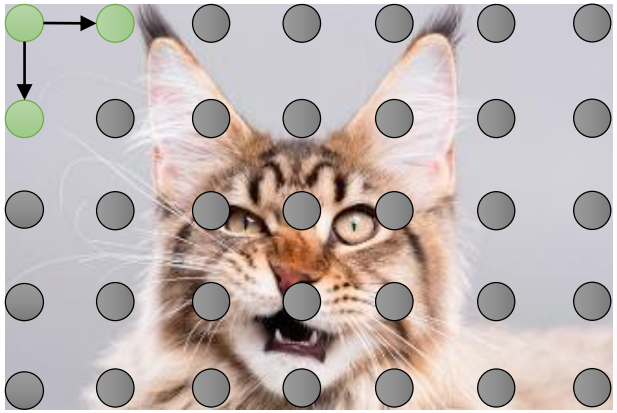


Scan the image according to a schedule and encode the dependency from previous pixels in the states of an RNN

Approximating the Conditional Probability

Using a deep learning approach that we have already encountered

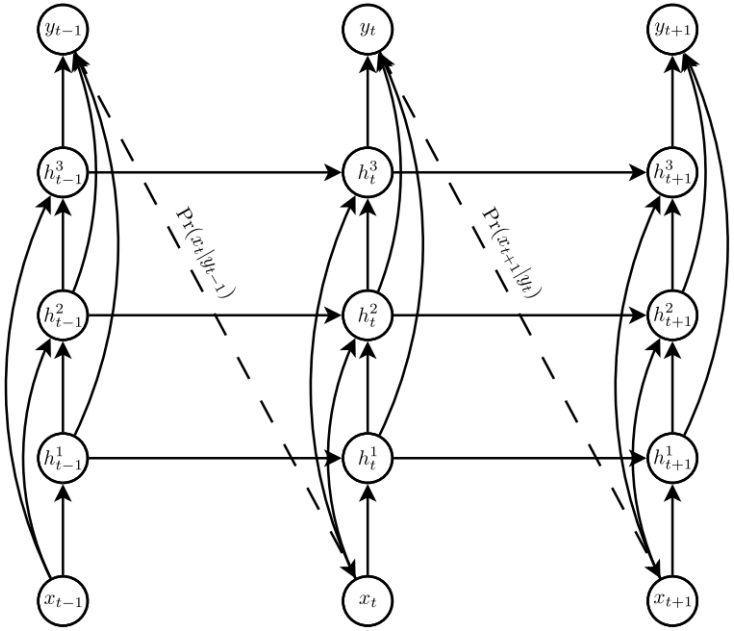


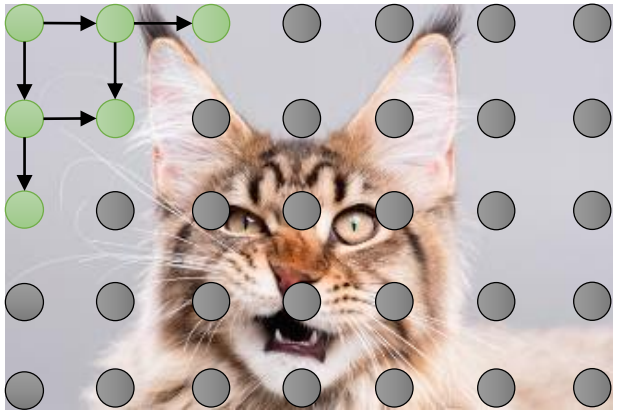


Scan the image according to a schedule and encode the dependency from previous pixels in the states of an RNN

Approximating the Conditional Probability

Using a deep learning approach that we have already encountered

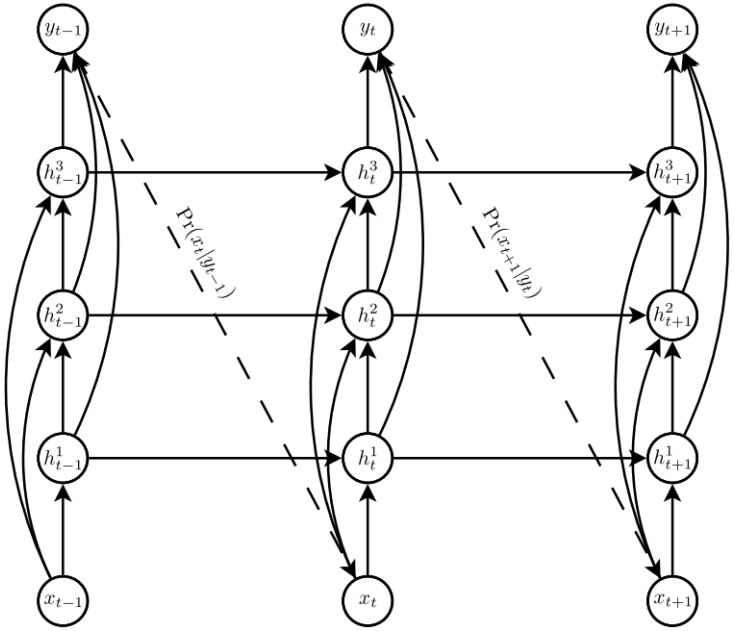


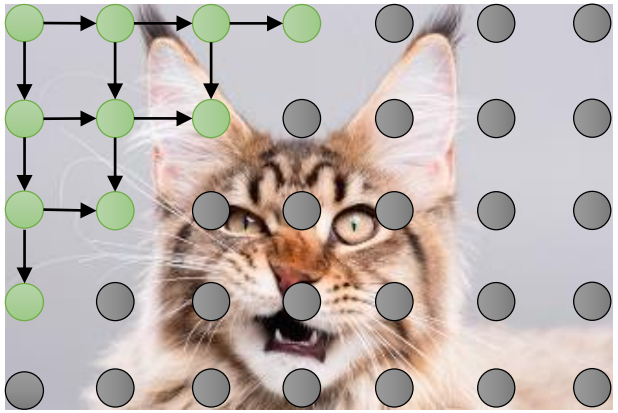


Scan the image according to a schedule and encode the dependency from previous pixels in the states of an RNN

Approximating the Conditional Probability

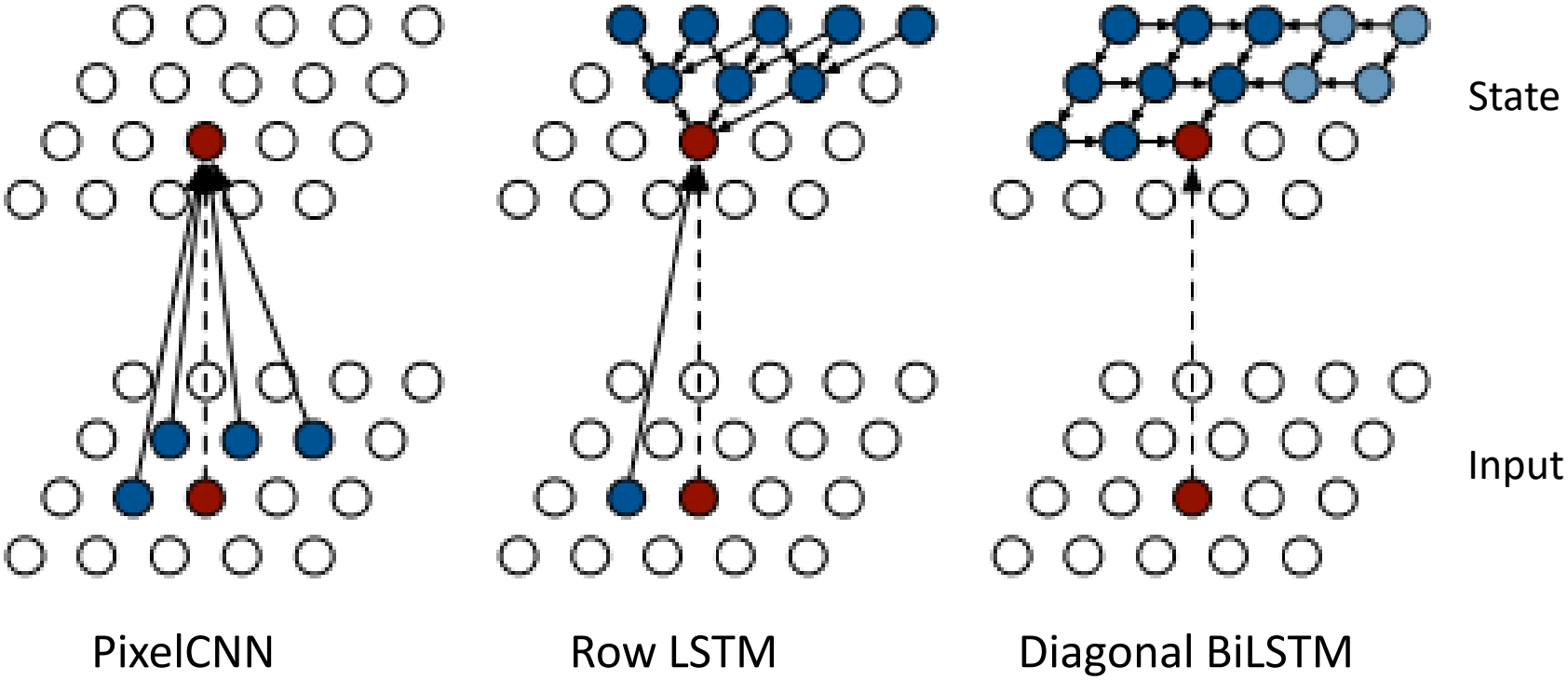
Using a deep learning approach that we have already encountered





