Autoencoders

Vahid Tarokh

ECE 685D, Fall 2025

Unsupervised Learning

- Unsupervised learning: we only use the inputs $\mathbf{x}^{(t)}$ for learning
  - automatically extract meaningful features for your data
  - leverage the availability of unlabeled data
  - add a data-dependent regularizer to training (in some cases)

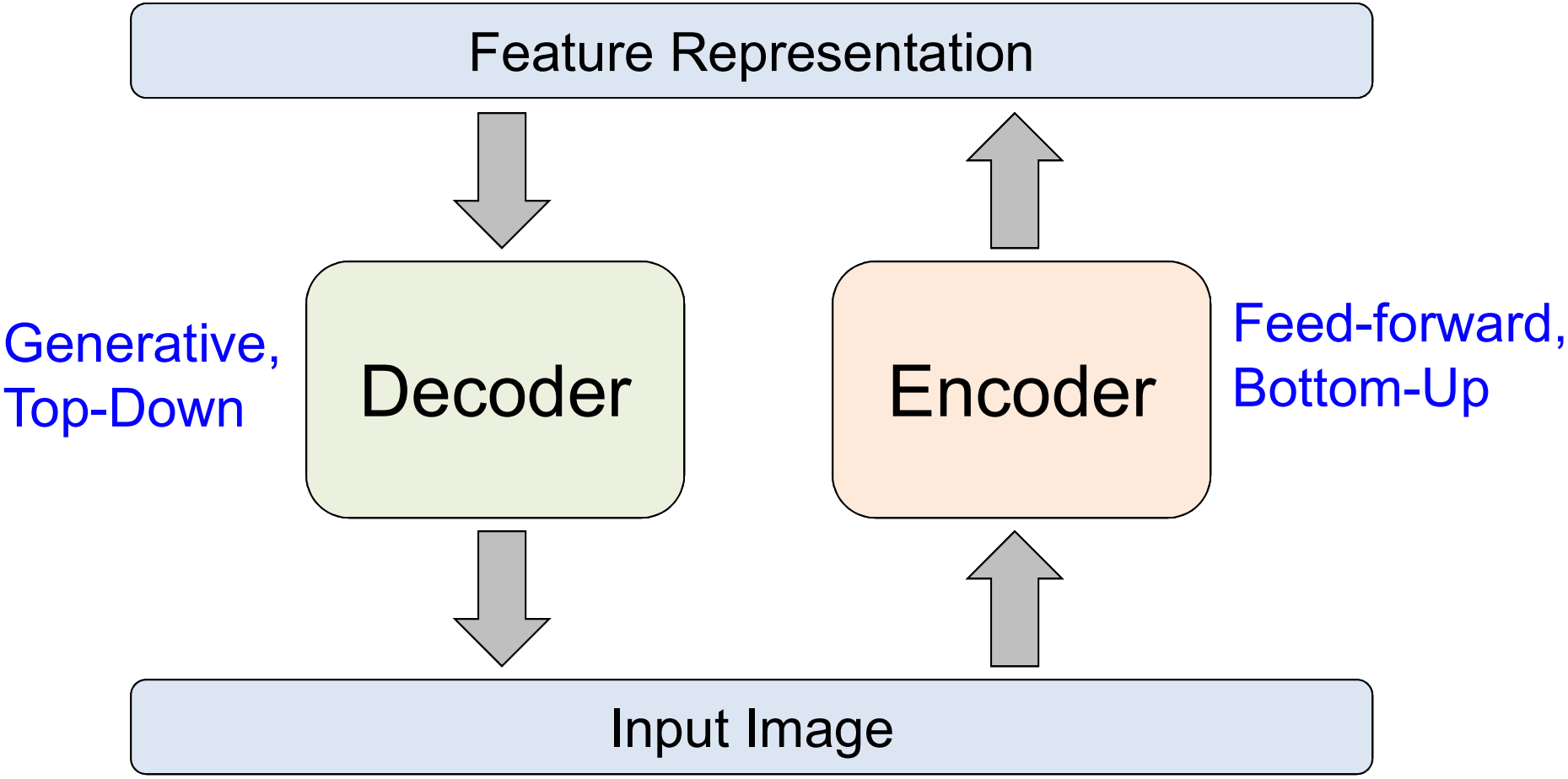
- We will consider 3 models for unsupervised learning that will form the basic building blocks for deeper models:
  - **Autoencoders**
  - Sparse Coding
  - Restricted Boltzmann Machines

Autoencoders

- Map high-dimensional data to lower dimensions
  - Visualization
  - Compression (reducing the file size)
- Learn abstract features in an unsupervised way for downstream supervised tasks
  - Unlabeled data usually are much more plentiful than labeled data
- Build some generative models

Autoencoders

An autoencoder is a feed-forward neural net whose job is to take an input x and output $\hat{x}$



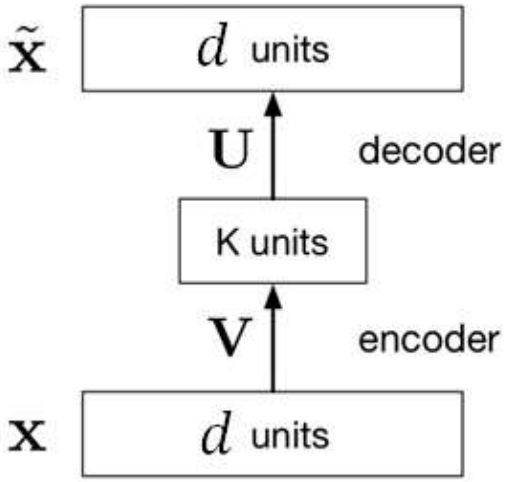
- Details of what goes insider the encoder and decoder matter!

Totally Linear Case: PCA

- The simplest kind of autoencoder has one hidden layer, linear activations, and squared error loss.

$$\mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$$

- This network computes $\tilde{\mathbf{x}} = \mathbf{UVx}$, which is a linear function.
- If $K \geq d$, we can choose $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{UV}$ is the identity. This isn't very interesting.



If $k < d$, then we have dimensionality reduction.

A linear autoencoder with MSE loss, one hidden layer, and tied weights learns the exact same subspace as PCA. Autoencoders are a nonlinear generalization of PCA.

PCA

- Dimensionality reduction technique for data in $\mathbb{R}^d$
- Problem statement
  - given m points $x_1, \ldots, x_m$ in $\mathbb{R}^d$
  - and target dimension $k < d$
  - find "best" k-dimensional subspace approximating the data
- Formally: find matrices $U \in \mathbb{R}^{d \times k}$ and $V \in \mathbb{R}^{k \times d}$
- that minimize

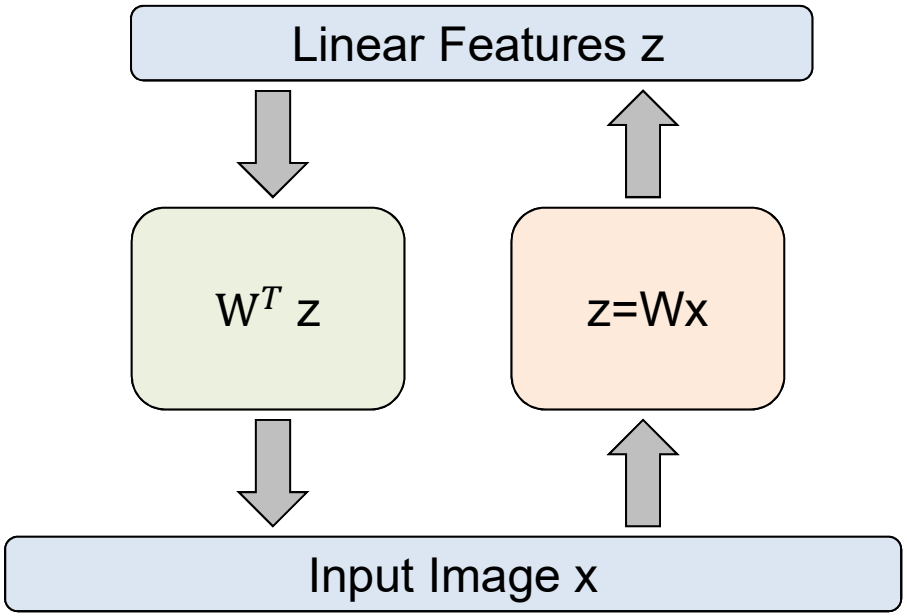
$$f(U, V) = \sum_{i=1}^{m} \|x_i - UVx_i\|_2^2$$

- $V : \mathbb{R}^d \to \mathbb{R}^k$ is "compressor", $U : \mathbb{R}^k \to \mathbb{R}^d$ is "decompressor"
- Is $f : \mathbb{R}^{d \times k} \times \mathbb{R}^{k \times d} \to \mathbb{R}$ convex?
- No! $f(U, \cdot)$ and $f(\cdot, V)$ both convex, but not $f(\cdot, \cdot)$
- **Claim:** optimal solution achieved at $U = V^\top$ and $U^\top U = I$ (columns of U are orthonormal)

PCA

- Optimization problem: minimize$_{U \in \mathbb{R}^{d \times k}, V \in \mathbb{R}^{k \times d}} \sum_{i=1}^{m} \|x_i - UVx_i\|_2^2$
- **Claim:** optimal solution achieved at $U = V^\top$ and $U^\top U = I$
- Proof:
  - For any U, V, linear map $x \mapsto UVx$ has range R of dimension k
  - Let $w_1, \ldots, w_k$ be orthonormal basis for R; arrange into columns of W.
  - Hence, for each x_i there is $z_i \in \mathbb{R}^k$ such that $UVx_i = Wz_i$.
  - Note: $W^\top W = I$.
  - Which z minimizes $f(x_i, z) := \|x_i - Wz\|_2^2$?
  - For all $x \in \mathbb{R}^d, z \in \mathbb{R}^k$,
    $f(x, z) = \|x\|_2^2 + z^\top W^\top Wz - 2z^\top W^\top x = \|x\|_2^2 + \|z\|_2^2 - 2z^\top W^\top x$.
  - Minimize w.r.t. z: $\nabla_z f = 2z - 2W^\top x = 0 \implies z = W^\top x$.
  - Therefore
    $$\sum_{i=1}^{m} \|x_i - UVx_i\|_2^2 = \sum_{i=1}^{m} \|x_i - Wz_i\|_2^2 \geq \sum_{i=1}^{m} \|x_i - WW^\top x_i\|_2^2.$$
  - U, V are optimal, so $\sum_{i=1}^{m} \|x_i - UVx_i\|_2^2 = \sum_{i=1}^{m} \|x_i - WW^\top x_i\|_2^2$.
  - So instead of U, V can take $W, W^\top$. □
- $WW^\top x$ is the **orthogonal projection** of x onto R.

Linear Autoencoder (PCA)



- If the hidden and output layers are linear, it will learn hidden units that are a linear function of the data and minimize the squared error.
- The K hidden units will span the same space as the first k principal components. The weight vectors may not be orthogonal.
- If $K < d$, and W with orthogonal $rows$, then we have PCA.

