Generative Modeling

ENEE 6583 Neural Nets

Dr. Alsamman

Slide Credits:

Supervised vs Unsupervised Learning

- Supervized (SL) : $\{X, Y\}$: $M\{X\} \rightarrow Y$
  - Given inputs X, corresponding labels (outputs) Y
  - Learn mapping M that maps X to Y
- Unsupervised (USL): $M\{X\} \rightarrow Y$
  - Given only inputs X
  - Find a mapping to Y that optimizes some objective function

Why Unsupervised : No label possible

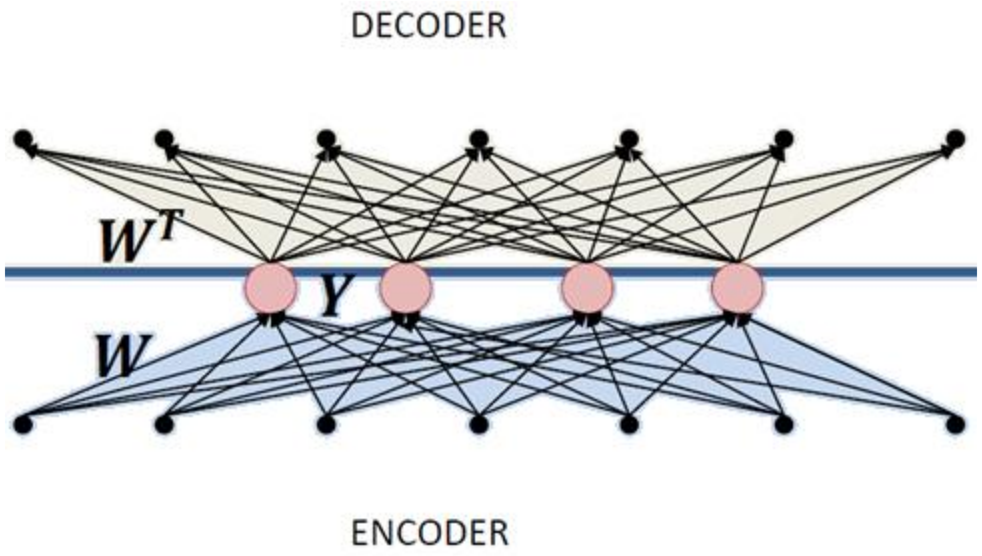
- Hidden data representation
  - Data compression
  - Data organization
  - Explore hidden structures within data
- Applications:
  - Organize computer clusters
  - Group users according to interest
  - Marketing: Recommend products/services
  - Detect fault/intrusion
  - Find similarity
- Driven by an objective function

Why Unsupervised : Price

- Data is
  - Decreasing in price
  - Increasing in: volume, speed,
  - Varying in modality
- Advanced tech =>
  - cheaper tech => cheaper data (price)
  - better sensors => more data (volume, speed)
  - more tech services => user data (modes, speed)
- Expert labeling is expensive
  - Mechanical Turk
    Is 'data labeling' the new blue-collar job of the AI era?
    www.techrepublic.com/article/is-data-labeling-the-new-blue-collar-job-of-the-ai-era/

Autoencoder

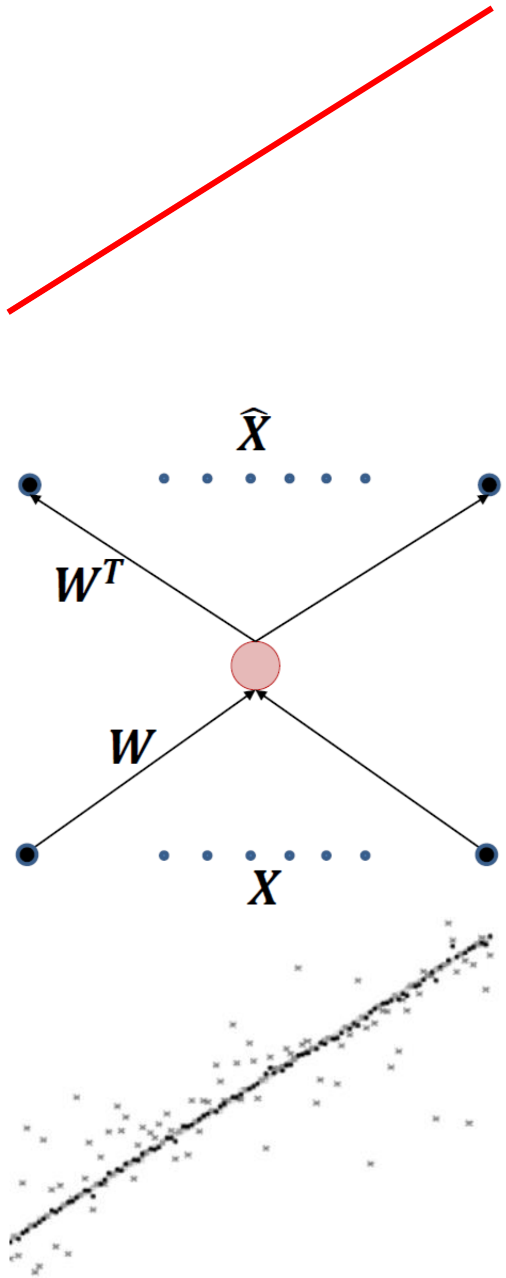
- Encode the input: $M\{X\} \rightarrow Y$
  - Analysis
- Decoder is the reverse: $M^{-1}\{Y\} \rightarrow \hat{X}$
  - Synthesis
  - Identical to encoder network
- Unsupervised learning
- Objective function: $X \approx \hat{X}$
  - $E = \left|X - \hat{X}\right|^2 = 0$
  - Find W for $E \approx 0$



Linear AE

- Linear encoding: Linear activations
- Equations:

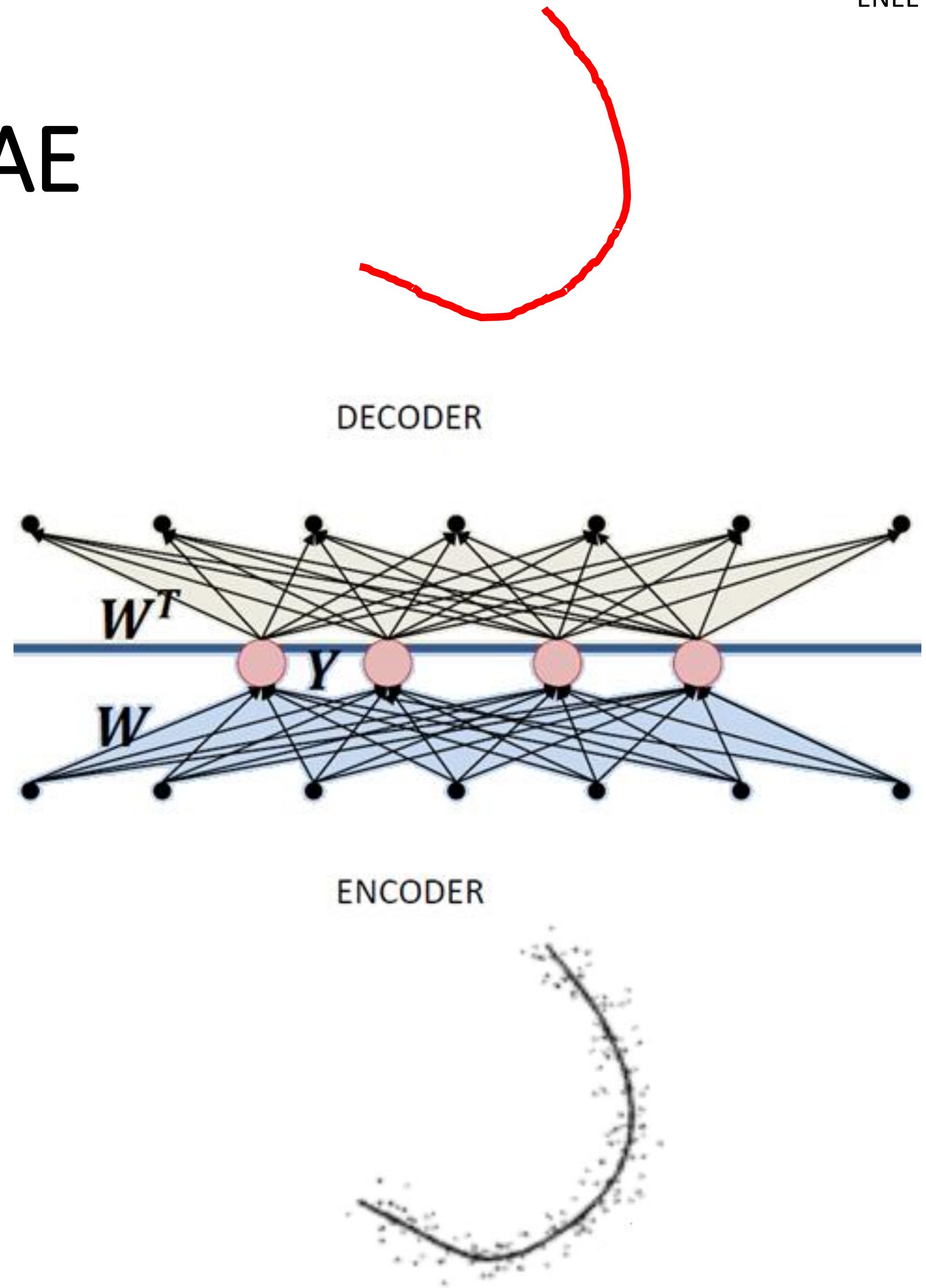
$$Y = WX$$
$$\hat{X} = W^T Y$$
$$E = |X - W^T W X|^2$$