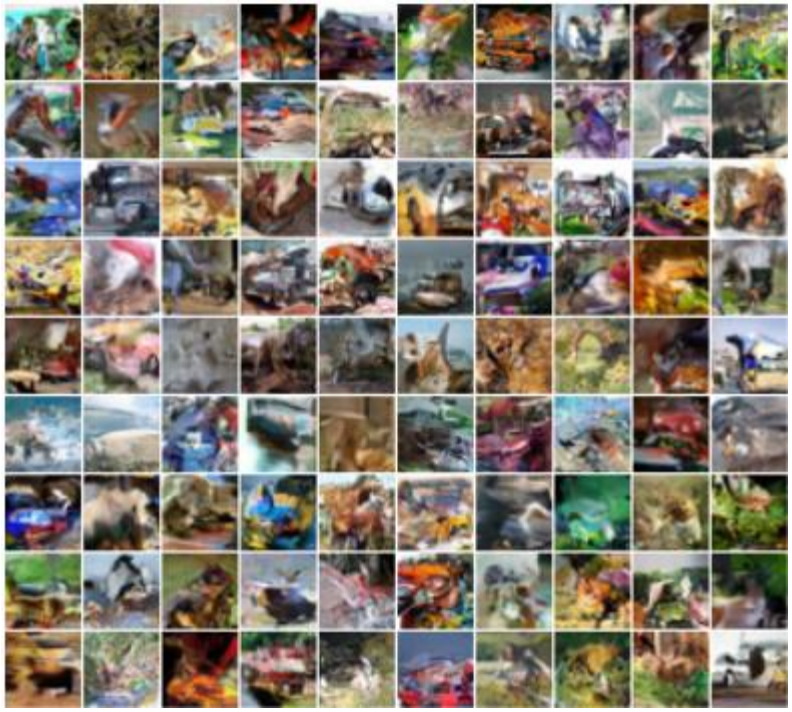
Scan the image according to a schedule and encode the dependency from previous pixels in the states of an RNN

Generating Images Pixel by Pixel



A. van der Oord et al., Pixel Recurrent Neural Networks, 2016

Generating Images Pixel by Pixel - Results



32x32 CIFAR-10

32x32 ImageNet

A. van der Oord et al., Pixel Recurrent Neural Networks, 2016

From Visible to Latent Information

With only visible information, we try to learn the θ parameterized model distribution

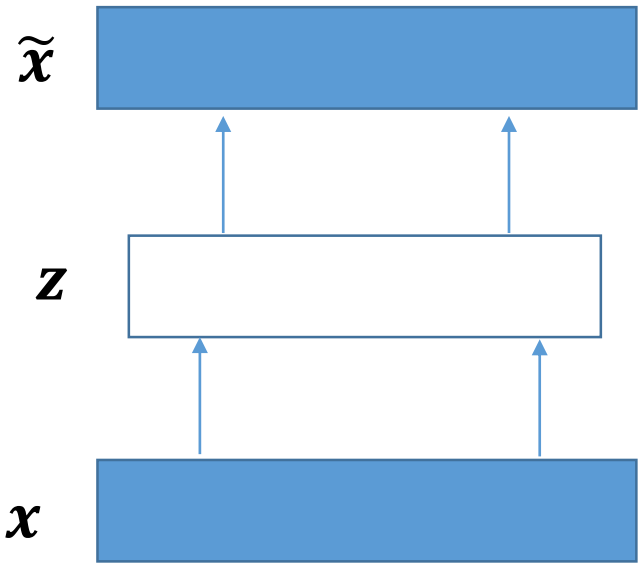
$$P_\theta(\boldsymbol{x}) = \prod_i^N P_\theta(x_i | x_1, \dots, x_{i-1})$$

Now we introduce a latent process regulated by unobservable variables $\boldsymbol{z}$

$$P_\theta(\boldsymbol{x}) = \int P_\theta(\boldsymbol{x}|\boldsymbol{z}) P_\theta(\boldsymbol{z}) d\boldsymbol{z}$$

Typically intractable for non trivial models (cannot be computed for all $\boldsymbol{z}$ assignments)

A Neural Network with Latent Variables?

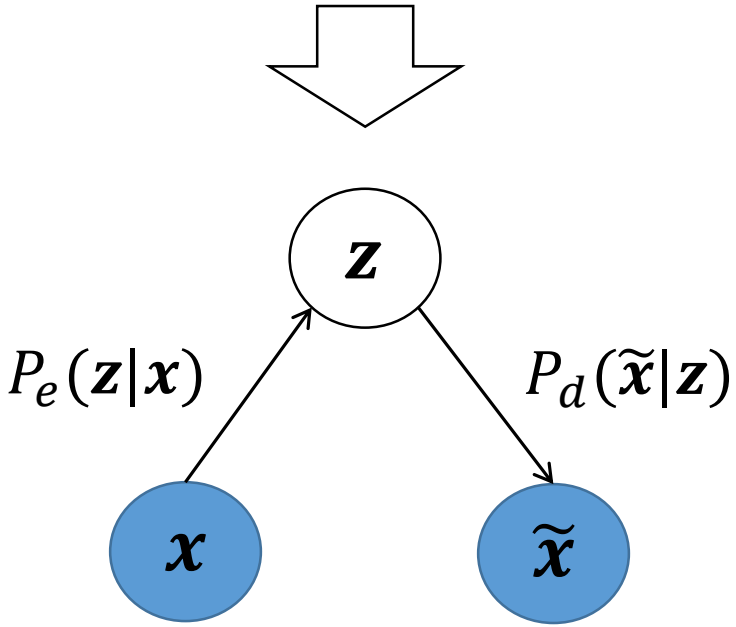


Decoder

Encoder

Autoencoder (AE) neural networks

We have already introduced a probabilistic twist on AE



A Deeper Probabilistic Push

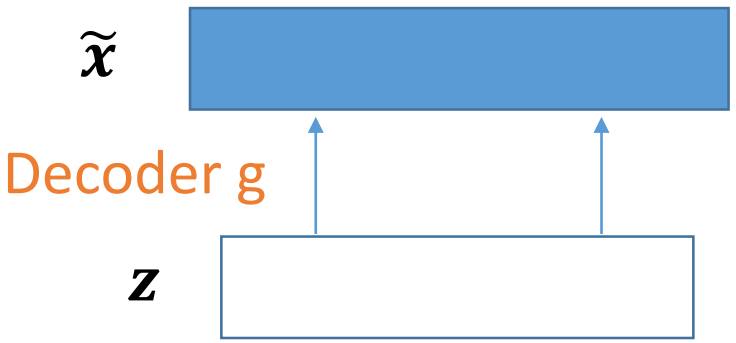
As an additional push in the probabilistic interpretation, we assume to be able to generate the reconstruction from a sampled latent representation

Sample from the true conditional $P(\tilde{x}|z)$

Sample latent variables from the true prior $P(z)$

Of course we don't have access to the true distributions, so how do we approximate them?

Variational Autoencoders (VAE) – The Catch



Represent the $P(\tilde{x}|z)$ distribution through a neural network g (remember the denoising autoencoder)

At training time sample **z** conditioned on data **x** and train the decoder g to reconstruct **x** itself from **z**

$$z \sim \mathcal{N}(\mu(x), \sigma(x))$$

Sample z from a simple distribution such as a Gaussian

VAE Training – Is it all this easy?

Unfortunately for you: no!