With nonlinear hidden units, we have a **nonlinear generalization of PCA.**

Optimality of the Linear Autoencoder

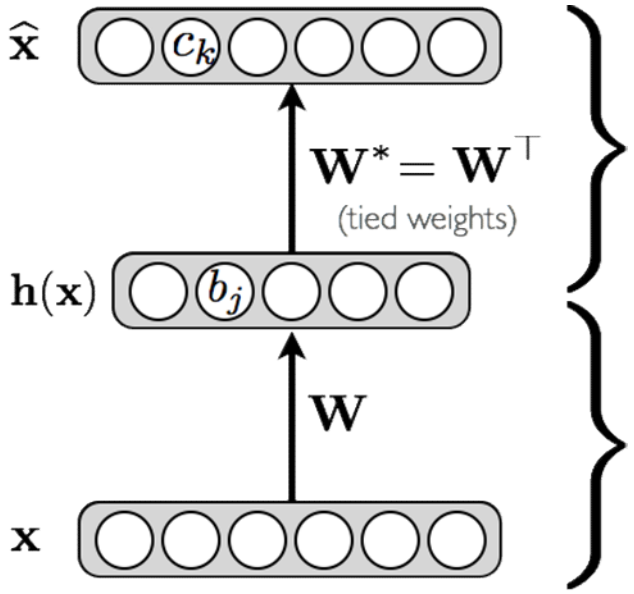
- Let us consider the following theorem:

- let $\mathbf{A}$ be the empirical covariance of data points
- Let $\mathbf{A} = \mathbf{U}\,\Sigma\,\mathbf{U}^\top$
- singular value decomposition
  - Σ is a diagonal matrix
  - $\mathbf{U}$ are orthonormal matrices (columns/rows are orthonormal vectors)

Linear Autoencoder (PCA)

- The optimal pair of encoder and decoder is given by:

$$\mathbf{h}(\mathbf{x}) = \underbrace{(\mathbf{U}_{\cdot,\leq k})^\top}_{\mathbf{W}}\mathbf{x} \qquad \hat{\mathbf{x}} = \underbrace{\mathbf{U}_{\cdot,\leq k}}_{\mathbf{W}^*}\mathbf{h}(\mathbf{x})$$

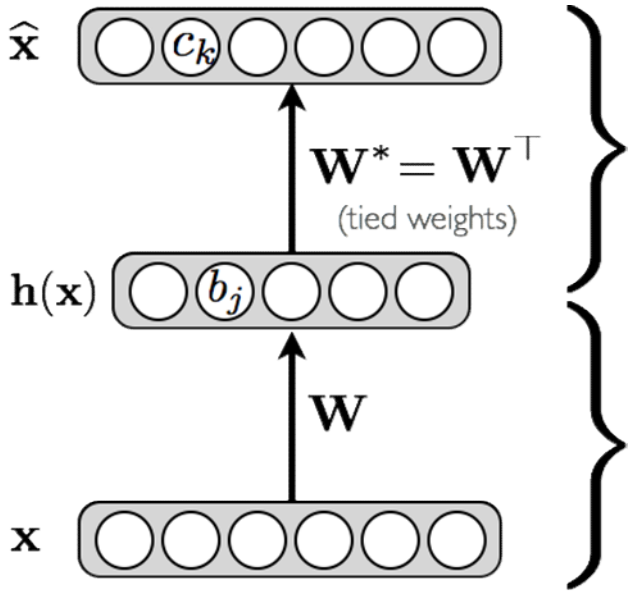


- for the sum of squared difference error
- for an auto-encoder with a linear decoder
- where optimality means "has the lowest training reconstruction error"

Optimality of the Linear Autoencoder

- So an optimal pair of encoder and decoder is

$$\mathbf{h}(\mathbf{x}) = \underbrace{(\mathbf{U}_{\cdot,\leq k})^\top}_{\mathbf{W}}\mathbf{x} \qquad \hat{\mathbf{x}} = \underbrace{\mathbf{U}_{\cdot,\leq k}}_{\mathbf{W}^*}\mathbf{h}(\mathbf{x})$$



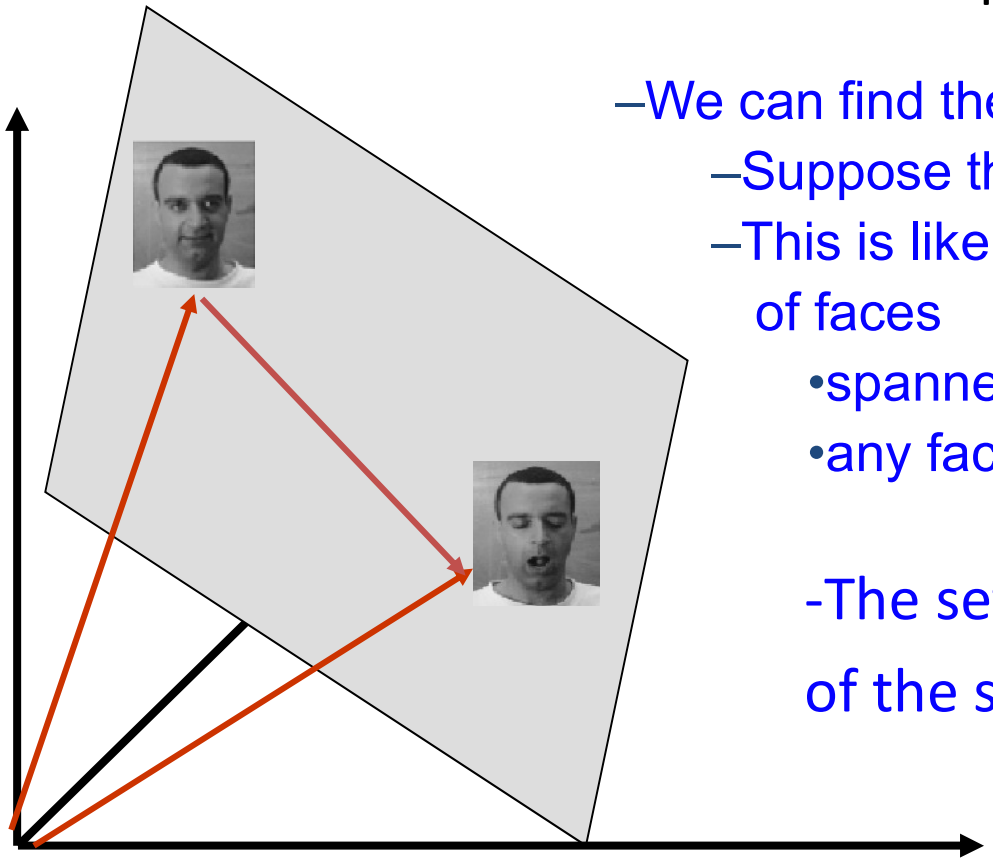
- If inputs are normalized as follows:

$$\mathbf{x}^{(t)} \leftarrow \frac{1}{\sqrt{T}}\left(\mathbf{x}^{(t)} - \frac{1}{T}\sum_{t'=1}^{T}\mathbf{x}^{(t')}\right)$$

  - encoder corresponds to Principal Component Analysis (PCA)
  - singular values and (left) vectors = the eigenvalues/vectors of covariance matrix

Example: Applications of PCA

- **Face Recognition**
- An image is a point in a high-dimensional space
  - An $N \; x \; M$ image is a point in R^{NM}
  - We can define vectors in this space



- We can find the best subspace using PCA
  - Suppose this is a K-dimensional subspace
  - This is like fitting a "hyper-plane" to the set of faces
    - spanned by vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_K}$
    - any face $\mathbf{x} \sim a_1\mathbf{v_1} + a_2\mathbf{v_2} + , \ldots, + a_K\mathbf{v_K}$

-The set of faces is a "subspace" of the set of images.

Application of PCA

- Let $F_1, F_2, \ldots, F_M$ be a set of training face images.
- Let F be their mean and $f_i = F_i$ – F
- Use principal components to compute the eigenvectors and eigenvalues of the covariance matrix of the $f_i's$
- Choose the vector u of most significant M eigenvectors to use as the basis.
- Each face (with F removed) is represented as a linear combination of eigenfaces $u = (u_1, u_2, u_3, u_4, u_5)$;
- $f_{27} = a_1 * u_1 + a_2 * u_2 + \ldots + a_5 * u_5$

unknown face image → convert to its eigenface representation → $A = (a_1, a_2, \ldots, a_m)$