  - W is a principal component (PCA)
  - Line along that max energy
  - Matrix theory: max Eigen vector

Nonlinear AE

- Nonlinear activations
- Nonlinear CA
- Learn the nonlinear manifold



Applications

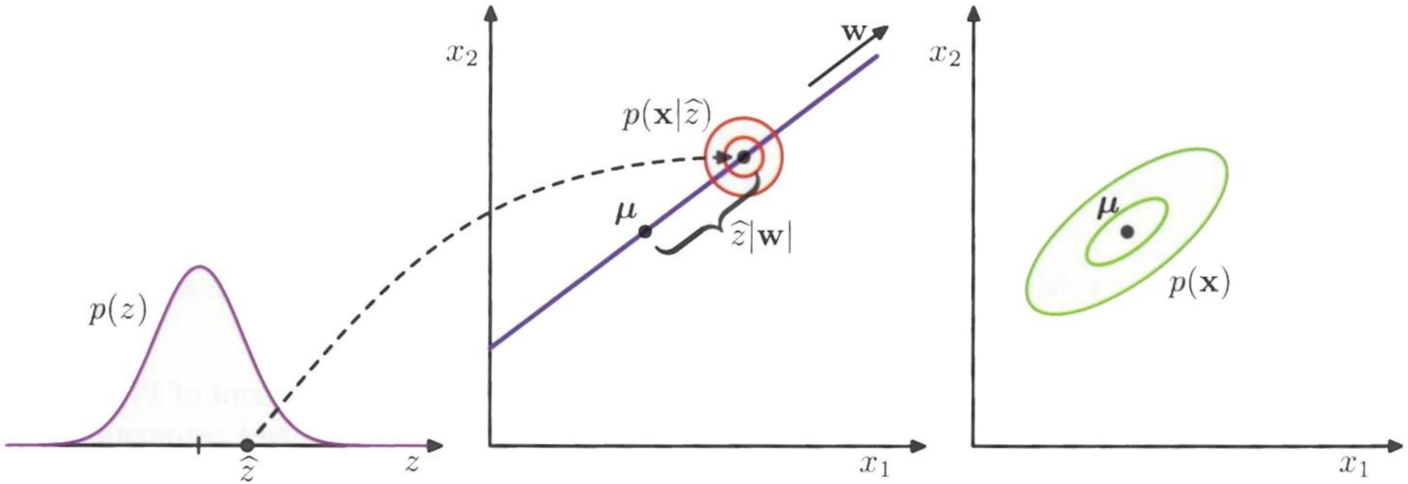
- Denoising data
- Data compression/encryption
- Classification
  - Reduced dimension leads to a unique manifold
- Mix source separation
  - Multiple sources with unique manifolds linearly mixed together
  - Encoder: Source separation
  - Decoder: Generate sources

Generative Models

- Discriminative learning: classification
  - Supervised process
  - Find difference
  - E.g.: MNIST classify digit as 0,1,…,9
- Generative learning: creation
  - Unsupervised learning
  - Find similarity
  - E.g.: machine translation

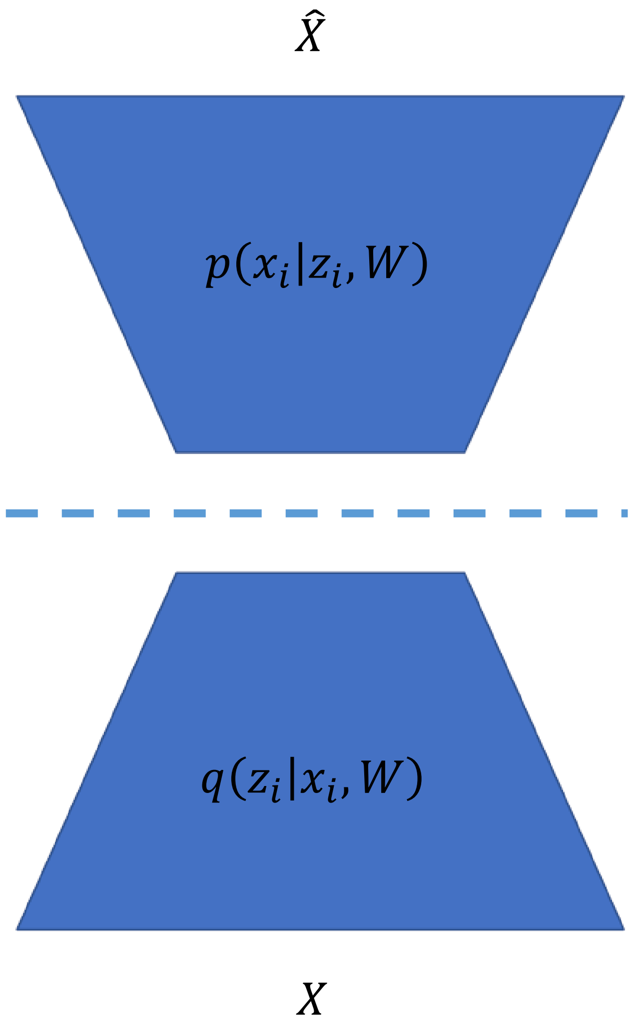
Latent Space Generative Models

- Data are generated from a real-valued latent space
- Latent space: unknown model space of given data
  - Model is probabilistic (PDF/PMF, μ, Σ)
- Data: samples from that space are used to create
- Factor Analysis:
  - Assume a model PDF/PMF
  - Objective: based on given data observation, $\mathbf{x}$, determine statistics (μ, Σ) and $p(\mathbf{x})$