Ideally, one would like to train maximizing

$$L(D) = \prod_{i=1}^{N} P(\boldsymbol{x_i}) = \prod_{i=1}^{N} \int P(\boldsymbol{x_i}|\boldsymbol{z})P(\boldsymbol{z})d\boldsymbol{z}$$

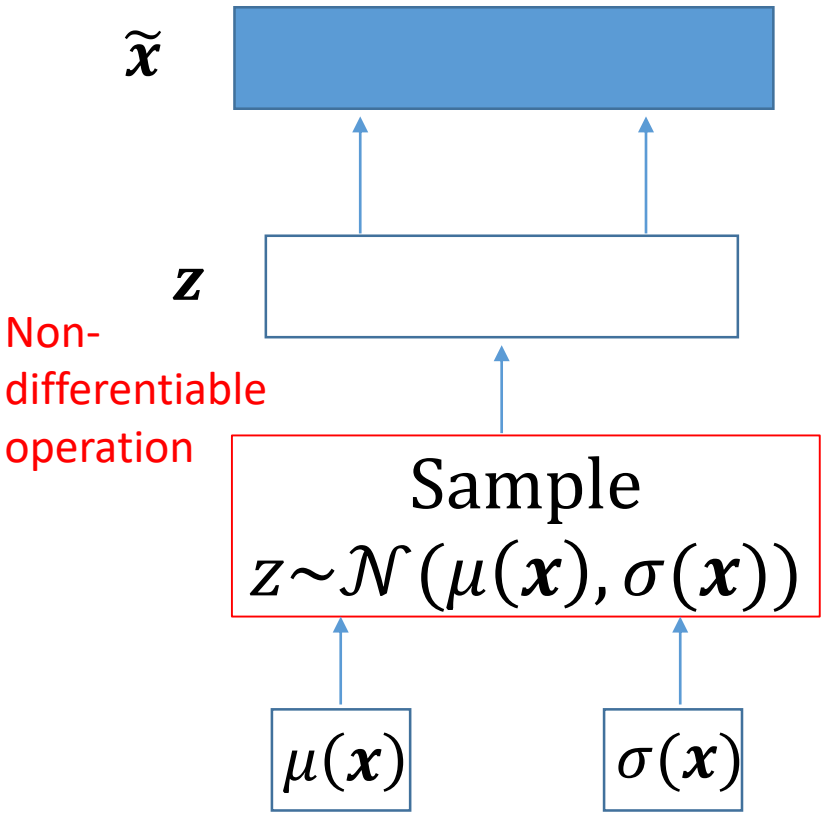
Intractable

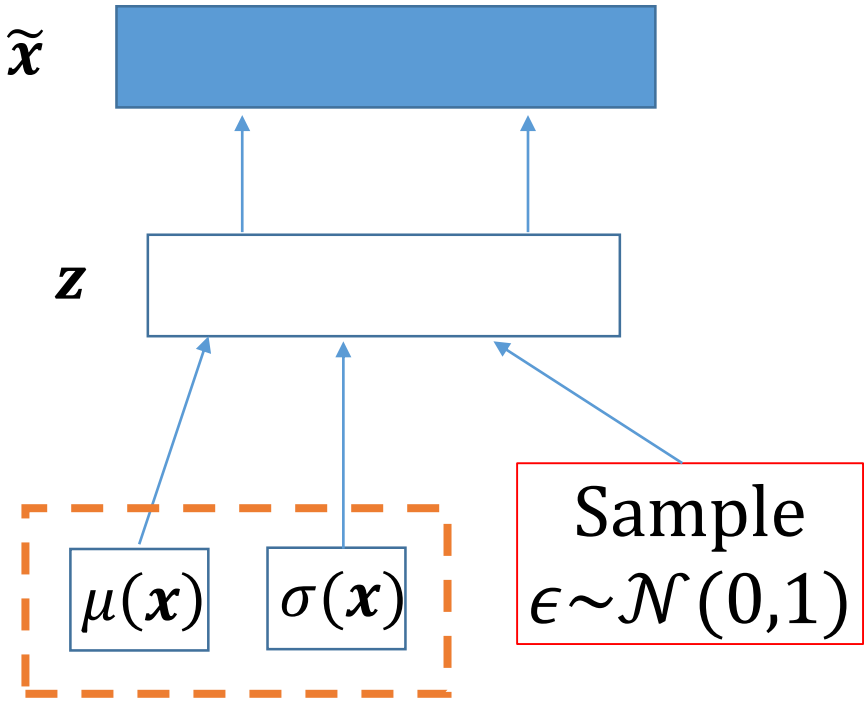
Non differentiable

Variational approximation

Reparameterization

Reparameterization Trick





Sampling is limited to non differentiated variable $\epsilon \Longrightarrow$ Can backpropagate

Variational Approximation

The revenge of the ELBO (Evidence Lower BOund)

$$\log P(x|\theta) \geq \mathbb{E}_Q[\log P(x,z)] - \mathbb{E}_Q[\log Q(z)] = \mathcal{L}(x,\theta,\phi)$$

Maximizing the ELBO allows approximating from below the intractable log-likelihood $\log P(x)$

$$\mathcal{L}(x,\theta,\phi) = \mathbb{E}_Q[\log P(x|z)] + \mathbb{E}_Q[\log P(z)] - \mathbb{E}_Q[\log Q(z)]$$

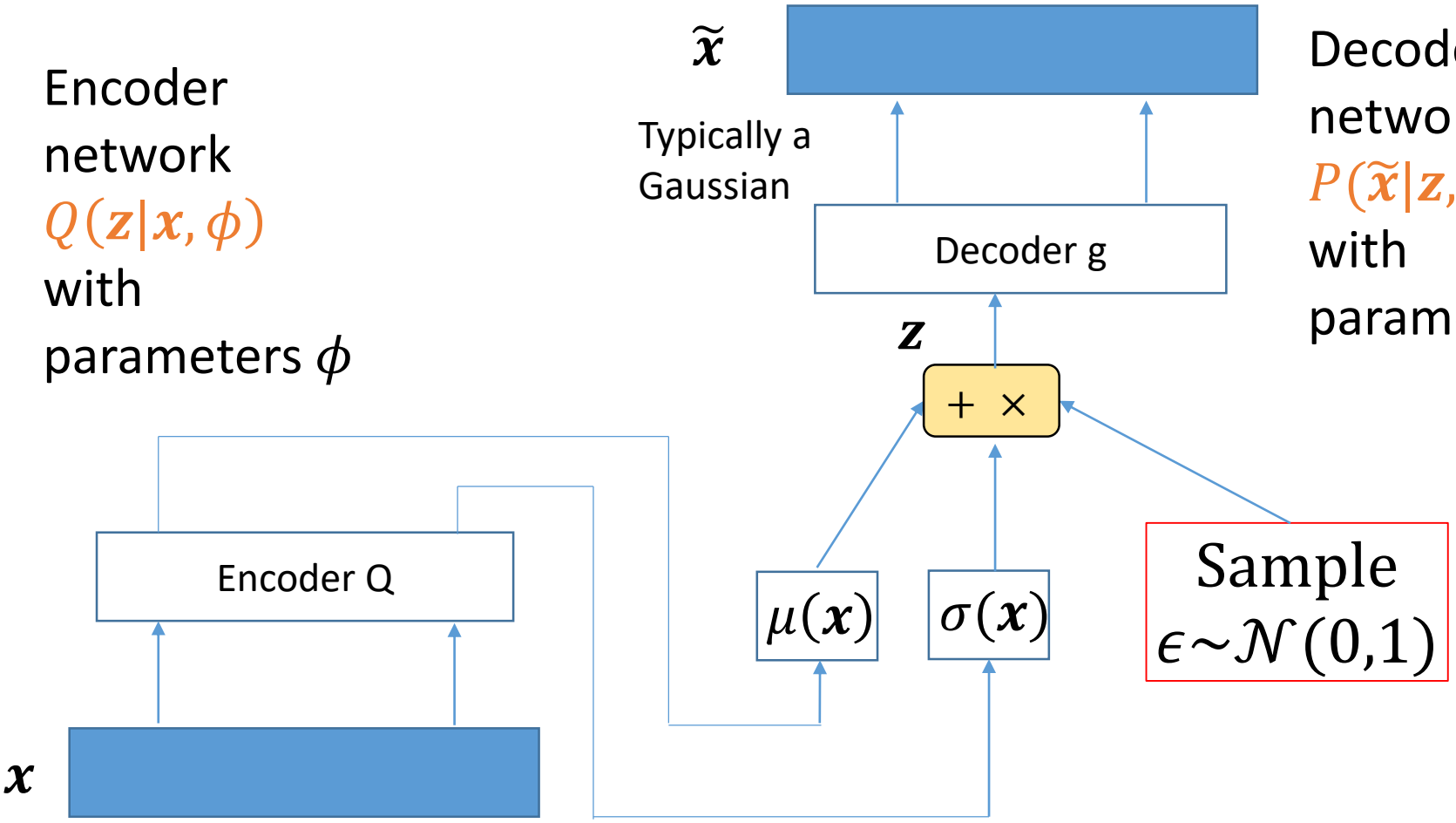
$$KL(Q(z|\phi)||P(z|\theta))$$

Decoder estimate of the conditional, made possible and differentiable through the reparameterization trick

Need a Q(z) function to approximate P(z)

Variational Autoencoder – The Full Picture

Encoder network $Q(\boldsymbol{z}|\boldsymbol{x},\phi)$ with parameters ϕ



Decoder network $P(\tilde{\boldsymbol{x}}|\boldsymbol{z},\theta)$ with parameters θ

Training time architecture

VAE Training

Training is performed by backpropagation on θ, ϕ to optimize the ELBO

reconstruction

$$\mathcal{L}(x, \theta, \phi) = \mathbb{E}_Q\left[\log P(x|z = \mu(x) + \sigma^{1/2}(x) * \epsilon, \theta)\right]$$
$$- KL(Q(z|x, \phi)||P(z|\theta))$$