Find the face class k that minimizes

$$||A - A_k||$$

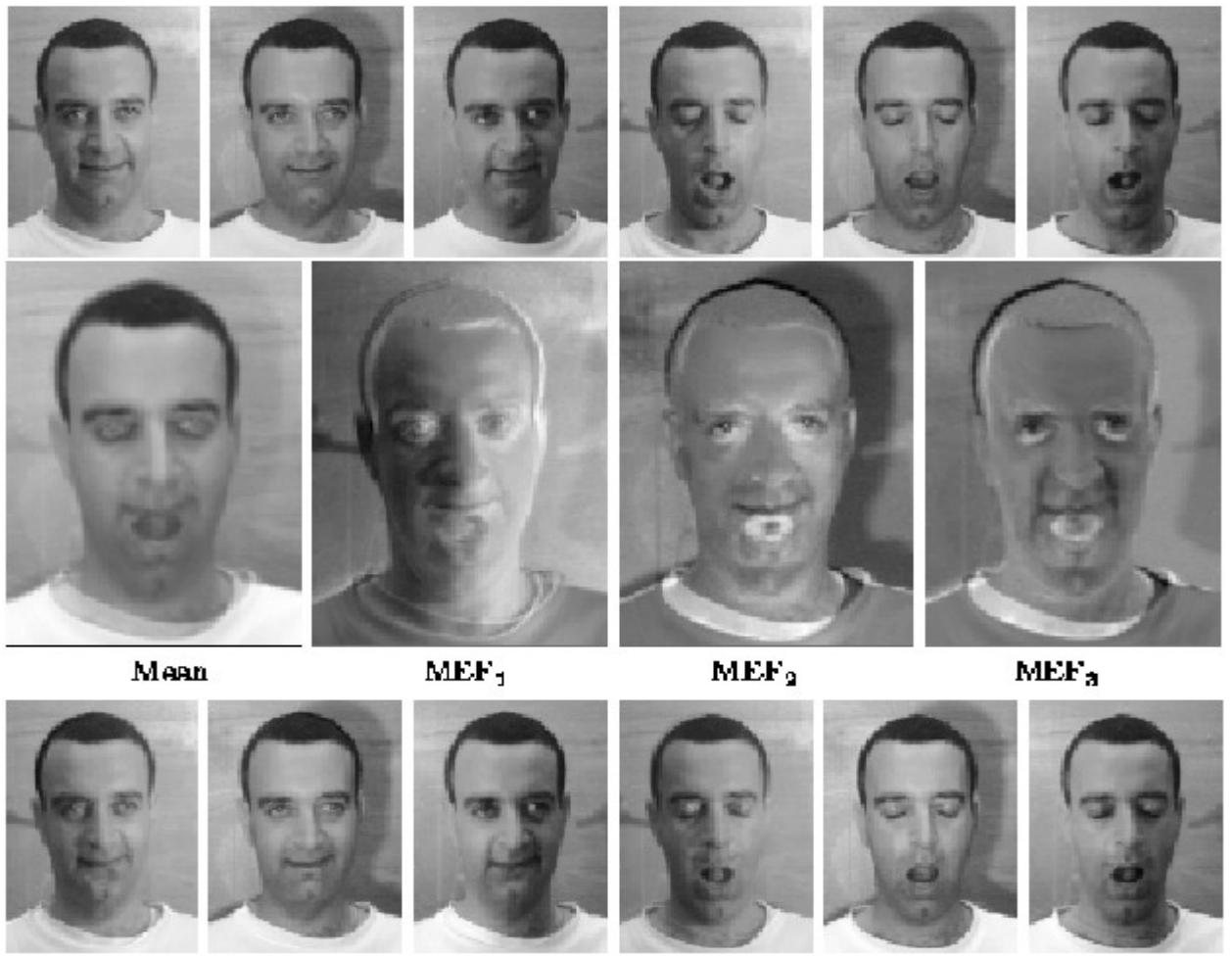
Application of PCA

training images

mean image

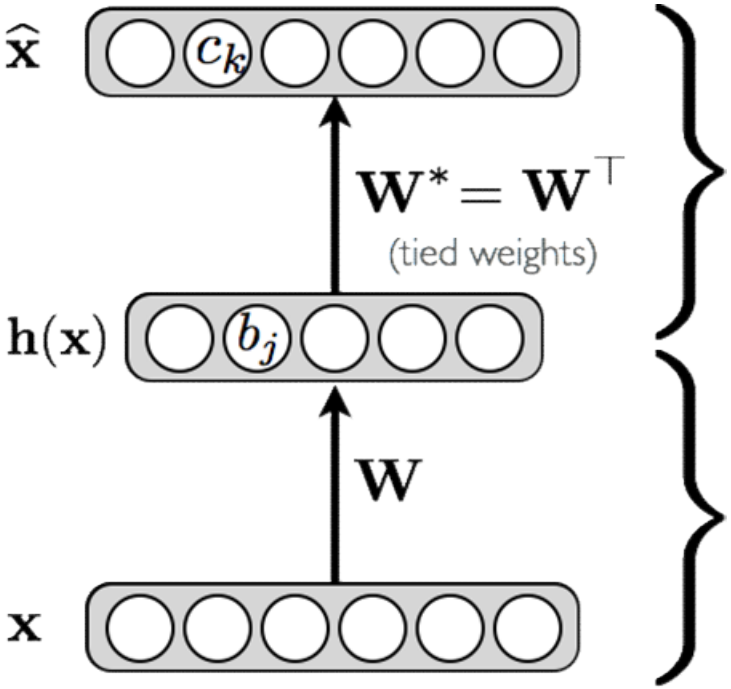
3 eigen-images

linear approxi-mations



Nonlinear Autoencoder (Example)

- Feed-forward neural network trained to reproduce its input at the output layer
- Nonlinear generalization of PCA



Decoder

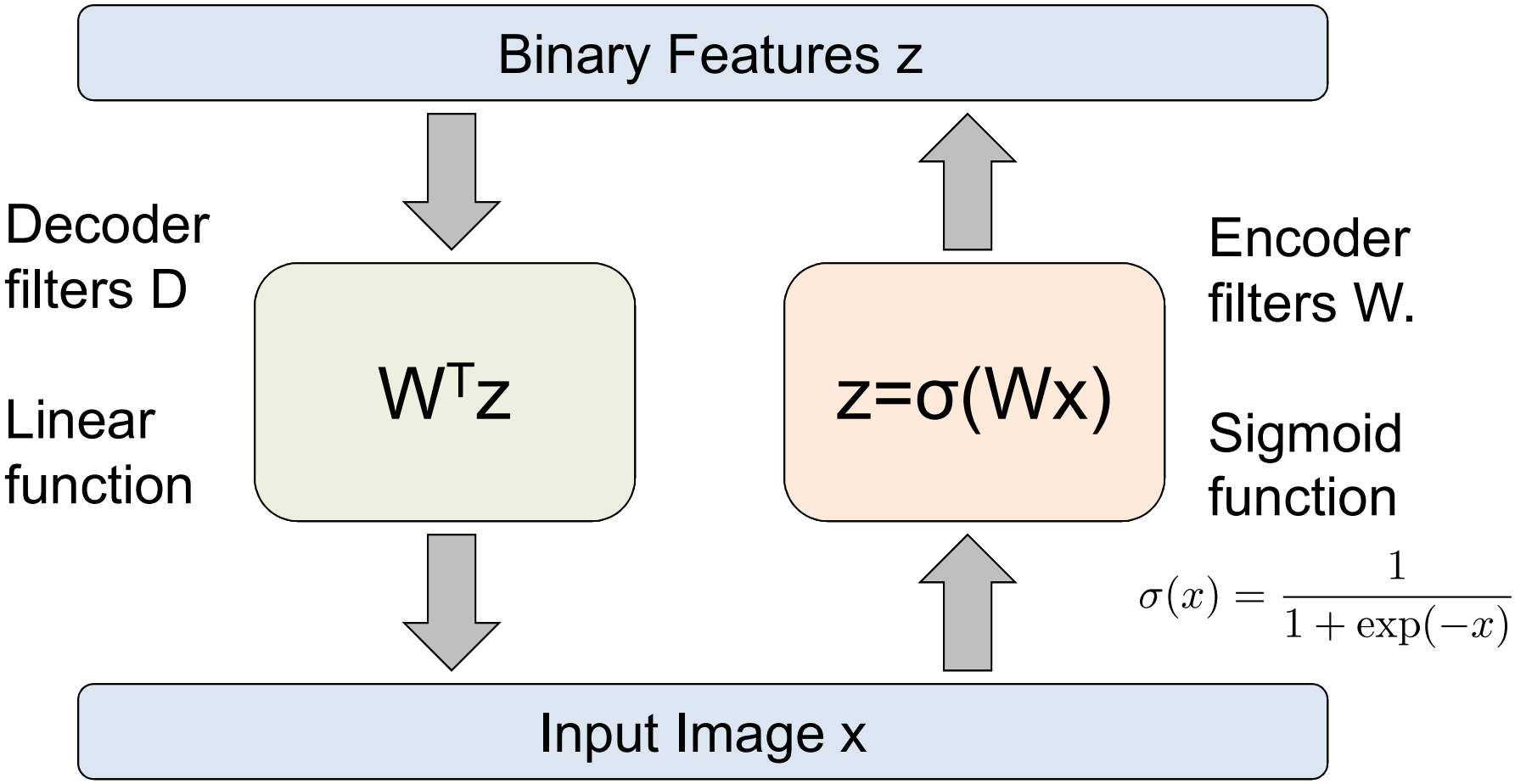
$$\begin{aligned}\hat{\mathbf{x}} &= \hat{\mathbf{a}}(\mathbf{x}) \\ &= \underbrace{\mathrm{sigm}(\mathbf{c} + \mathbf{W}^* \mathbf{h}(\mathbf{x}))}_{\text{For binary units}}\end{aligned}$$

Encoder

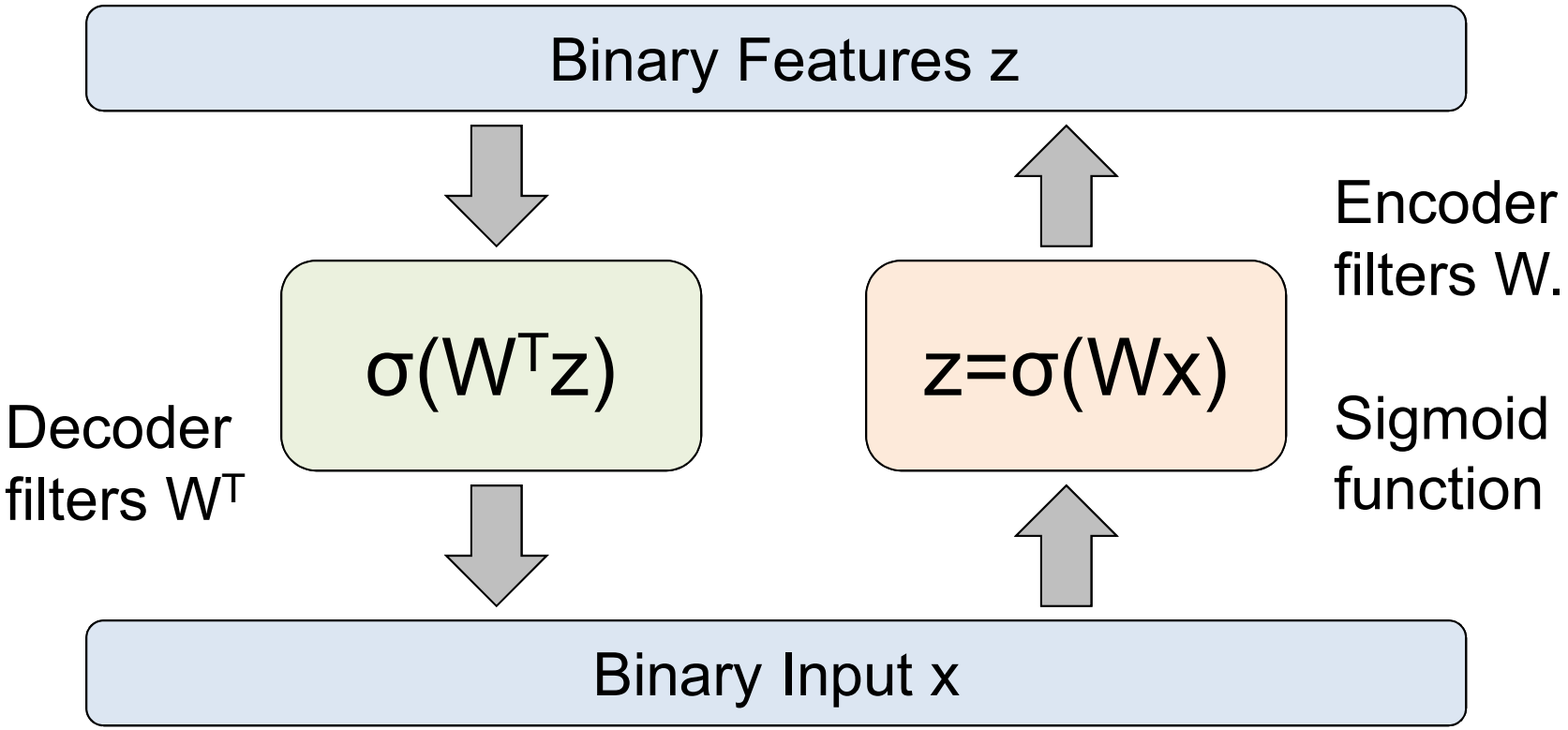
$$\begin{aligned}\mathbf{h}(\mathbf{x}) &= g(\mathbf{a}(\mathbf{x})) \\ &= \mathrm{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})\end{aligned}$$

- In general it is hard to prove optimality results in nonlinear case.