Variational AE

- Decoder: $p(x_i|z_i, W)$
- Encoder: $q(z_i|x_i, W)$



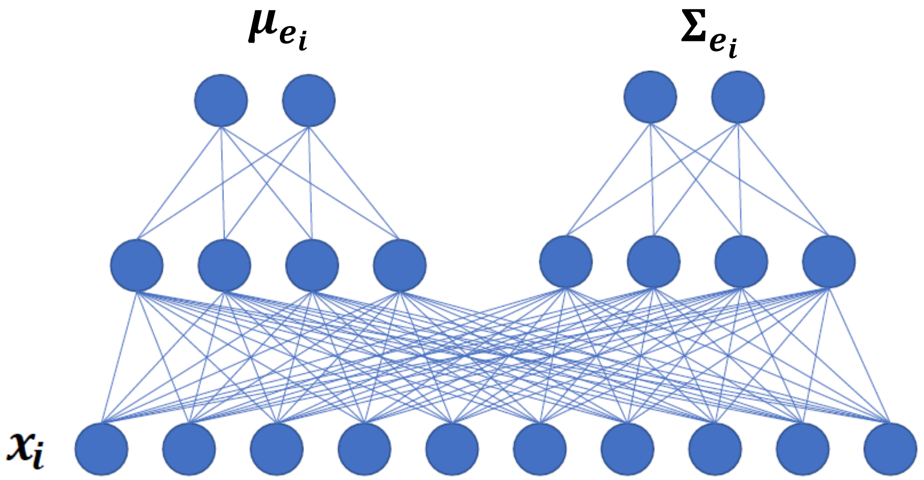
- Nnet, Generative model
- Estimates the probability distribution of input X given the laten variable Z

- Normally distributed Latent Space
- characterized by μ, σ

- Nnet, Inference model
- Estimates the probability distribution of the latent space given the data X

Encoder

- $q(z_i|x_i, \phi) = \mathcal{N}\left(z_i|\mu_{e_i}, \Sigma_{e_i}\right)$
- Encoder output is :$\mu_{e_i} = u_e(x_i, W_1)$, $\Sigma_{e_i} = \text{diag}\left(s_e(x_i, W_2)\right)$
- Two networks: u_e, s_e
- W_1: weights of network u_e
- W_2: weights of network s_e
- ϕ: combination of weights W_1, W_2



Decoder

- $p(x_i|z_i, \theta) = \mathcal{N}\left(x_i|\mu_{e_i}, \Sigma_{e_i}\right)$
- Sample Z space: generate z_i based on μ_{e_i} and Σ_{e_i}
- Decoder output: $\widehat{x_i}$

KL Divergence

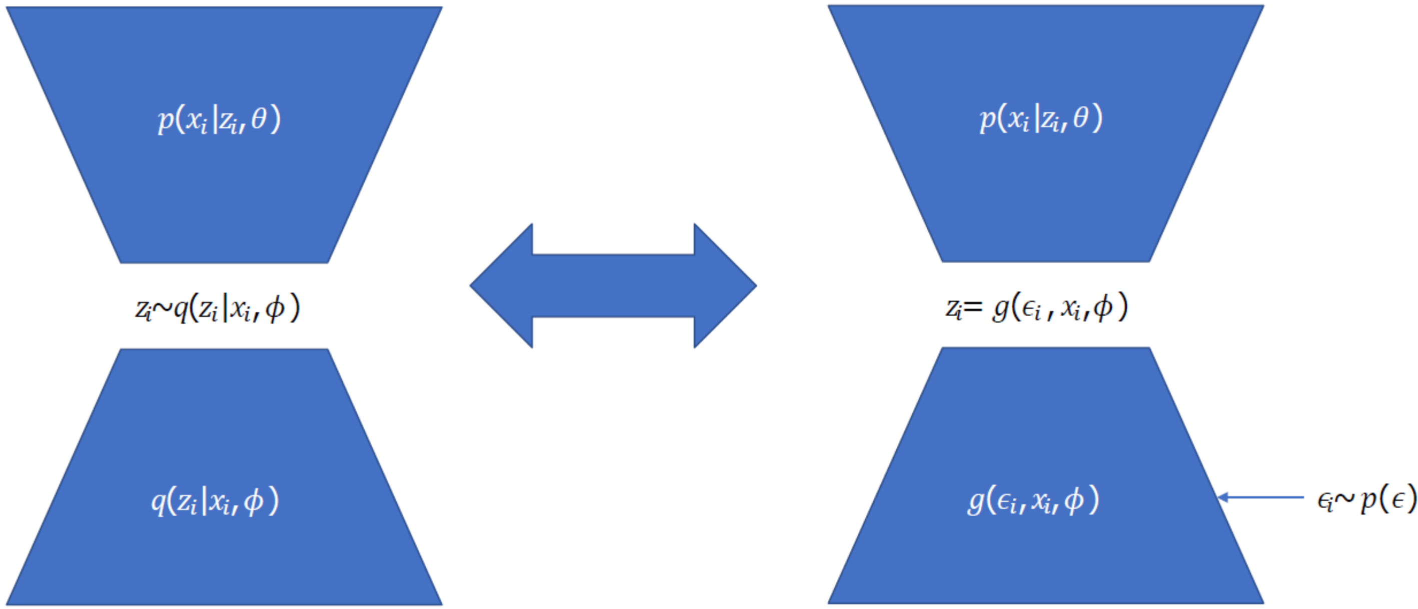
- Kullback-Leibler divergence
  - Measures the information lost when a probability distribution, q, is used to approximate another probability distribution p.

$$D_{KL}(p(x)||q(x)) = \sum_{x} p(x) \log \frac{p(x)}{q(x)}$$

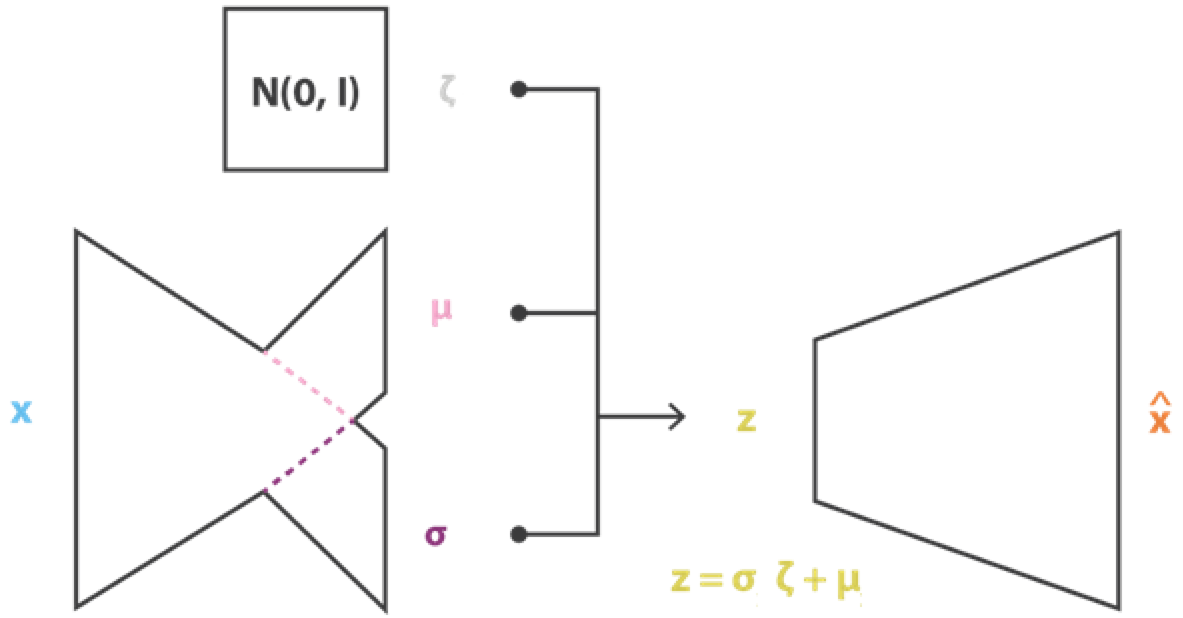
  - Weighted sum of similarity
  - Measures how much p differs from q

Reparametrization

- Let $z_i = g(\epsilon_i, x_i, \phi)$
- ϵ_i drawn from Gaussian $p(\epsilon)$
- Z deterministic depends on ϕ
- Now we can backpropagate!
- $z = \mu + \sigma \odot \epsilon$
- $\epsilon \sim \mathcal{N}(0,1)$