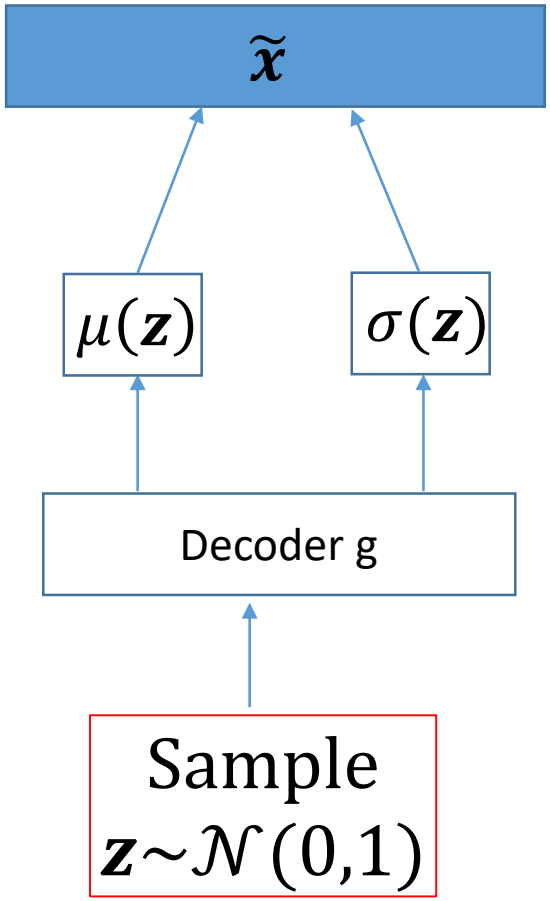
regularization

Can be computed in closed form when both Q(z) and P(z) are Gaussians

$$KL(\mathcal{N}(\mu(x), \sigma(x)) \,||\mathcal{N}(0,1))$$

Train the encoder to behave like a Gaussian prior with zero-mean and unit-variance

Sampling the VAE (a.k.a. testing)



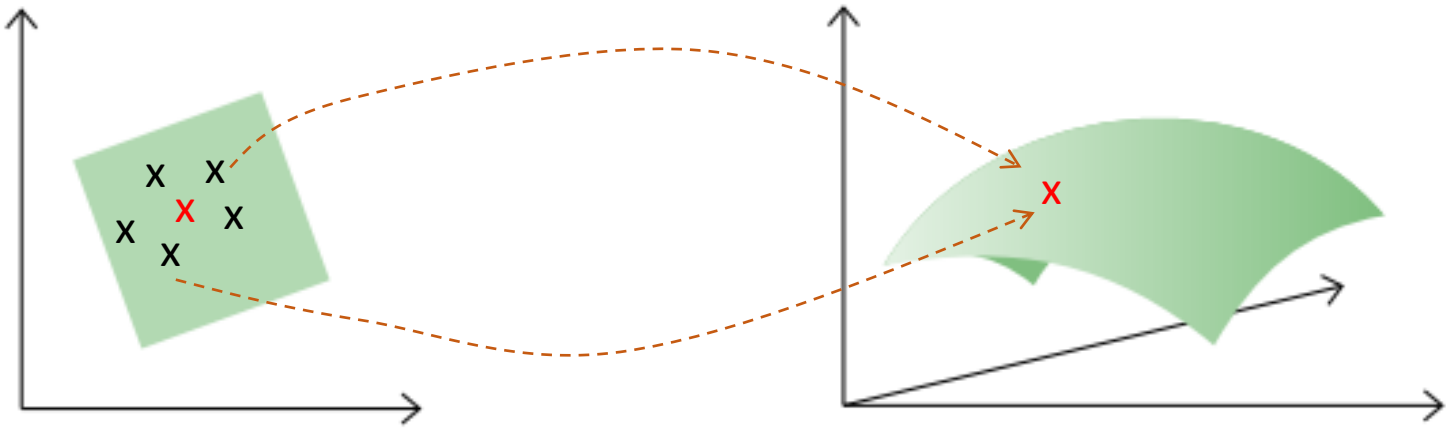
Sample
$\tilde{x} \sim \mathcal{N}(\mu(z), \sigma(z))$

At test time detach the encoder, sample a random encoding and generate the sample as the corresponding reconstruction