Autoencoders (Example)



Autoencoder Example



- Need additional constraints to avoid learning an identity.
- In general, Encoder and Decoder filters can be different.

Loss Function (Examples)

- Reproduce (reconstruct) the input at the output layer
- $f(x)$ is the output of autoencoder (model prediction)

- Loss function for binary inputs

$$l(f(\mathbf{x})) = -\sum_k \left(x_k \log(\widehat{x}_k) + (1 - x_k) \log(1 - \widehat{x}_k)\right)$$

  - Cross-entropy error function (reconstruction loss) $f(\mathbf{x}) \equiv \widehat{\mathbf{x}}$

- Loss function for real-valued inputs

$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\widehat{x}_k - x_k)^2$$

  - sum of squared differences (reconstruction loss)
  - we use a linear activation function at the output

Loss Function

- Parameter gradients are obtained by back-propagating the gradient $\nabla_{\widehat{\mathbf{a}}(\mathbf{x}^{(t)})} l(f(\mathbf{x}^{(t)}))$ like in a regular network

  - Important: when using tied weights ($\mathbf{W}^* = \mathbf{W}^\top$), $\nabla_{\mathbf{W}} l(f(\mathbf{x}^{(t)}))$ is the sum of two gradients (Show this as an exercise).
  - this is because $\mathbf{W}$ is present in the encoder and in the decoder

Autoencoder example

Encoder architecture:

Convolutional layer

16 output channels , kernel size = 3, stride=3, padding=1

ReLU

MaxPooling

size = 2X2, stride=2

Convolutional layer

8 output channels, kernel size =3, stride=2, padding=1

ReLU

MaxPooling

size = 2, stride=1

Decoder architecture:

Convolutional Transpose layer

8 input channels , 16 output channels, kernel size=3, stride=2, padding=1

ReLU

Convolutional Transpose layer

16 input chennels, 8 output channels, kernel size =5, stride=3, padding=1

ReLU

Convolutional Transpose layer

8 input channels , 1 output channel, kernel size=2, stride=2, padding=1

Tanh

Example: MNIST

100 samples from test data set

100 generated samples using Autoencoder

- **20 epochs, batch size = 128, Adam optimizer, learning rate = 0.001**
- **Normalized images with mean 0.5 and std equals to 0.5**

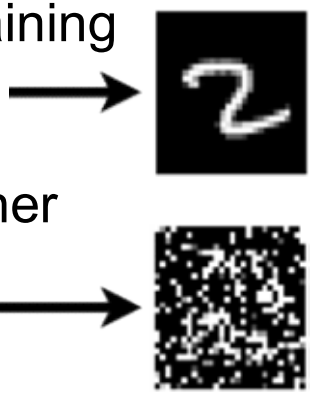
Autoencoder

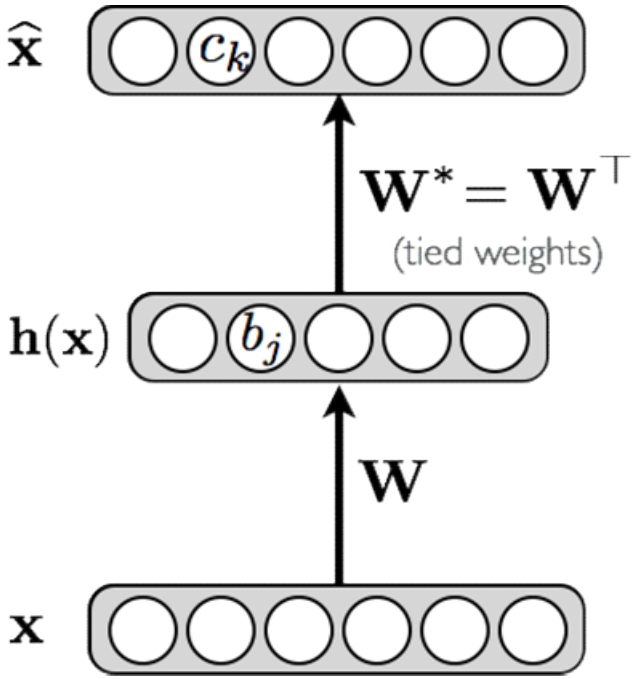
- How to extract meaningful hidden features?

1. **Undercomplete Representation**
2. **Injecting noise to the input (Denoising Autoencoder)**
3. **Penalizing the solution (Contractive Autoencoders)**

Undercomplete Representation

- Hidden layer is undercomplete if smaller than the input layer (bottleneck layer, e.g. dimensionality reduction):
  - hidden layer "compresses" the input
  - may compress well only for the training distribution
- Hidden units will be
  - good features for the training distribution
  - may not be robust to other types of input



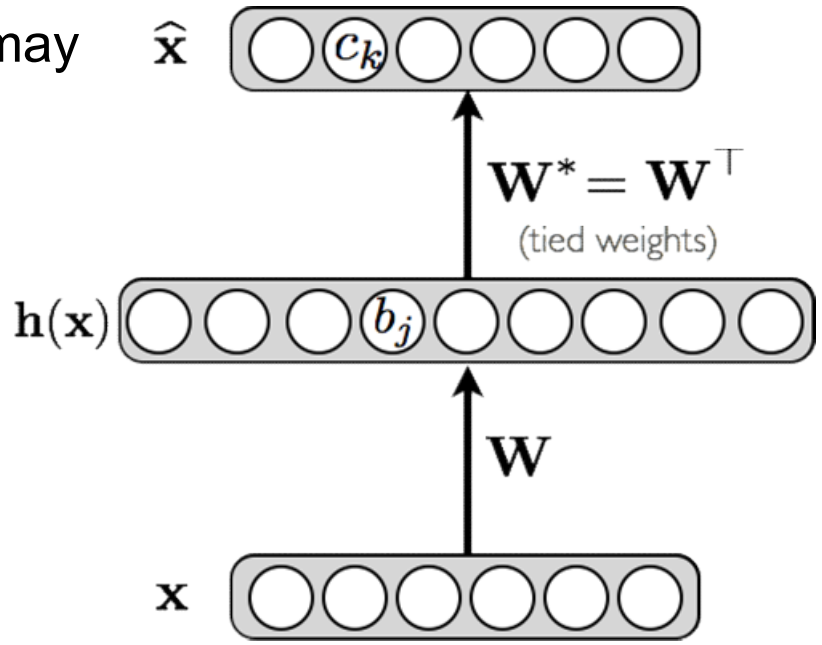


Overcomplete Representation

- Hidden layer is overcomplete if greater than the input layer

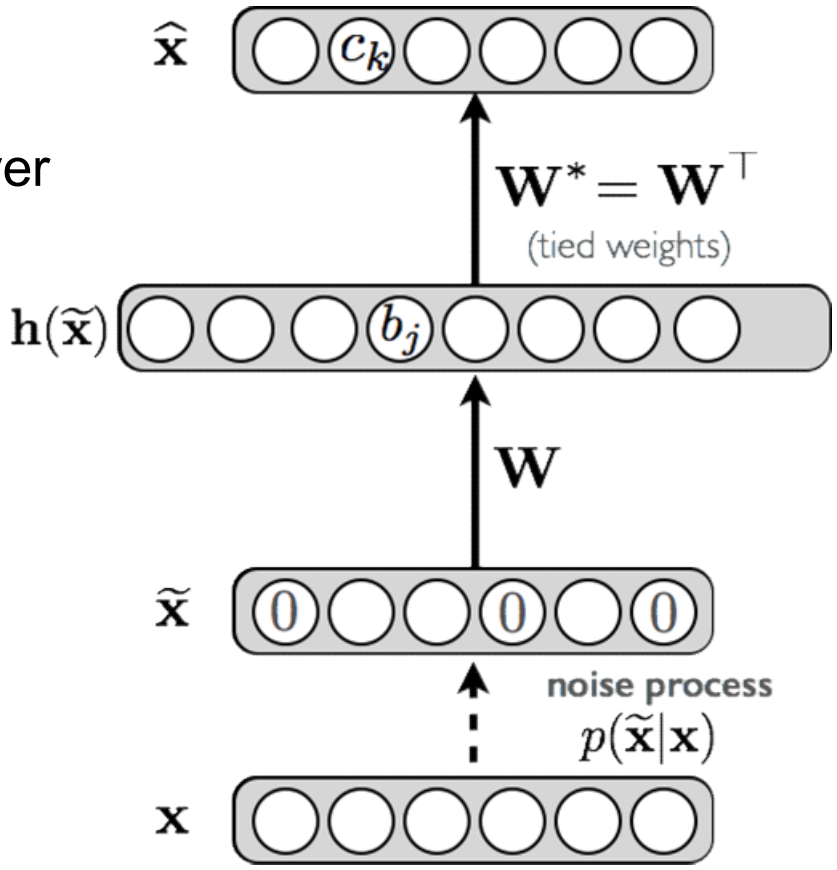
  - no compression in hidden layer
  - In some cases each hidden unit may copy a different input component