Re-parametrization



$$\text{loss} = C\,\|x - \hat{x}\|^2 + KL[\,N(\mu_x, \sigma_x), N(0, I)\,]$$

Edited from https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

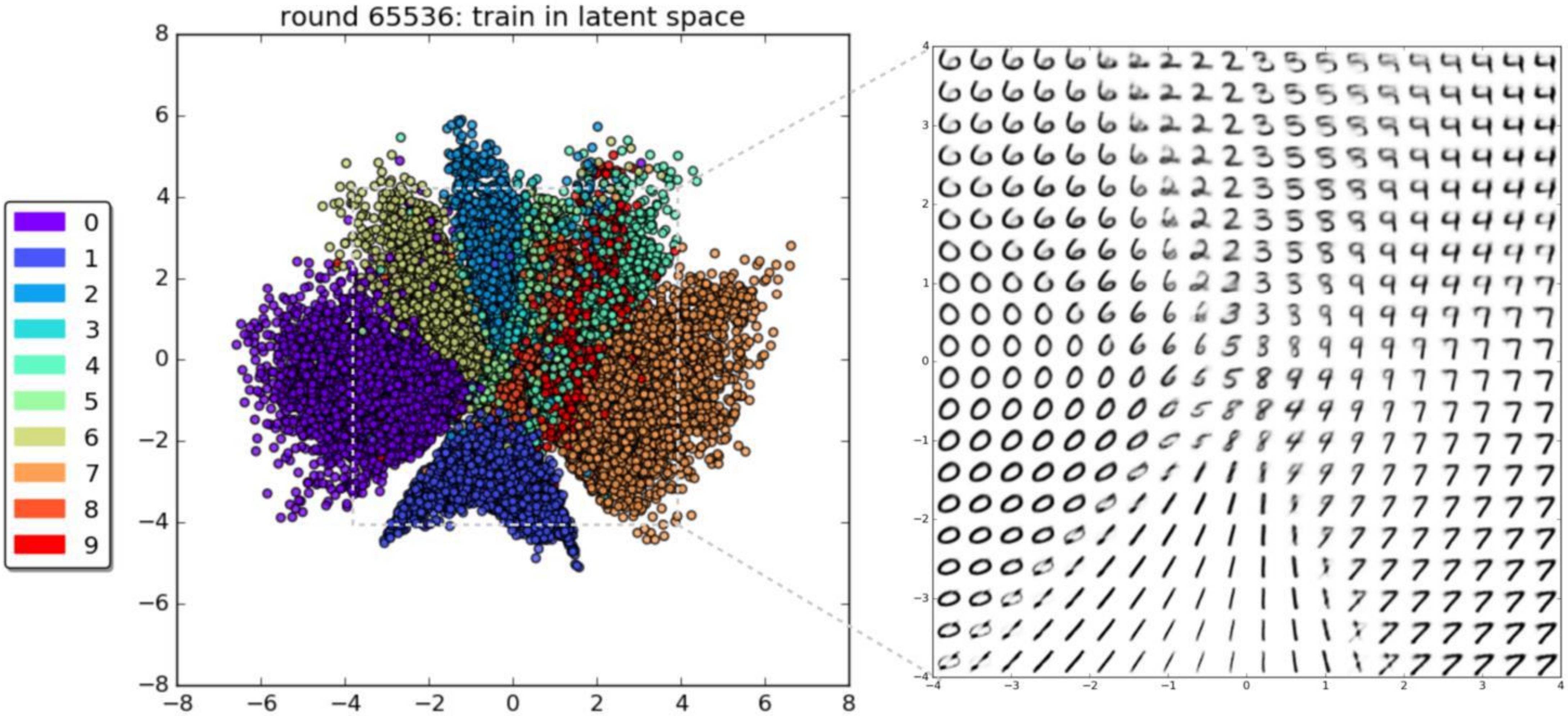
VAE KL loss

- Measure how different our normal distribution, $p \sim N(\mu, \sigma)$, differs from $N(0,1)$

$$D_{KL}\big(N(\mu,\sigma)||N(0,1)\big) = \frac{1}{2}\sum 1 + \log\sigma^2 - \mu^2 - \sigma^2$$

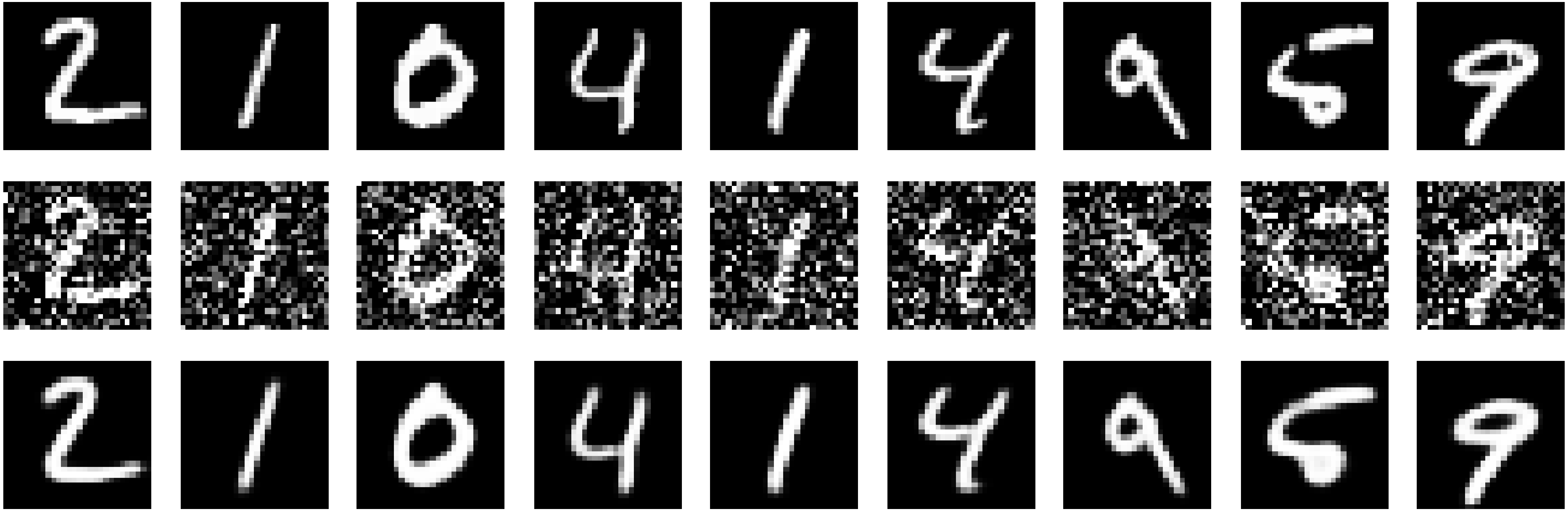
  - $\log\sigma^2$ aka $\log var$
  - $\sigma^2 = \exp(\log var)$
  - Sum of dimensions of latent space
  - KL loss is min when $\mu = 0, \sigma = 1$
    - Other statistics increase cost
    - Forces clusters of similar points to be very close to each other
    - Efficient use of space around origin