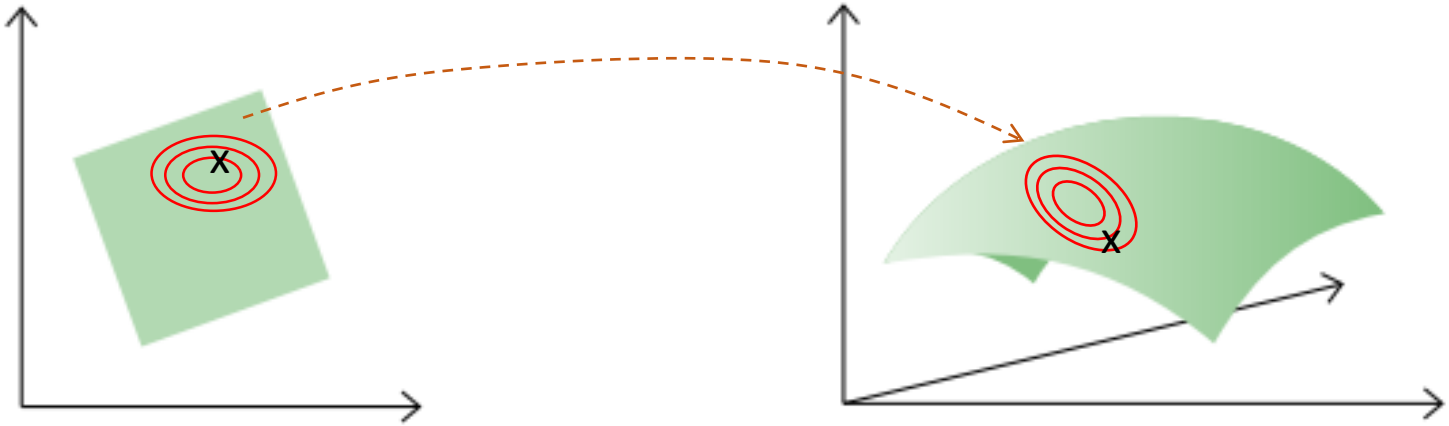
VAE vs Denoising/Contractive AE

Contractive AE

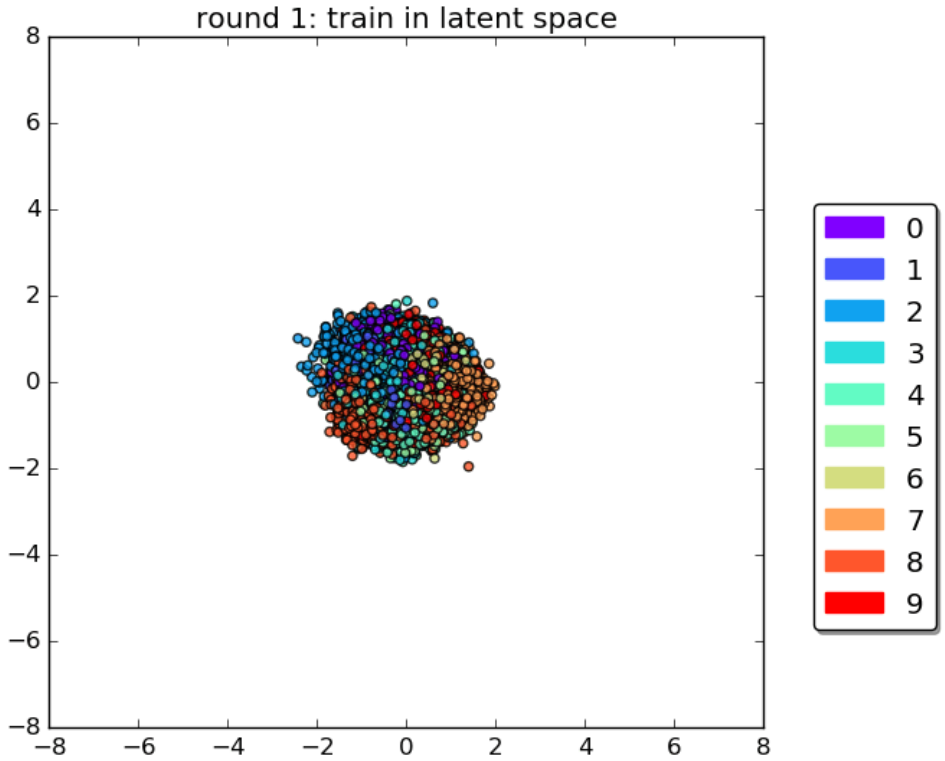
x noisy samples x original sample



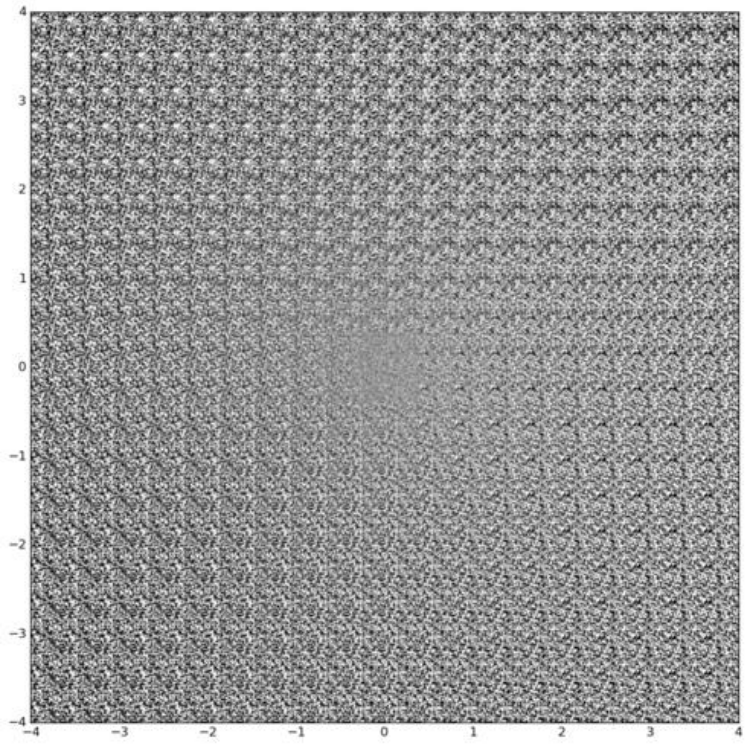
Variational AE



VAE Examples - Digits



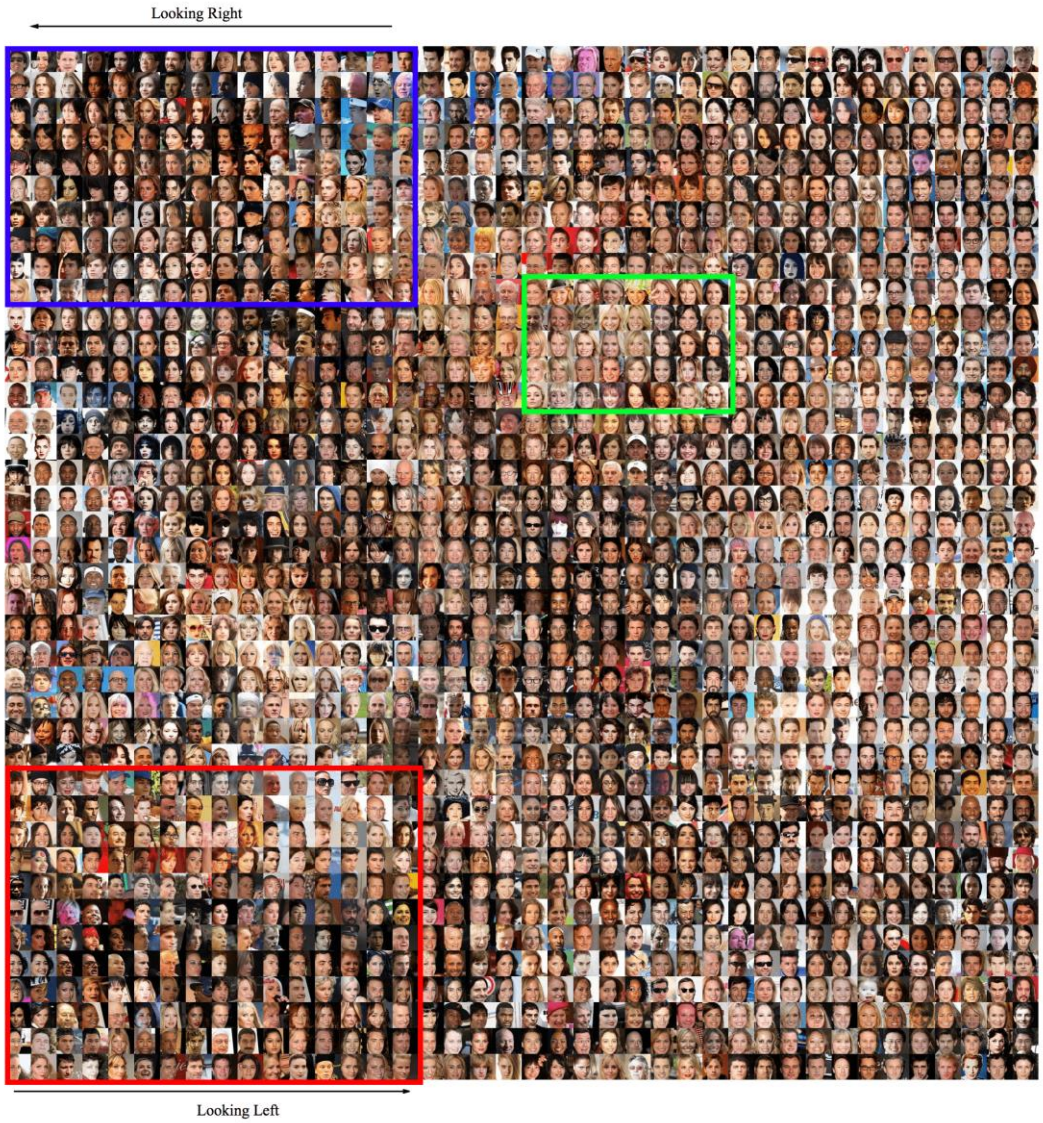
Organization of data in the latent space

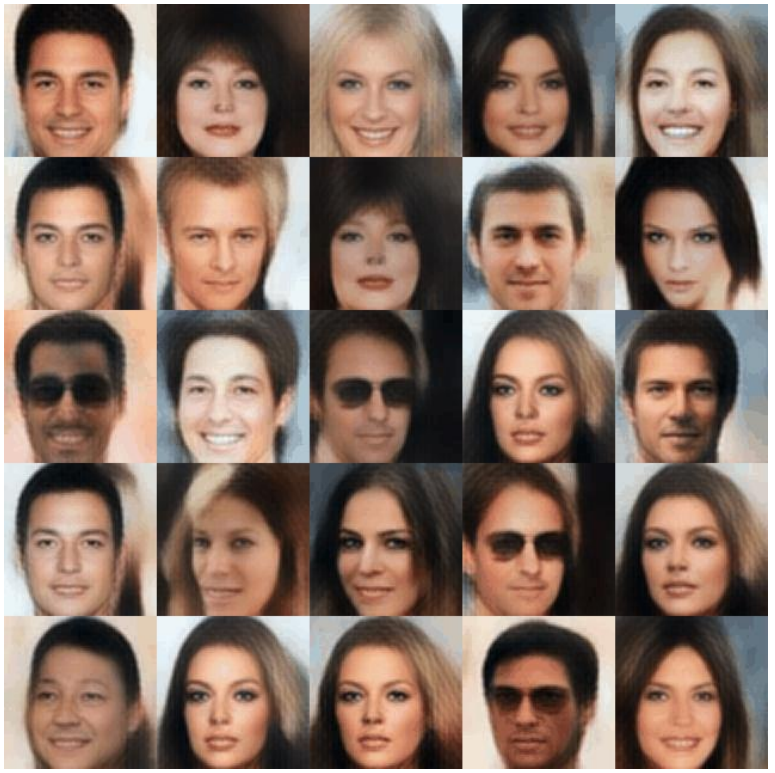


Reconstruction of points sampled from latent space

Image credits @ fastfowardlabs.com

VAE Examples - Faces





Latent space interpolation

Hou et al, Deep Feature Consistent Variational Autoencoder, 2017

Distribution Learning Vs Learning to Sample

- Variational AEs learn to approximate an intractable distribution

$$P_\theta(\boldsymbol{x}) = \int P_\theta(\boldsymbol{x}|\boldsymbol{z}) P_\theta(\boldsymbol{z}) d\boldsymbol{z}$$