- No guarantee that the hidden units will extract meaningful structure



Denoising Autoencoder

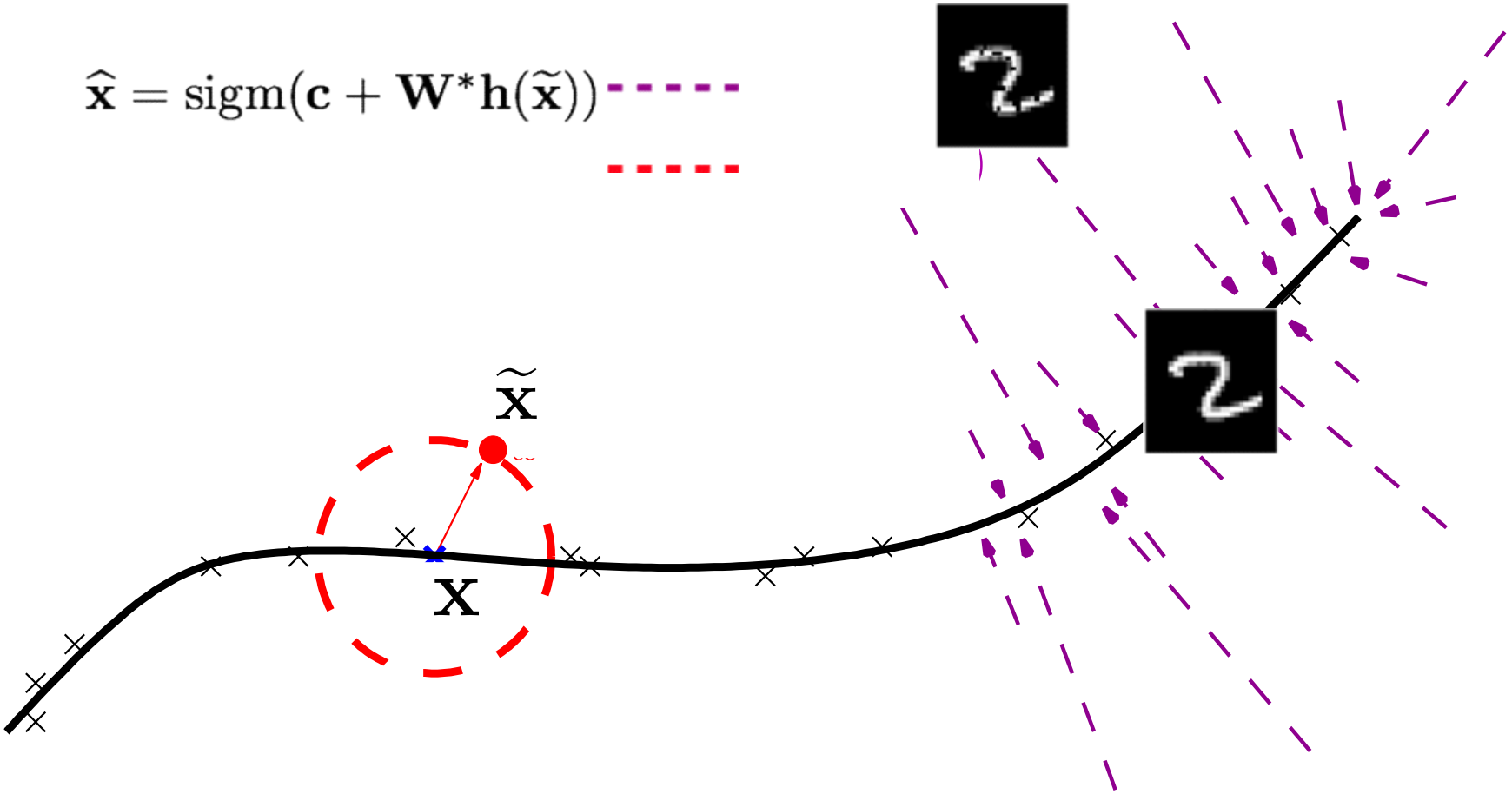
- Idea: representation should be robust to introduction of noise:
  - random assignment of subset of inputs to 0, with probability ν
  - Similar to dropouts on the input layer
  - Similar idea for Gaussian additive noise
- Reconstruction $\widehat{\mathbf{x}}$ computed from the corrupted input $\widetilde{\mathbf{x}}$
- Loss function compares $\widehat{\mathbf{x}}$ reconstruction with the noiseless input $\mathbf{x}$



(Vincent et.al., ICML 2008)

Denoising Autoencoder

$$\widehat{\mathbf{x}} = \mathrm{sigm}(\mathbf{c} + \mathbf{W}^* \mathbf{h}(\widetilde{\mathbf{x}}))$$

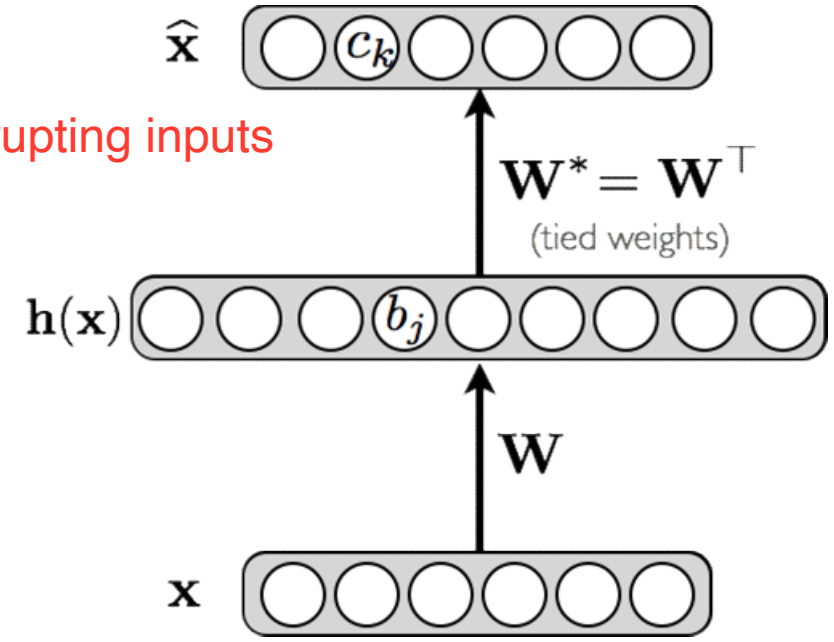


Contractive Autoencoder

- Alternative approach to avoid uninteresting solutions
  - add an explicit term in the loss that penalizes that solution

You want denoising behavior without explicitly corrupting inputs

- We wish to extract features that only reflect variations observed in the training set
  - we'd like to be invariant to the other variations



Contractive Autoencoder

- Consider the following loss function:

$$\underbrace{l(f(\mathbf{x}^{(t)}))}_{\text{Reconstruction Loss}} + \lambda \underbrace{||\nabla_{\mathbf{x}^{(t)}} \mathbf{h}(\mathbf{x}^{(t)})||_F^2}_{\text{Jacobian of Encoder}}$$

- Example for binary observations:

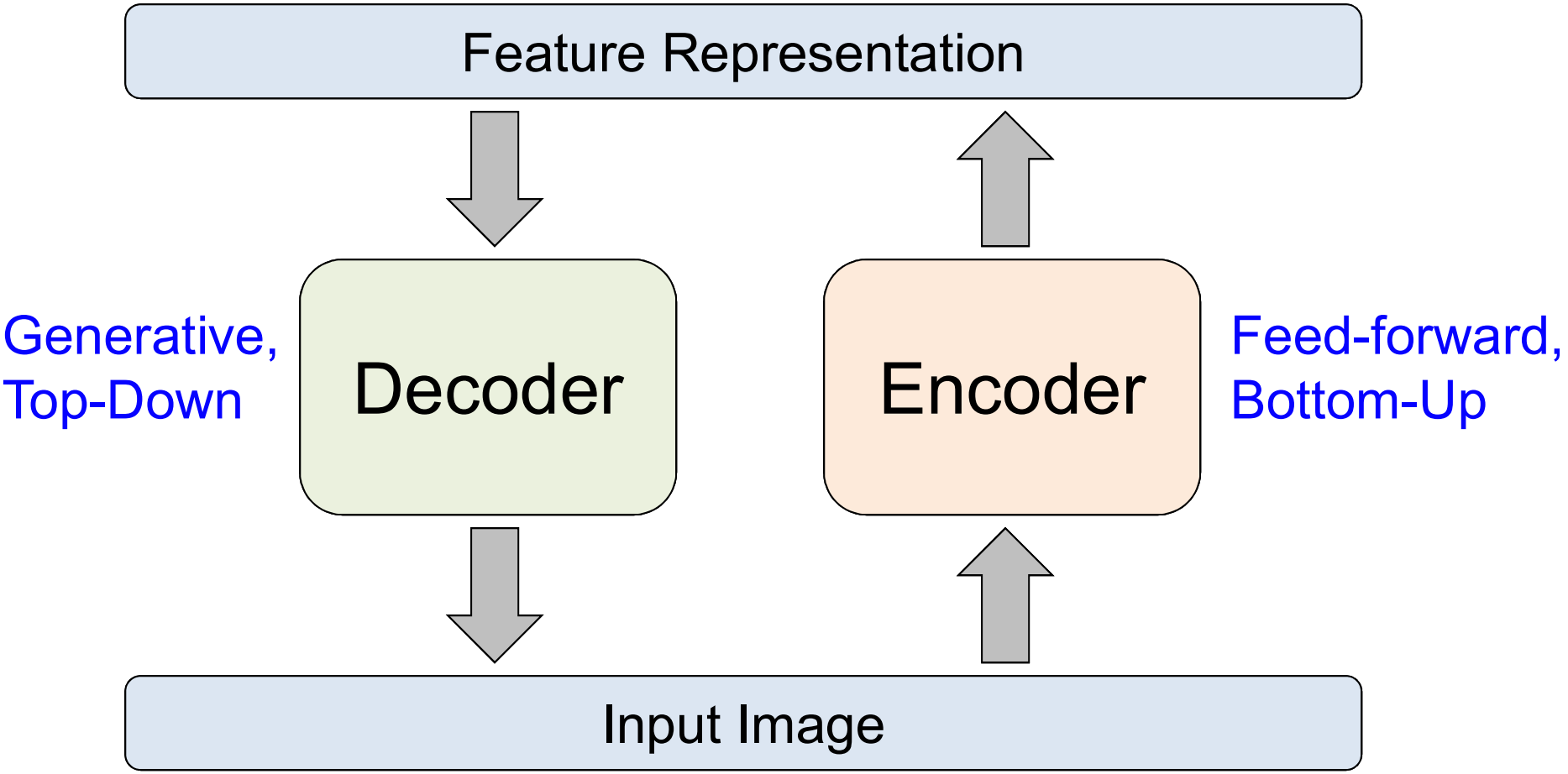
$$l(f(\mathbf{x}^{(t)})) = -\sum_k \left(x_k^{(t)} \log(\hat{x}_k^{(t)}) + (1 - x_k^{(t)}) \log(1 - \hat{x}_k^{(t)}) \right)$$

$$||\nabla_{\mathbf{x}^{(t)}} \mathbf{h}(\mathbf{x}^{(t)})||_F^2 = \sum_j \sum_k \left(\frac{\partial h(\mathbf{x}^{(t)})_j}{\partial x_k^{(t)}} \right)^2$$