Manifold Learning



https://blog.keras.io/building-autoencoders-in-keras.html

Denoising

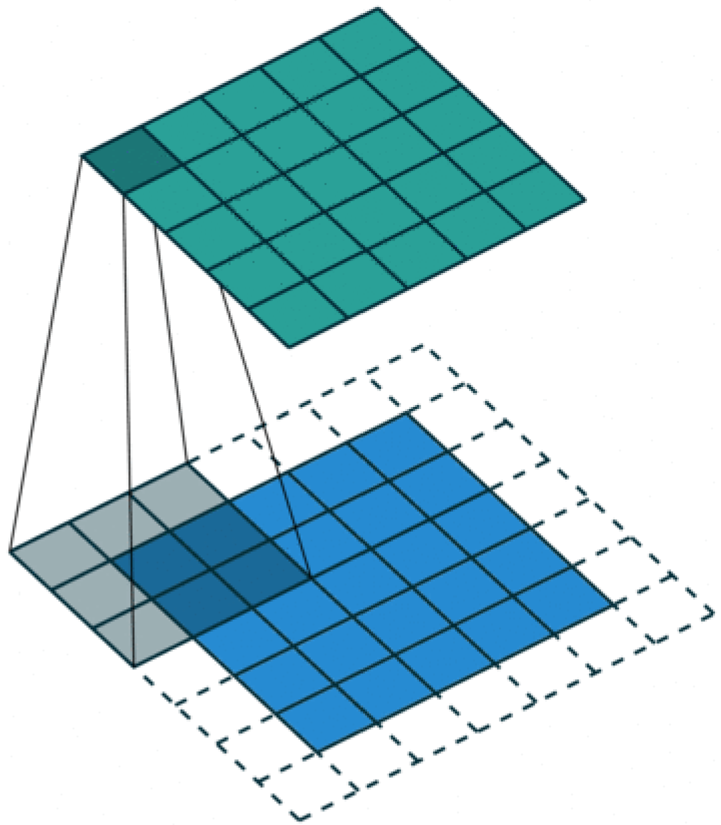


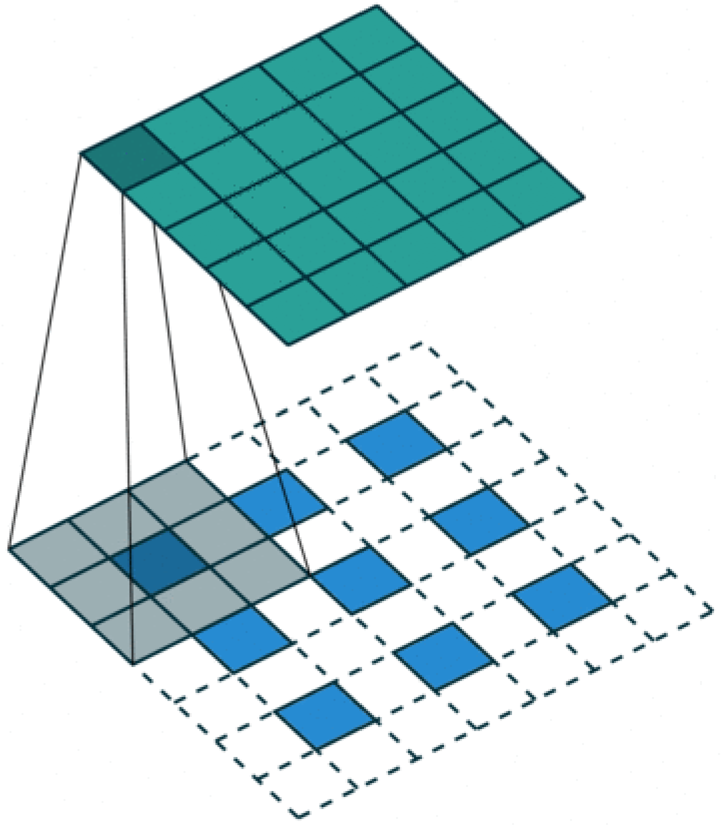
https://blog.keras.io/building-autoencoders-in-keras.html

Encoder-latent z-Decoder

- **Encoder:**
  - Can be a “dense” network
  - Can be CNN
- **Latent z:**
  - 2 dense networks to generate μ and $\log var$
  - Generated using reparameterization trick: $z = \mu + \sigma \odot \epsilon$
- **Decoder:**
  - Flip/mirror of encoder
  - Transpose convolution, unmax pooling can be used to mirror encoder