  then sample it to generate the output

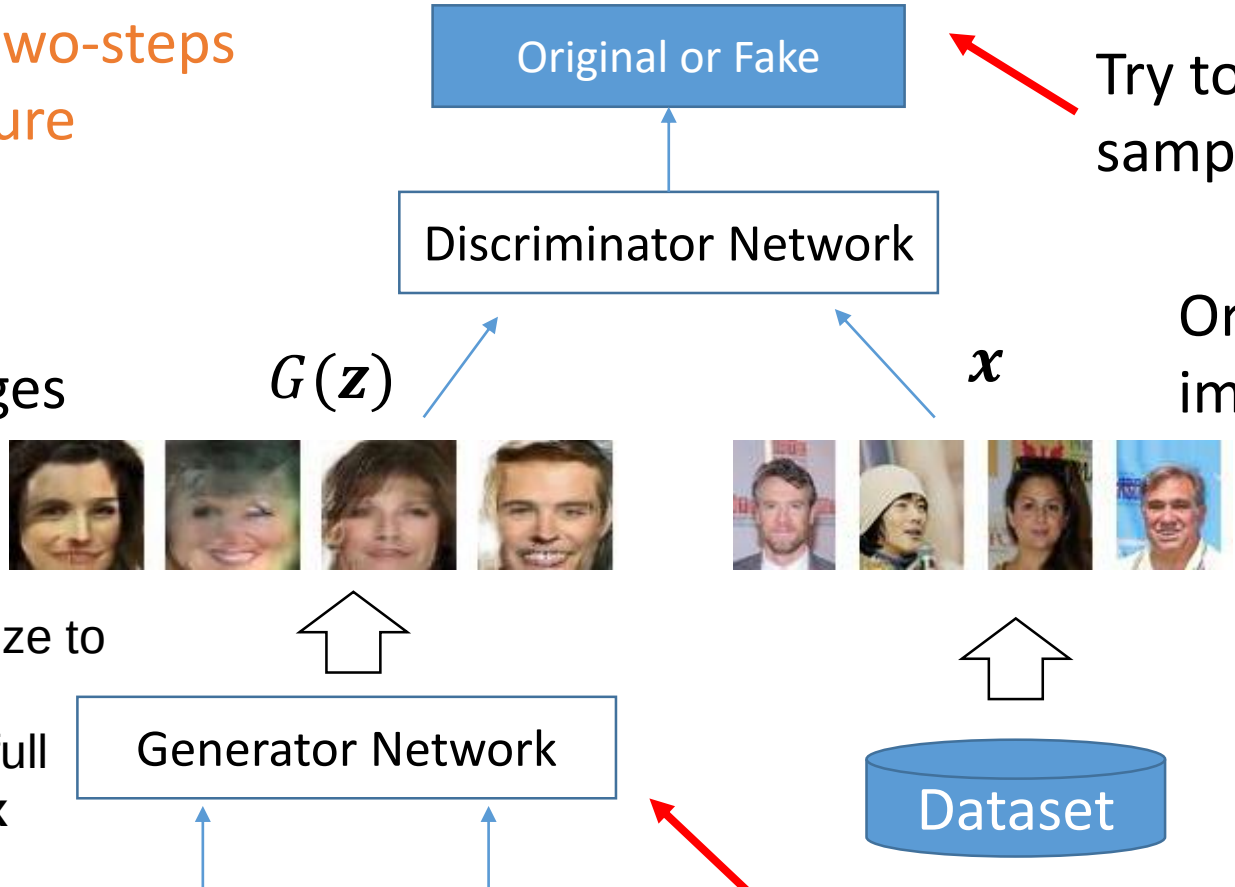
- What if we learn to generate samples rather than learning the distribution?
  - Generative Adversarial Networks (GAN)
  - Game theoretic approach

The GAN Catch

- We need to learn to sample from a complex, high-dimensional training distribution
  - No straightforward way to do this
- The catch
  - Sample from a simple distribution: random noise
  - Train a differentiable function (neural network) to transform random noise to the training distribution

Generative Adversarial Networks

Again, a two-steps architecture



GAN Training - A Game for 2 Players

$$C = \min_{\theta_G} \max_{\theta_D} \left[\mathbb{E}_x \left[\log D_{\theta_D}(x) \right] - \mathbb{E}_z \left[\log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right] \right]$$

Discriminator output for real data *x*

Discriminator output for fake data G(z)

- Discriminator output is likelihood of input being real
- Discriminator tries to maximize C s.t.
  - $D_{\theta_D}(x) \rightarrow 1$ and $D_{\theta_D}(G_{\theta_G}(z)) \rightarrow 0$
- Generator tries to minimize C s.t.
  - $D_{\theta_D}(G_{\theta_G}(z)) \rightarrow 1$

Alternate Optimization

$$C = \min_{\theta_G} \max_{\theta_D} \left[\mathbb{E}_x \left[\log D_{\theta_D}(x) \right] - \mathbb{E}_z \left[\log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right] \right]$$

1. Discriminator gradient ascent

$$C_D = \max_{\theta_D} \left[\mathbb{E}_x \left[\log D_{\theta_D}(x) \right] - \mathbb{E}_z \left[\log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right] \right]$$

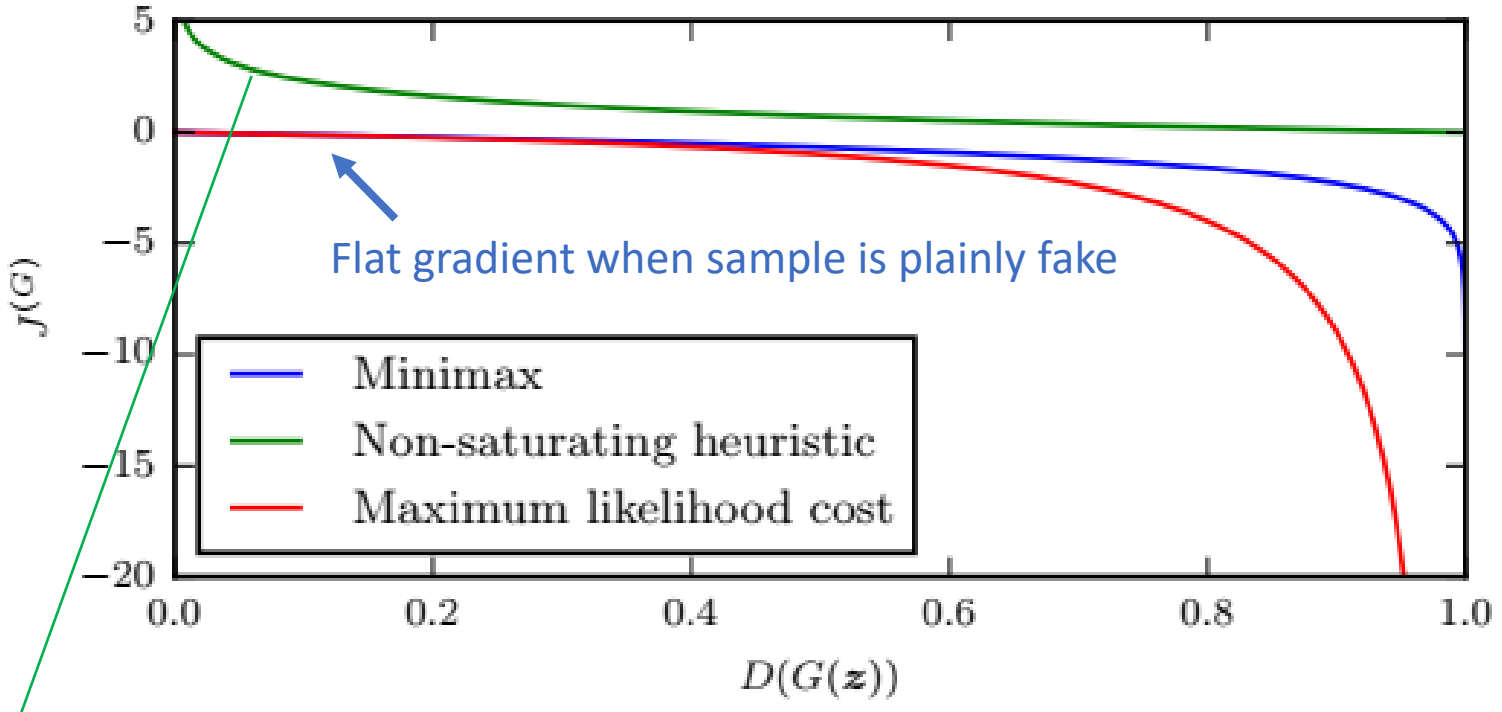
2. Generator gradient descent

$$C_G = \min_{\theta_G} \left[\mathbb{E}_z \left[\log(1 - D_{\theta_D}(G_{\theta_G}(z))) \right] \right]$$

Optimizing this doesn't really work

The Issue and a Solution

The cost that the Generator receives in response to generate $G(\boldsymbol{z})$ depends only on the Discriminator response

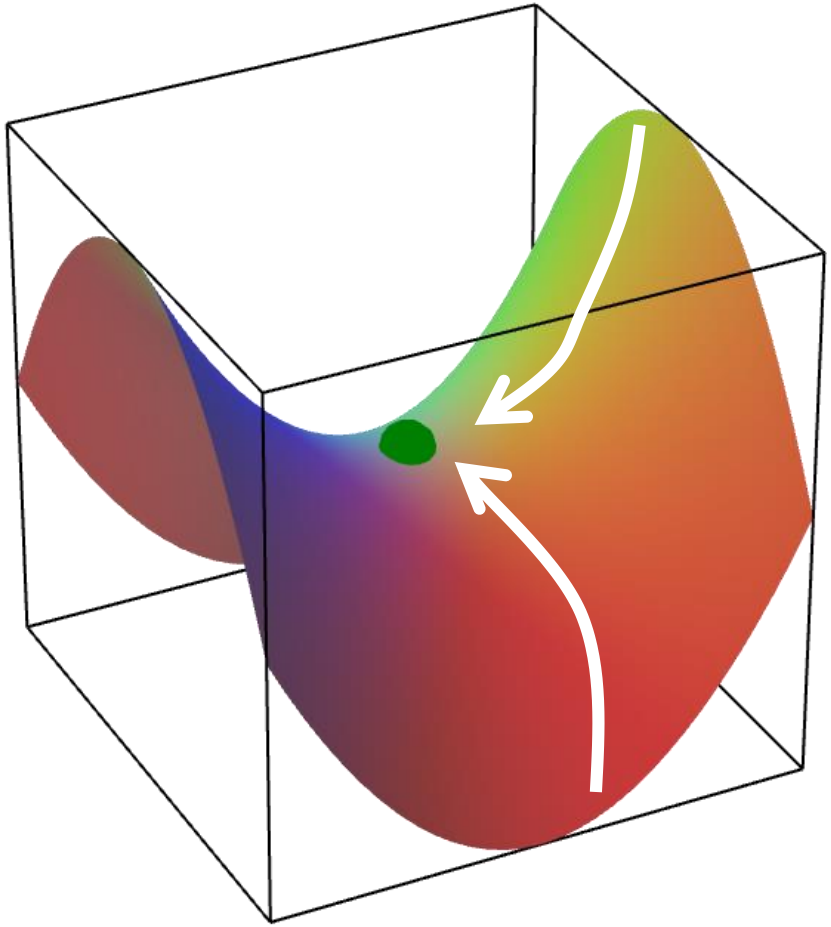


$$C_G = \max_{\theta_G} \left[\mathbb{E}_z \left[\log\left(D_{\theta_D}\left(G_{\theta_G}(z) \right) \right) \right] \right]$$

maximize likelihood of discriminator being wrong

A Hard Two-Player Game

- The optimal solution of the min-max problem is a saddle point
- Little stability
  - Lot of heuristic work
  - Open problem