Autoencoder attempts to preserve all information

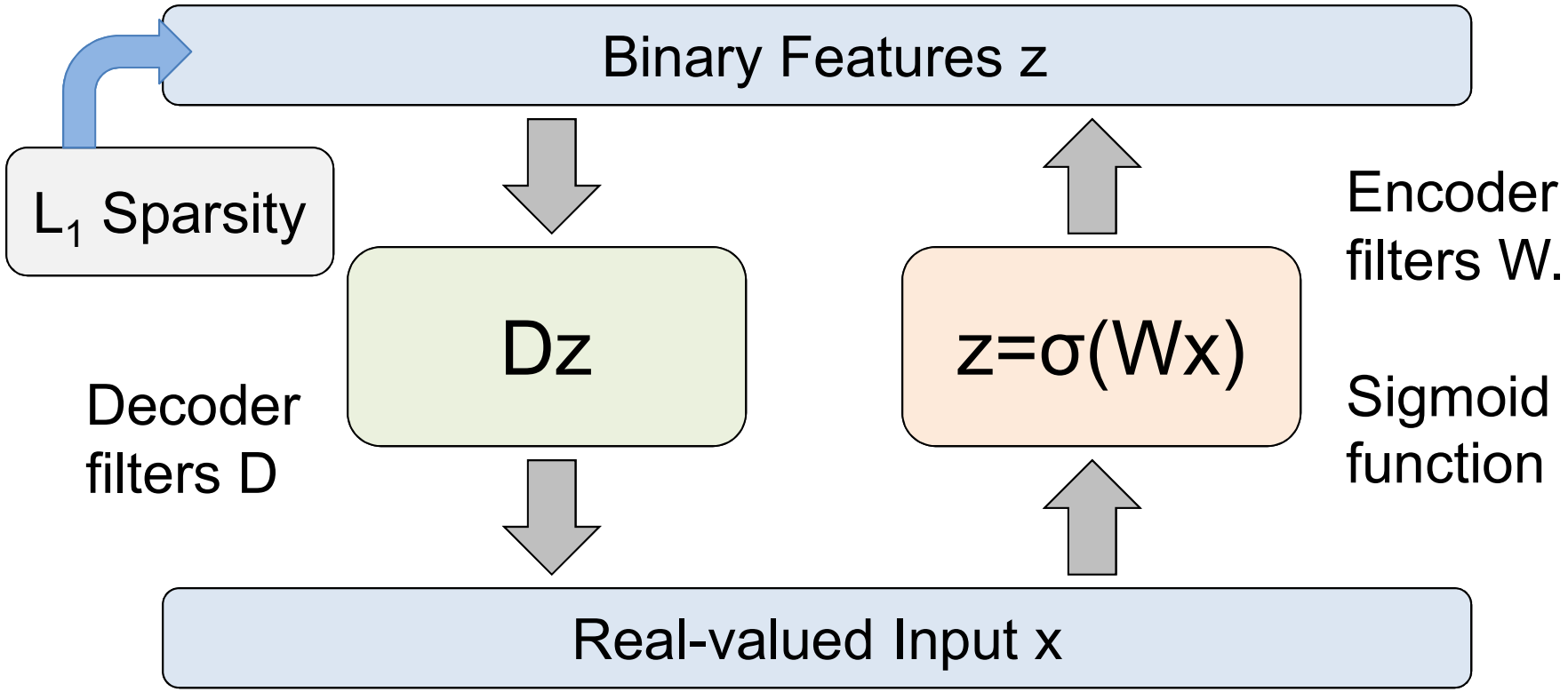
Encoder throws away all information

Autoencoder



- Details of what goes insider the encoder and decoder matter!

Predictive Sparse Decomposition

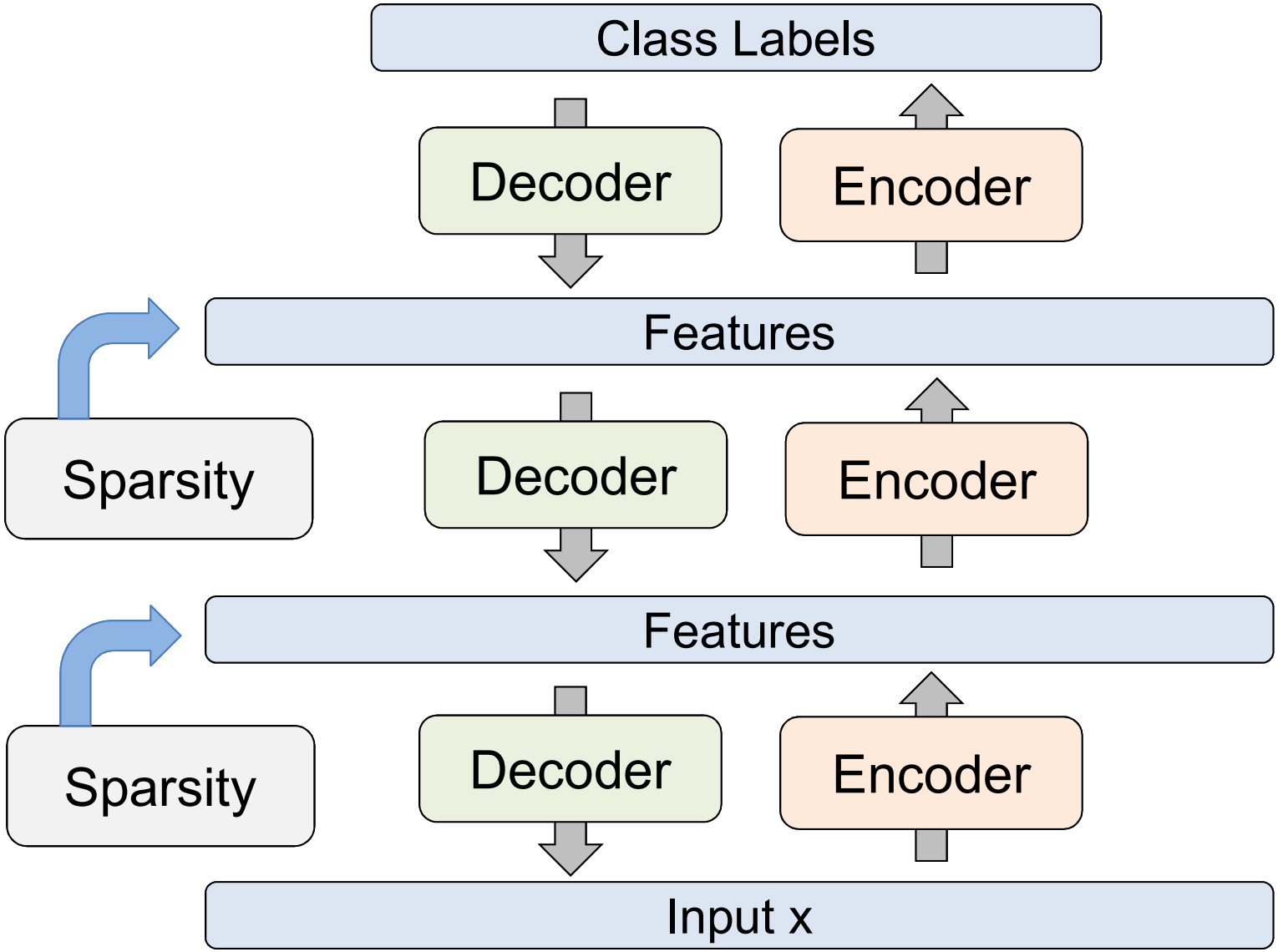


At training time

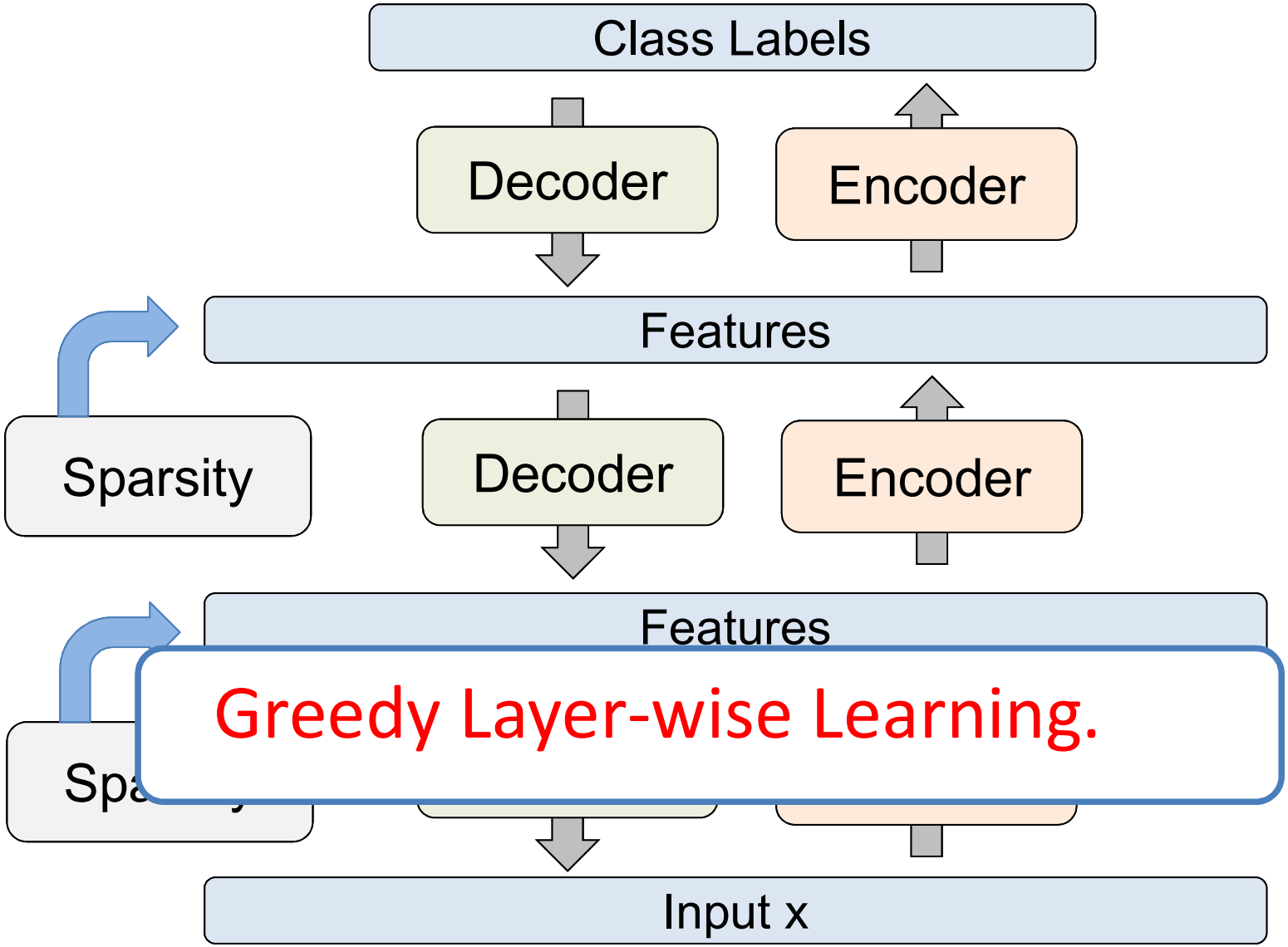
$$\min_{D,W,\mathbf{z}} \underbrace{||D\mathbf{z}-\mathbf{x}||_2^2+\lambda|\mathbf{z}|_1}_{\text{Decoder}} + \underbrace{||\sigma(W\mathbf{x})-\mathbf{z}||_2^2}_{\text{Encoder}}$$