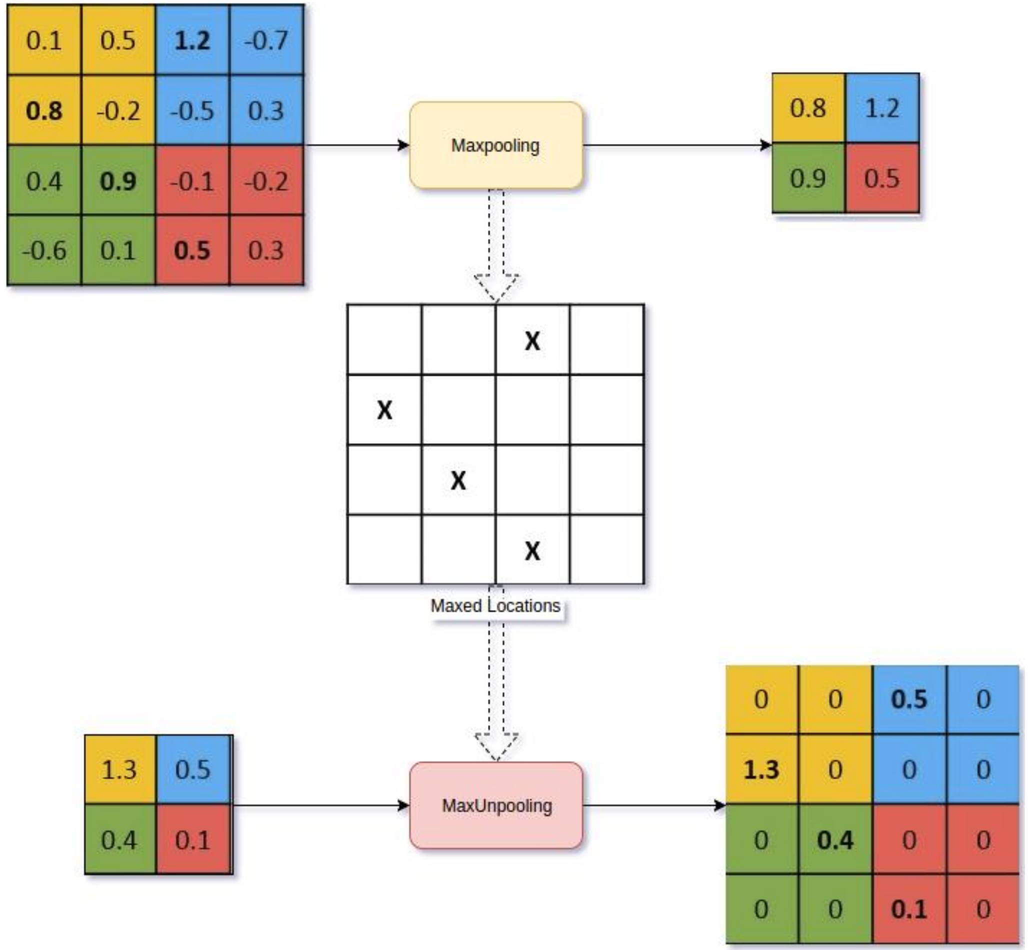
Convolution vs Transposed Convolution





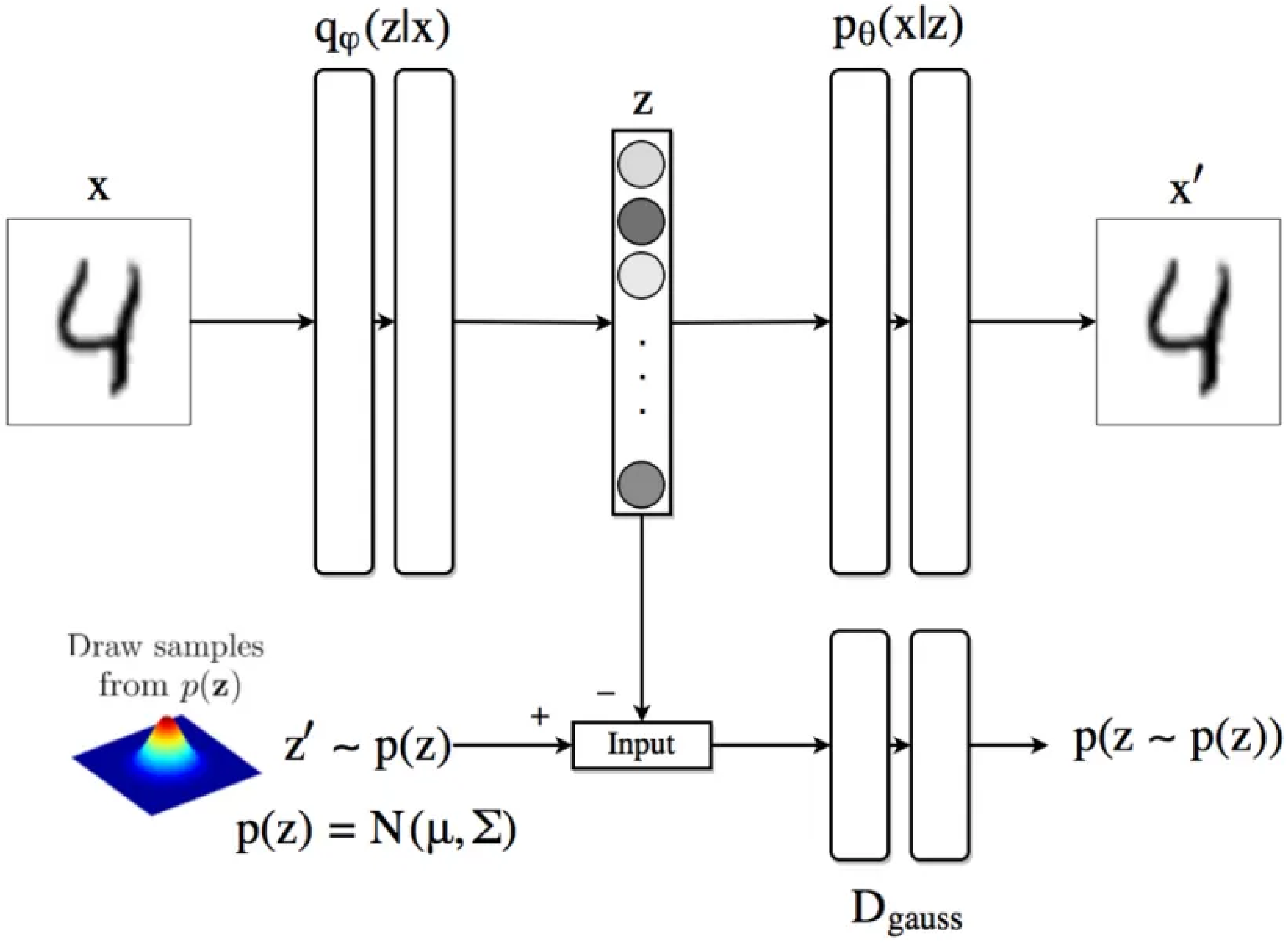
https://github.com/vdumoulin/conv_arithmetic

Max Pooling vs Unpooling



https://www.oreilly.com/library/view/hands-on-convolutional-neural/9781789130331/6476c4d5-19f2-455f-8590-c6f99504b7a5.xhtml

Adversarial Autoecoder (AAE)



AAE

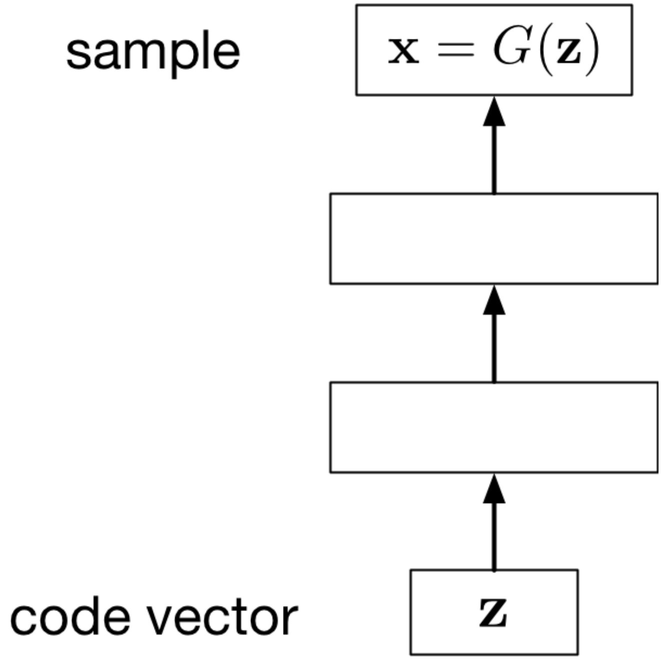
- uses adversarial loss to regularize the latent code instead of the KL-divergence
- 3 components:
  - Encoder, Decoder, Discriminator
- Encoder options:
  - Deterministic (AE)
  - Gaussian Posterior (VAE)
  - Universal Approximator Posterior
- Discriminator:
  - Input: random vector z sampled from the chosen distribution (real)
  - Input: latent code z (fake) from the encoder
  - Determines if input is real or fake.

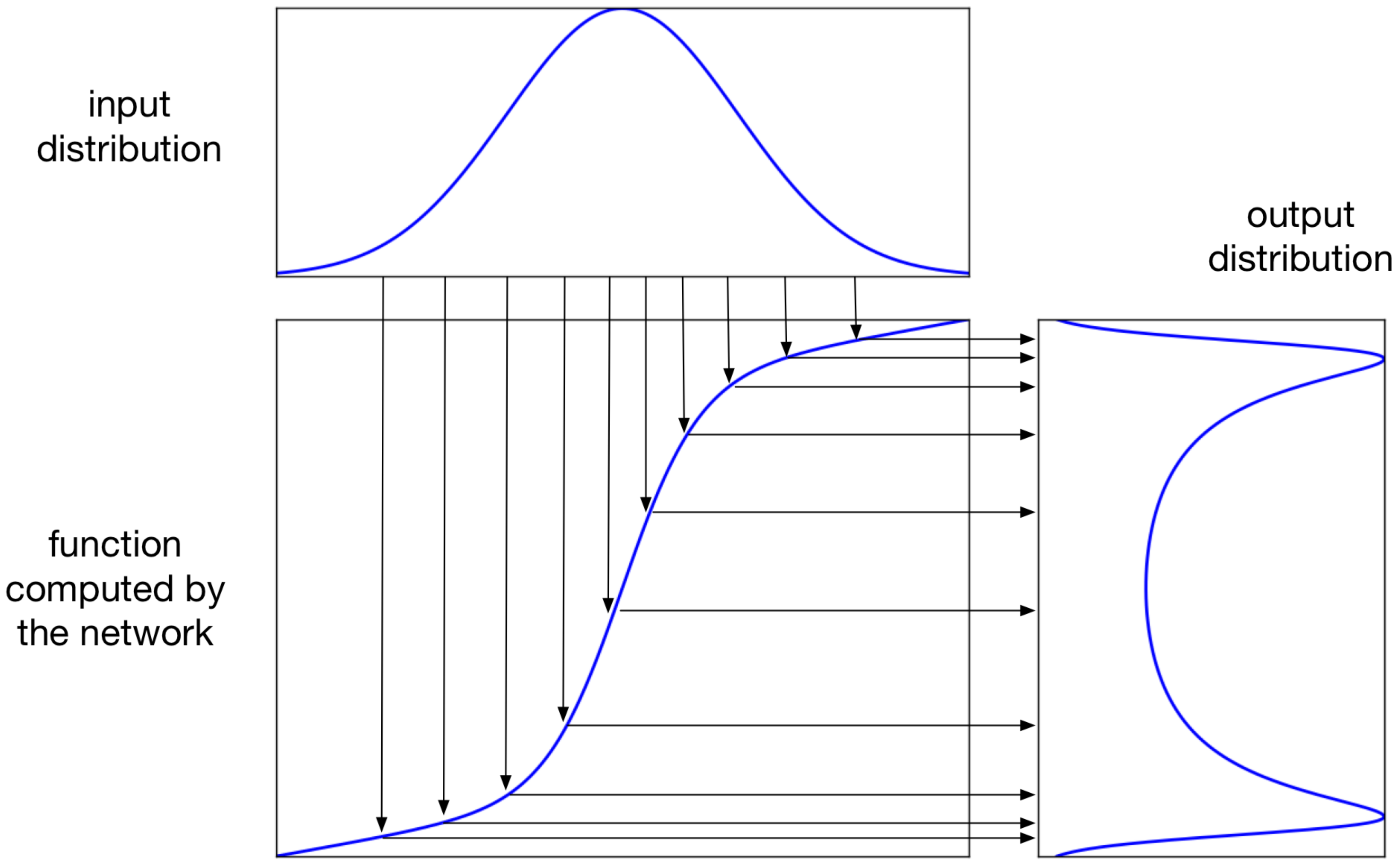
AAE Application

- Anomaly/intrusion/fake detection
  - Label deficient problem
    - Unsupervised learning
  - Application 1: Data is encoded and compared to non-anomalous encodings
    - Distance measure to encoding mean
  - Application 2: Data is reconstructed and compared to original
    - Difference in data is considered anomalous
- Denoising/Super-resolution/Style transfer
  - Generation can used to learn