GAN Training Pseudo-Algorithm

Stability trick but difficult to choose k

for number of training iterations **do**

for k steps **do**

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

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

end for

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

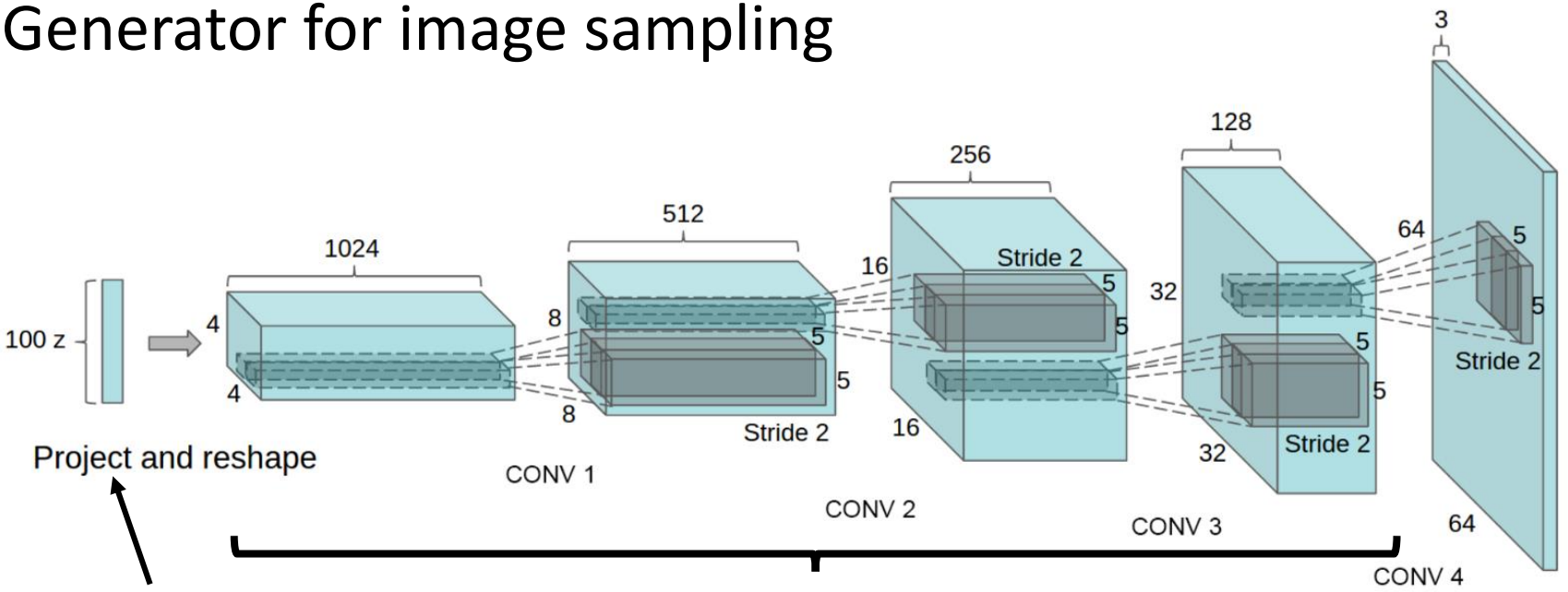
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Expectation

The DCGAN Architecture

Generator for image sampling



Random vector

Upconvolution + batch normalization

Fake image

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

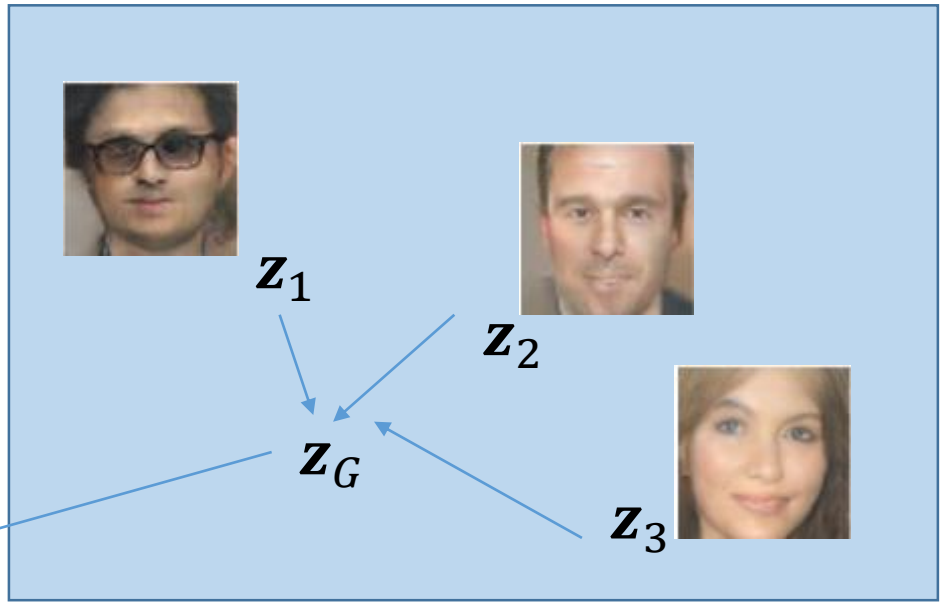
GAN and Images

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Latent Space Arithmetic

Can do sensible linear operations on noise vectors (arithmetic, interpolation)