Kavukcuoglu et al., '09

Stacked Autoencoders



Stacked Autoencoders



Linear Factor Models

Vahid Tarokh

ECE 685D, Fall 2025

Linear Factor Models

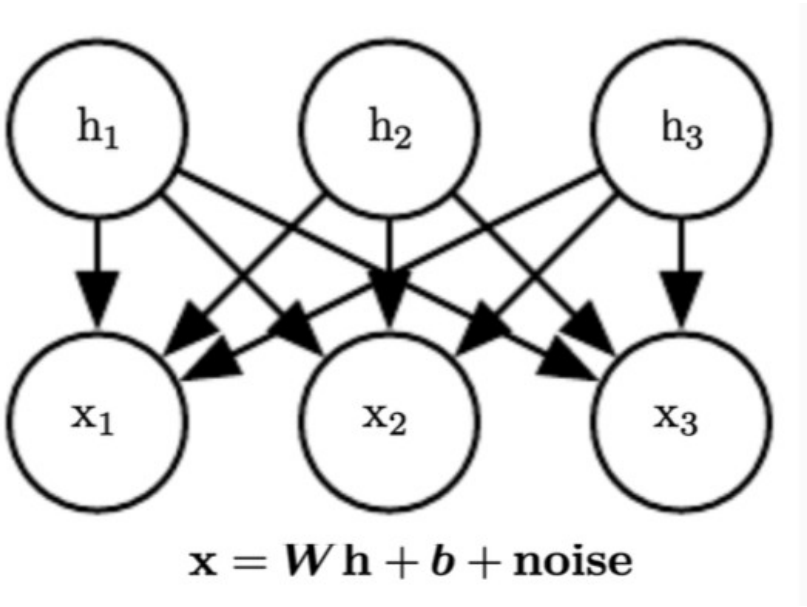
- As discussed before, many probabilistic models have latent variables.
- **Latent variables**, as opposed to **observable variables**, are variables that are not observed but instead inferred from observed variables
- **Linear factor models** are some of the simplest probabilistic models with latent variables.
- We may not implement any linear factor models to solve state-of-art problems, but LFMs provide a good building block for mixture models or deeper probabilistic models.
- Many of the approaches we discuss today are necessary to build generative models that more advanced deep models will expand upon

Linear Factor Models

- LFMs postulates that the data generation process is as follows:
  - Sample the explanatory factors h from a distribution p(h).
  - Sample the real-valued observable variables given the factors:

$$x = Wh + b + n$$

- Components of h are typically assumed to be i.i.d. and the noise n is assumed to be zero-mean Gaussians with independent (not necessarily i.i.d.) components.



Linear Factor Models

- In this setting W is a $q \times d$ matrix and h is d dimensional.
- The motivation is that if $q >> d$ then the latent variables h give a more parsimonious explanation of dependencies between components of the observations x.
- This must be reminiscent of PCA for the students in this course.
- In fact as we will see, LFM includes a many well-known techniques as special cases.

Factor Analysis

- Assume that the components of h are i.i.d. standard Gaussian. Then W colors x and b adds bias.
- Assume that the i-th noise component has variance σ_i^2.
- Then show that x is a multivariate normal with mean zero, mean b and variance

$$WW^T + \text{diag}(\sigma_1^2, \sigma_2^2, \ldots\ldots, \sigma_q^2).$$

- W and $\sigma_1^2, \sigma_2^2, \ldots\ldots, \sigma_q^2$ can be estimated using the MLE methods. Since no closed form exists, then iterative methods are commonly applied.
- Key issue is that given h components of x are ***independent*** but ***do not necessarily have the same variance.***
  - This is the key difference with classical PCA.
  - The subspace generated by the MLE estimate of W will not necessarily correspond to the principal subspace of the data.

Linear Factor Models

- Different types of LFMs make different choices about the form of the noise and of the prior p(h)
- The most important LFMs that we will discuss are:
  - PCA and Probabilistic PCA
  - Independent component analysis (ICA)
  - Slow feature analysis
  - Sparse coding

Probabilistic PCA

- Principal Component Analysis (PCA) is a old technique that is very well-established.
- Tipping and Bishop (1999) build on the work of Young (1940) and Whittle (1952) to propose a probabilistic version of PCA from LFM and factor analysis.
- Key assumption $\sigma_1^2 = \sigma_2^2 = \ldots.= \sigma_q^2 = \sigma^2$
- In this case, the MLEs in factor analysis have a closed form, and model parameter estimation can be performed iteratively and efficiently. Define:

$$\text{C} = WW^T + \sigma^2 I\,.$$

$$S = [\sum_{i=1}^{N}(x_{\text{i}}\text{- b})(x_i - b)^T]/\text{N}$$

C: the model covariance under PPCA. This is the covariance predicted by the PPCA model

S: the sample covariance.
This is exactly the empirical covariance of your dataset after subtracting the mean b

Then PPCA finds parameters: W(ML), σ^2(ML), b(ML) such that: C≈S. This is Maximum Likelihood estimation under PPCA.

Probabilistic PCA

- MLE for b is the mean of data, and for W is given by

$$W_{ML} = U_q\left(\Lambda_q - \sigma^2 I_d\right)^{1/2} R,$$

- With the q column vectors in the q × d matrix U_q denoting the principle eigenvectors of S with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_d$ in the d × d diagonal matrix Λ_d and R is an arbitrary d × d orthogonal matrix:

$$\sigma_{ML}^2 = \frac{1}{q-d}\sum_{j=d+1}^{q} \lambda_j.$$

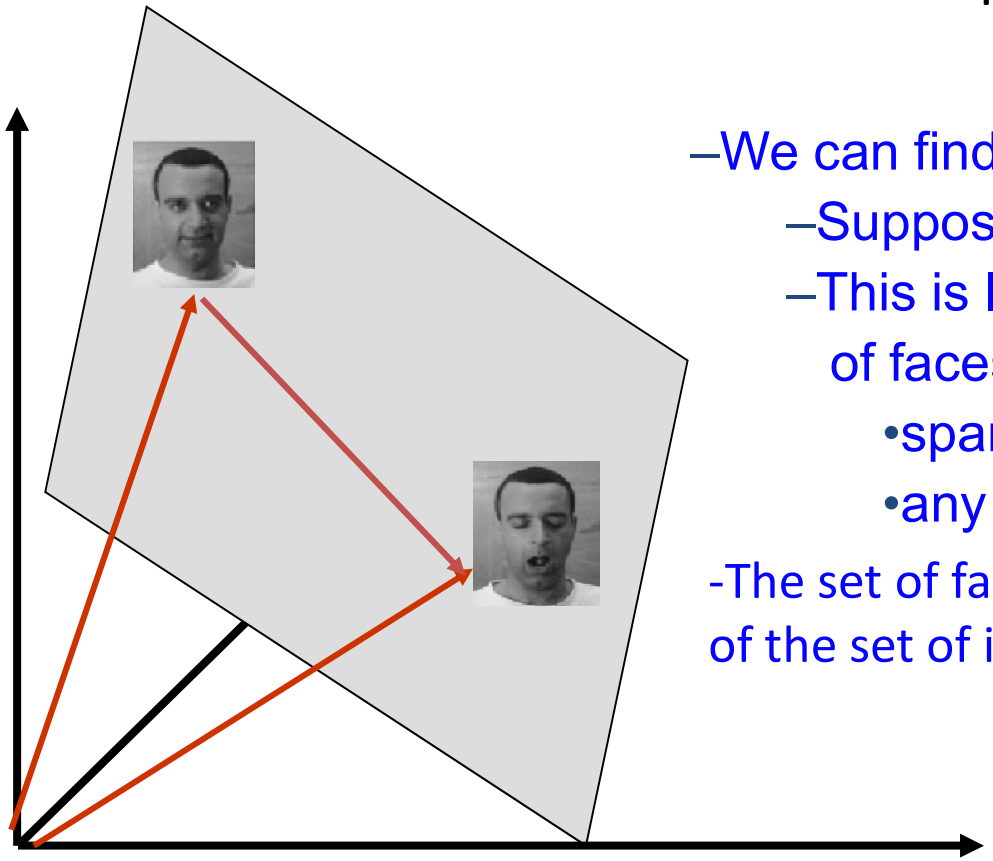
- Question: why does the existence of R makes sense in the above.

Probabilistic PCA

- Interestingly as $\sigma^2 \to 0$ we recover the classical PCA.
- This means that both PPCA and PCA are special cases of LFMs.

Application of PCA

- **Face Recognition**
- An image is a point in a high-dimensional space
  - An $N\ x\ M$ image is a point in R^{NM}
  - We can define vectors in this space



–We can find the best subspace using PCA

–Suppose this is a K-dimensional subspace

–This is like fitting a "hyper-plane" to the set of faces

•spanned by vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_K}$

•any face $\mathbf{x} \approx a_1\mathbf{v_1} + a_2\mathbf{v_2} + , \ldots, + a_K\mathbf{v_K}$

-The set of faces is a "subspace" of the set of images.

Application of PCA

- Let $F_1, F_2, \ldots, F_M$ be a set of training face images.
- Let F be their mean and $\Phi_i = F_i$ – F
- Use principal components to compute the eigenvectors and eigenvalues of the covariance matrix of the $\Phi_i's$
- Choose the vector u of most significant M eigenvectors to use as the basis.
- Each face is represented as a linear combination of eigenfaces
- $u = (u_1, u_2, u_3, u_4, u_5)$;
- $F_{27} = a_1 * u_1 + a_2 * u_2 + \ldots + a_5 * u_5$

unknown face image → convert to its eigenface representation → $\Omega = (\Omega_1, \Omega_2, \ldots, \Omega_m)$

Find the face class k that minimizes

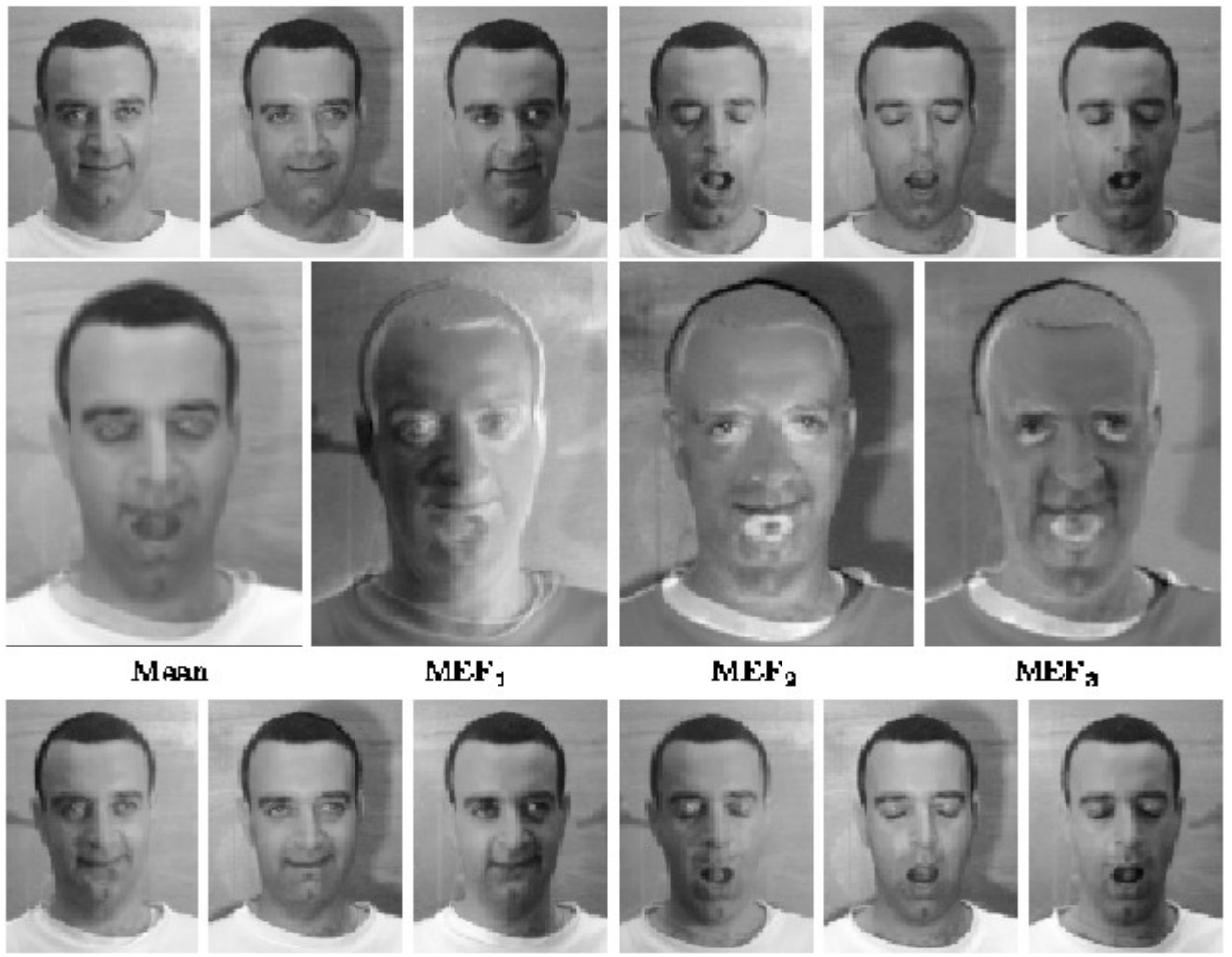
$$\varepsilon_k = ||\Omega - \Omega_k||$$