GANs: Density Network

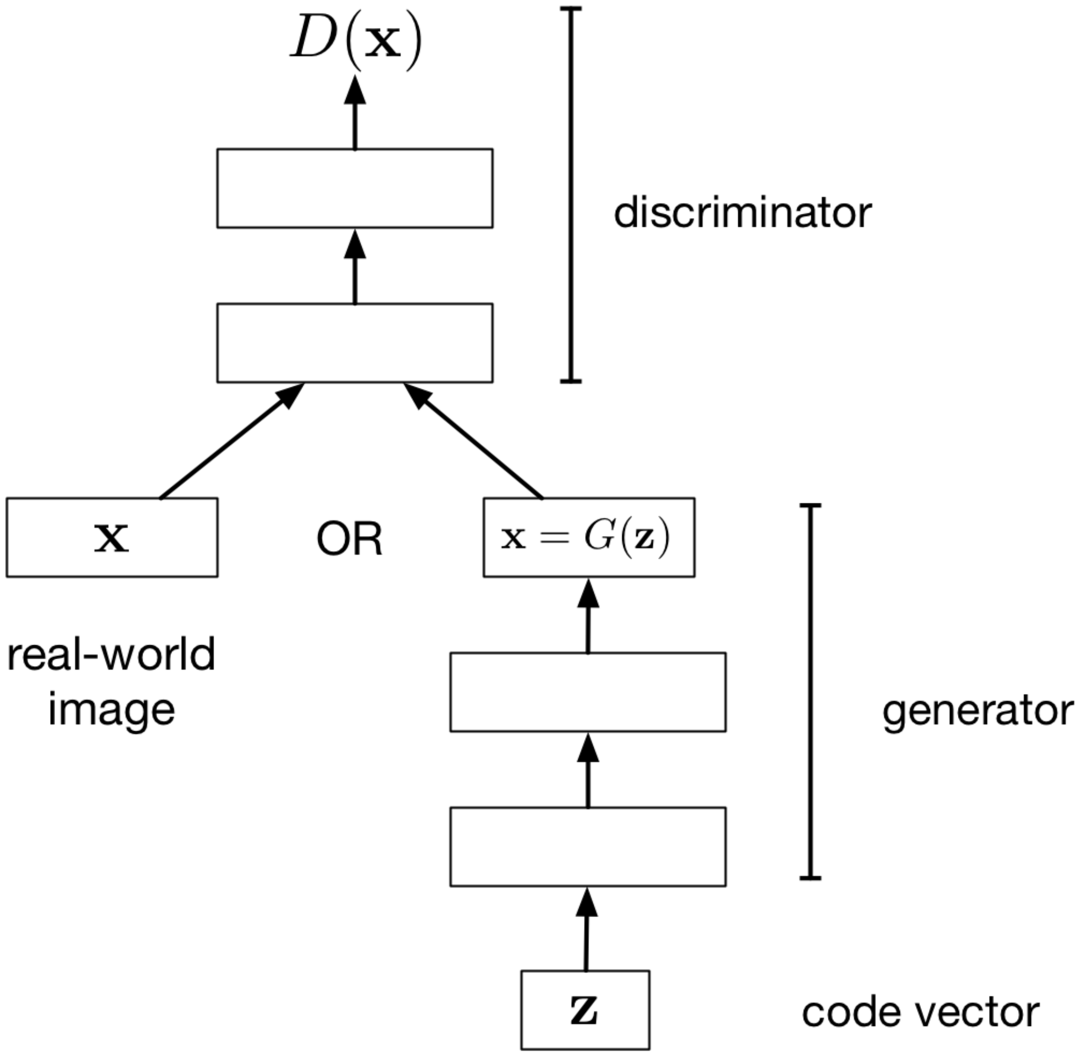
- x data sample
- z latent sample
  - simple distribution (Gaussian)
- Goal: compute function G
  - differentiable
  - maps z to an x in data space