$$z_G = z_1 - z_2 + z_3$$

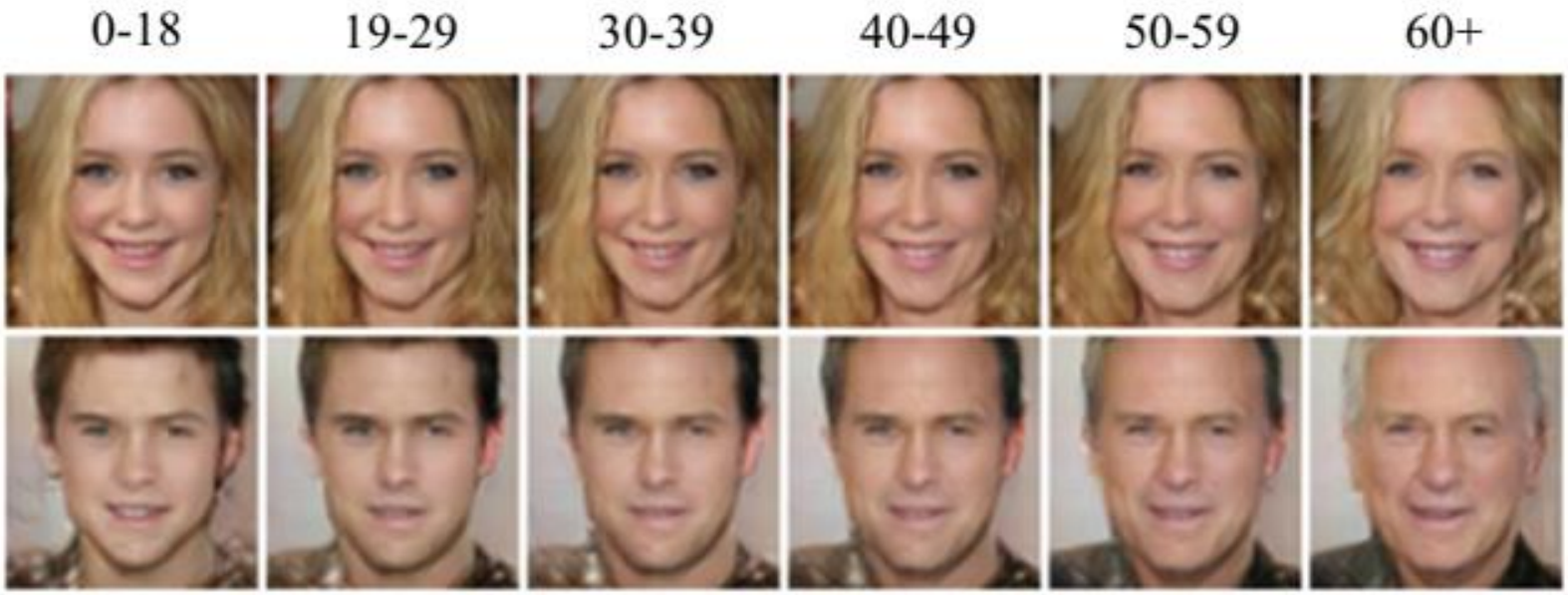


Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Conditional Generation

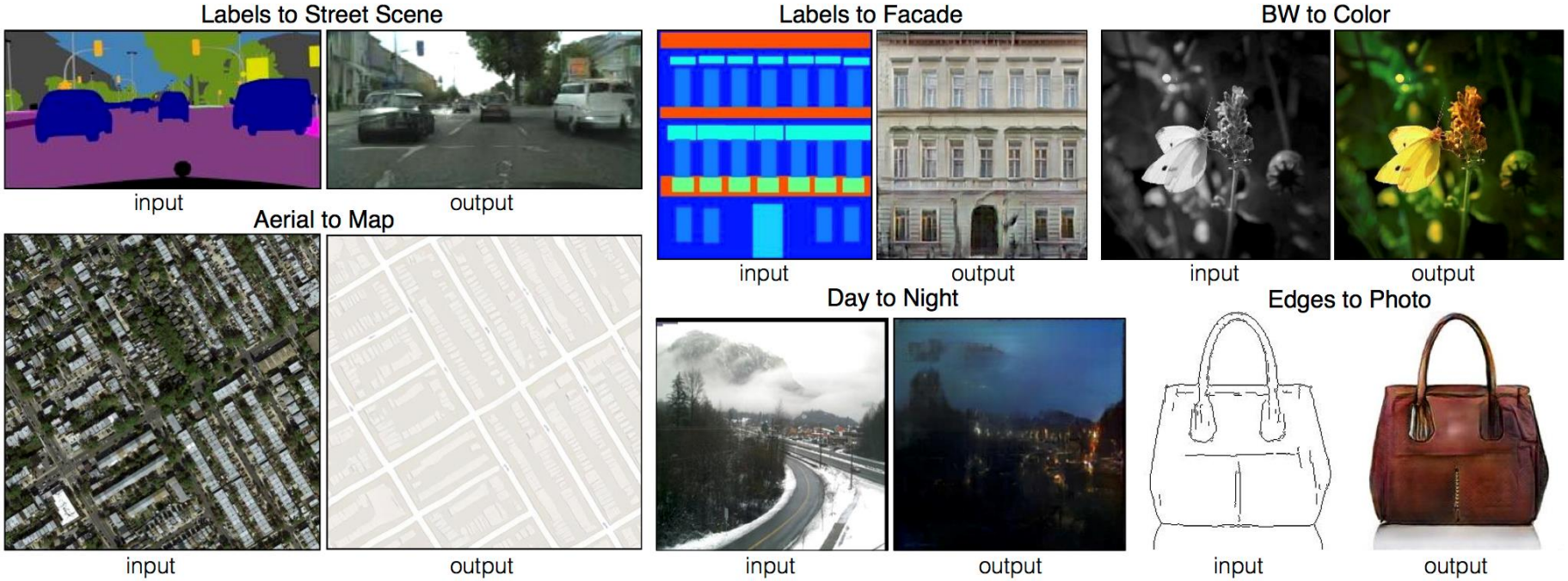
Learn a mapping from an observed side information **x** and a random noise vector **z** to the fooling samples **y**

$$G: \{x, z\} \rightarrow y$$



Antipov et al, “Face Aging With Conditional Generative Adversarial Networks”, ICIP 2017

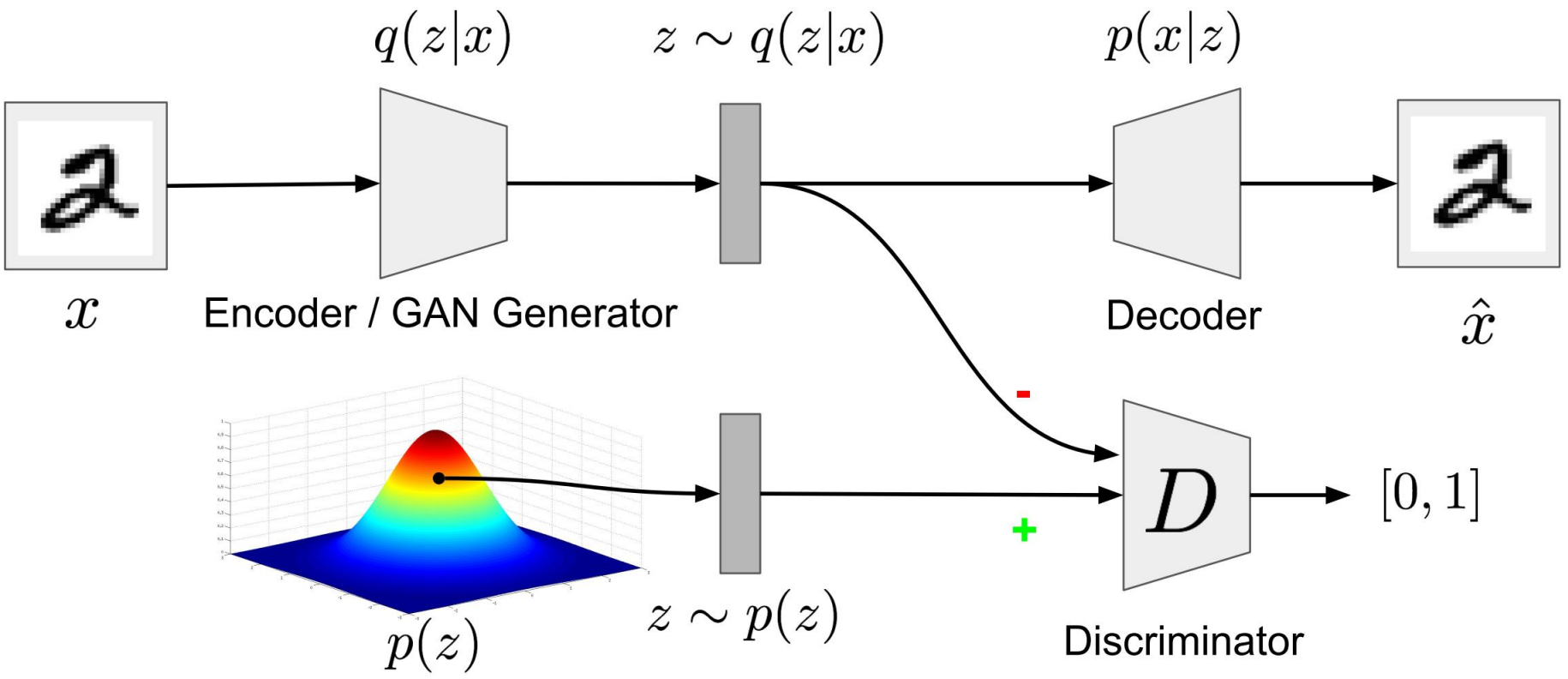
Conditional Generation – Image2Image



Isola et al, “Image-to-Image Translation with Conditional Adversarial Networks”, 2016

Best of 2 worlds?

Adversarial autoencoders (AAE)



Force the latent codes to be indistinguishable from samples of a priori distribution

Training AAE

$$\mathcal{L}(x) = \mathbb{E}_Q[\log P(x|z)] - KL(Q(z|x)||P(z))$$

Replaced by an adversarial loss

- Reconstruction phase - Update the encoder and decoder to minimize reconstruction error
- Regularization phase - Update discriminator to distinguish true prior samples from generated samples; update generator to fool the discriminator
- Adversarial regularization allows to impose priors for which we cannot compute the KL divergence

Wasserstein Distance Models

Attempts to solve the hardness of training adversarial generators by optimizing the Wasserstein distance (EMD) between the generator and empirical distribution filtered trough the discriminator function D

$$G^* = \underset{G}{\operatorname{argmin}}\, \boldsymbol{W}(\mu, \mu_G)$$
$$= \underset{G}{\operatorname{argmin}} \max_{\|D\|_L \leq 1} \left[\mathbb{E}_{x\sim\mu}[D(x)] - \mathbb{E}_{x\sim\mu_G}[D(x)]\right]$$

$\|D\|_L \leq 1$ Requires optimizing D under a constraint on Lipschitz seminorm

- Clipping D weights (slow to converge)

Software

- A TF implementation of PixelCNN
- A list of acknowledged VAE implementations is kept by Kingma here
- Plenty of DCGAN implementations
  - Torch
  - Tensorflow
- Conditional GAN for image-to-image
  - Pytorch code
  - Demo
- So many GANs: check out the GAN-Zoo

Take Home Messages

- PixelRNN/ PixelCNN – Learn explicit distributions by optimizing exact likelihood
  - Yields good samples
  - Inefficient sequential generation
- VAE – Learn complex distributions over latent variables through a variational approximation using neural networks
  - Learns a latent representation useful for inference
  - Poor generated sample quality
- GAN – Learn to sample rather than learn the distribution
  - State of the art generated sample quality
  - Unstable/difficult to train
  - Cannot perform inference (no distribution learning)
  - Need differentiable generator (how to generate discrete items?)