input
distribution
output
distribution
function
computed by
the network

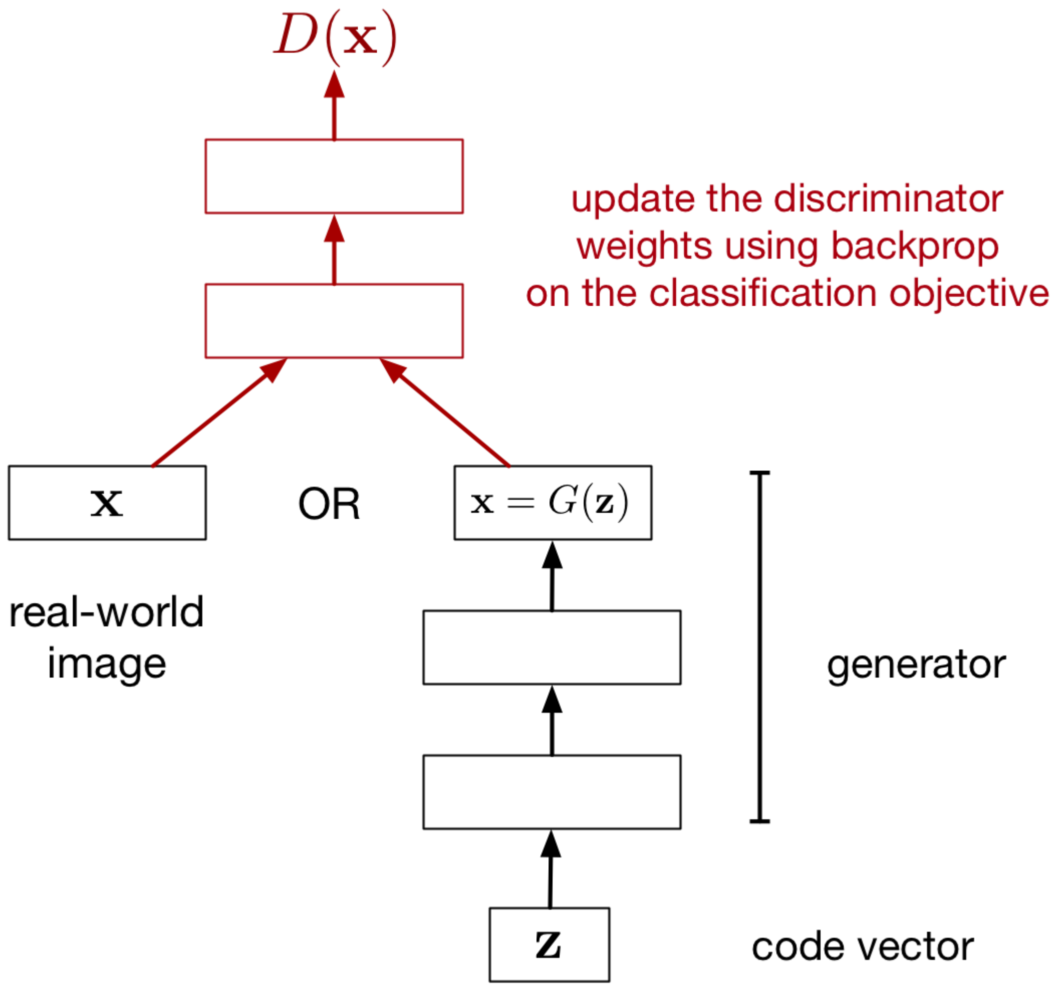
GAN

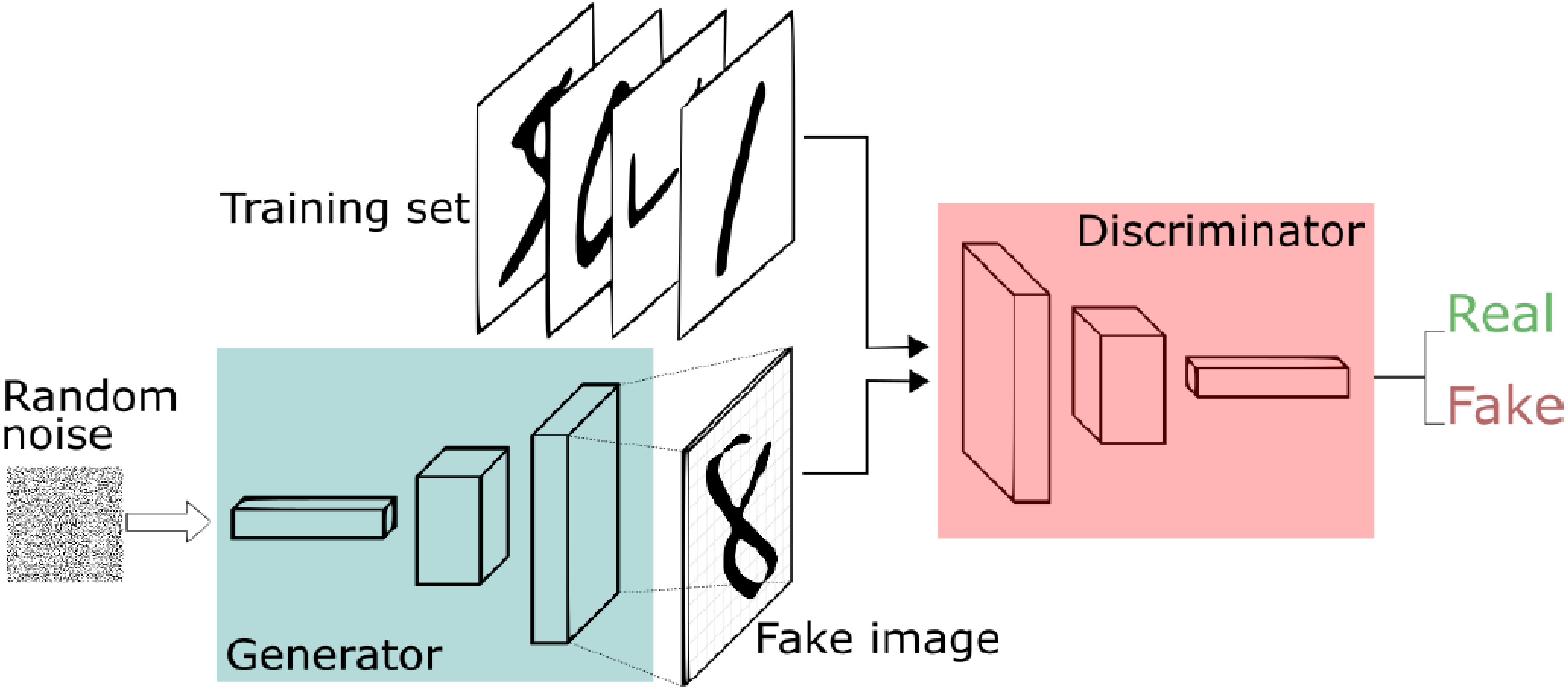
- Generative Adversarial Networks (GANs)
- Composed of two different networks
  - Generator network (G): a density network produces realistic-looking samples
  - Discriminator network (D): determines if input came from the training set or the generator network
- $G(z)$ is a data vector
  - Same dimension as x
- $D(x)$ is 0 or 1
  - x is fake or real
- Goal: G must fool D
  - z will be a good model of x



Learning $D(x)$

- Feed foward an input, x, or generated input $G(z)$.
- Calculate loss, backprop

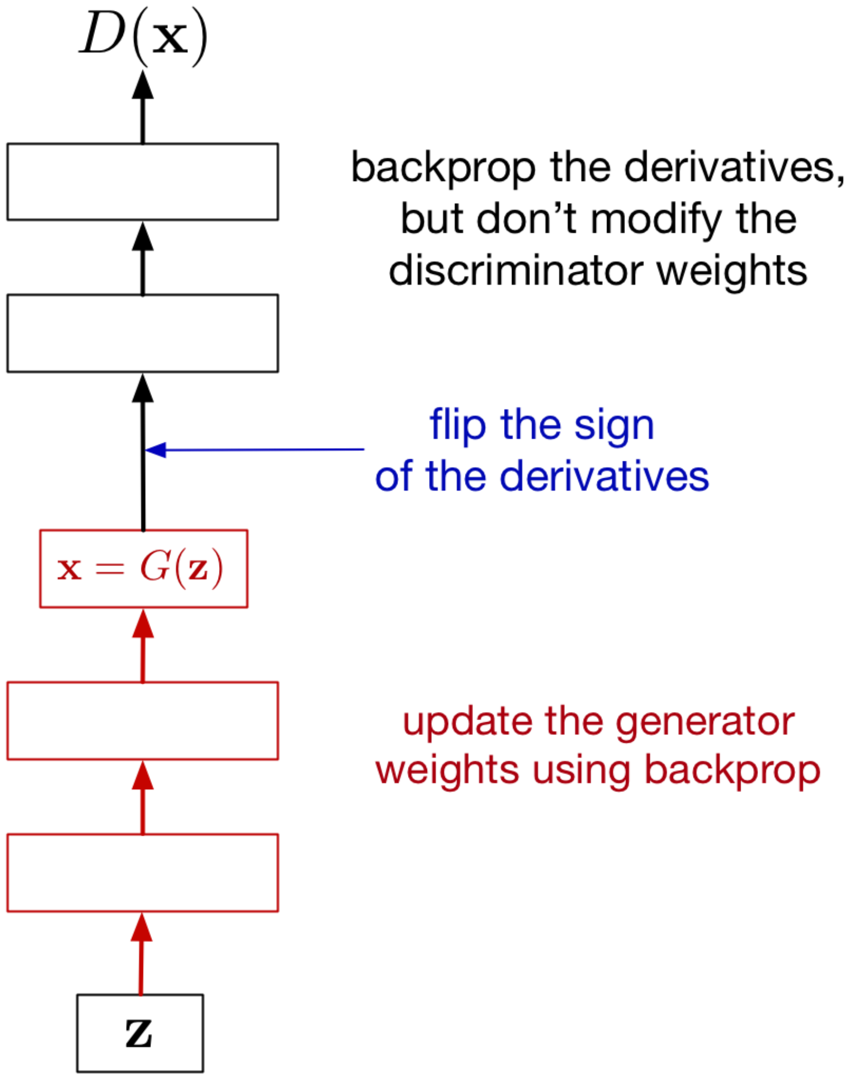




Training set
Random
noise
Generator
Fake image
Discriminator
Real
Fake

Learning $G(z)$

- G and D playing zero-sum game
  - D minimizes the probability of wrong guess
  - G maximizes the probability of wrong guess
- G objective function is the opposite of D
  - Minmax formulation



Saturation

- Square error causes saturation: $\left(1 - D\left(G(z)\right)\right)^2$
- Instead use cross-entropy: $-\log\left(D\left(G(z)\right)\right)$

Applications: Image to Image Translations

Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for

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

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.