Application of PCA

training images

mean image

linear approxi-mations



3 eigen-images

ICA

- Independent component analysis can also be cast as a special case of LFMs.
- In our setting, we assume $b = 0$ (this can be achieved by translation) and $n = 0$ (for simplicity):

$$\mathrm{x} = Ah.$$

  - $\mathrm{x} \in \mathbb{R}^n$ is called mixture vector.
  - $\mathrm{h} \in \mathbb{R}^n$ is called hidden, factor, or latent vector.
  - $\mathrm{A} \in \mathbb{R}^{n \times n}$ is called mixing matrix. It can also be generalized to rectangular matrix.
  - This can be thought of mixing sources given by components of h using mixing matrix A.
  - Both A and h are **unknown** (blind source (signal) separation)

ICA

- Here, we consider the **linear ICA**. The nonlinear version is related to the Autoencoders.
- **Assumptions of ICA:**
  - Each component of h is assumed to be statistically **independent**.
  - Independent components must have **non-Gaussian** distribution.
- **Ambiguities of ICA:**
  - Estimating the latent components up to a scaling factor

$$\mathrm{x} = (\alpha A)(\frac{1}{\alpha} h) \quad \text{for some } \alpha > 0.$$

    - Can force $\mathrm{E}(h_i^2) = 1$; however, there is still an ambiguity for the sign of hidden components.
    - Fortunately, this is insignificant in most applications.
  - Estimating the components up to the order of them

$$\mathrm{x} = AP^{-1}Ph \quad \text{where } P \text{ is a permutation matrix.}$$

ICA

- **Why non-Gaussian assumption?**
  - If h is jointly Gaussian, vector Rh is also Gaussian.
  - The mixing matrix can be any matrix given by AR^{-1} (A is unidentifiable) .

- ICA finds the independent components by maximizing (in some measure) distances of the estimated components from the Gaussian distribution.
  - Given **observed random vectors,** $\mathbf{x_i} \in \mathbb{R}^{\boldsymbol{n}}$, for $i = 1,2, \ldots m$, the goal is to find an unmixing matrix W and laten vector h.
  - Once W is estimated, the latent components of a given test mixture vector, x^* is computed by $h^* = W^{-1}\,\mathrm{x}^*$.
  - **How to estimate W matrix:**
    - W is estimated such that W^{-1} x is far from Gaussian.

ICA

- **Measure of non-Gaussianity:**
  - The absolute or squared kurtosis (the fourth-order cumulant).
  - For a R.V. Y the kurtosis is defined as follows:

$$\boldsymbol{kurt(Y) = E(Y^4) - 3(E(Y^2))^2}$$

  - If Y is a Gaussian r.v., then kurt(Y) $= 0$.
  - For most of non-Gaussian r.v's, the kutosis is non-zero.

- **Methods of estimating the independent components:**
  - Maximizing Neg-entropy
  - Maximization of non-Gaussianity (Using Kurtosis)

- Exercise: Refresh your knowledge of both PCA and ICA.

Application of ICA

Blind source separation

Consider 3 sources, defined as follows:

- Source 1: $S_1 = Sin(2t)$
- Source 2: $S_2 = Squarewave(3t)$
- Source 3: $S_3 = Sawtooth(2\pi t)$

Also, assume the mixing matrix is given by $W = \begin{bmatrix} 1 & 1 & 1 \\ 0.5 & 2 & 1 \\ 1.5 & 1 & 2 \end{bmatrix}$; as a result, observation is $X = WS^T$.

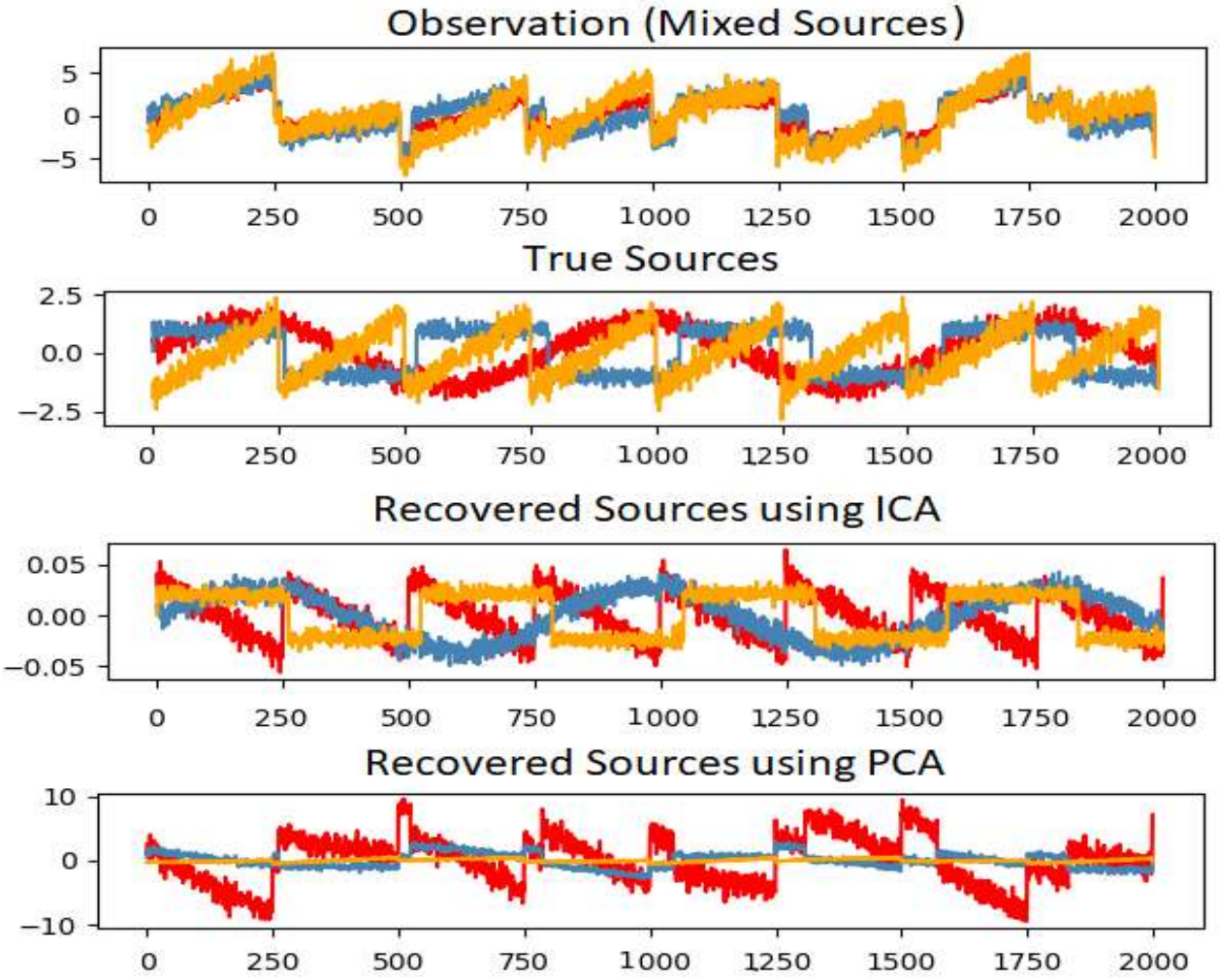
Now, our goal is to separate the three sources from X using ICA.

```
n_samples = 2000                          # Number of Samples
time = np.linspace(0, 8, n_samples)       # Time range

s1 = np.sin(2 * time)                     # Source 1: sinusoidal signal
s2 = np.sign(np.sin(3 * time))            # Source 2: square signal
s3 = signal.sawtooth(2 * np.pi * time)    # Source 3: saw tooth signal

S = np.c_[s1, s2, s3]
S += 0.2 * np.random.normal(size=S.shape)  # Add noise

S /= S.std(axis=0)                        # Standardizing data
W = np.array([[1, 1, 1], [0.5, 2, 1.0], [1.5, 1.0, 2.0]])   # Mixing matrix, W_3X3
X = np.dot(W, S.T)                        # Observations (Mixed Sources)
X =X.T

ica = FastICA(n_components=3)             # Compute ICA (Using FastICA [1] algorithm)

Ŝ = ica.fit_transform(X)                  # Recovered Sources
Ŵ = ica.mixing_                           # Estimated mixing matrix
```

[1]. A. Hyvärinen, "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis", IEEE Transactions on Neural Networks 10(3):626-634, 1999

Application of ICA

- **Blind source separation**



https://scikit-learn.org/stable/auto_examples/decomposition/plot_ica_blind_source_separation.html#sphx-glr-auto-examples-decomposition-plot-ica-blind-source-separation-py

Slow Feature Analysis

- Slow feature analysis is motivated by a general principle called the **slowness principle**.
  - The idea is that the important characteristics of scenes change very slowly compared to the individual measurements that make up a description of a scene.
  - For example, in computer vision, individual pixel values can change very rapidly. If a zebra moves from left to right across the image, an individual pixel will rapidly change from black to white and back again as the zebra's stripes pass over the pixel. By comparison, the feature indicating whether a zebra is in the image will not change at all, and the feature describing the zebra's position will change slowly.
  - We therefore may wish to regularize our model to learn features that change slowly over time.

Slow Feature Analysis

- The slowness principle predates slow feature analysis and has been applied to a wide variety of models.

- In general, we can apply the slowness principle to any differentiable model trained with gradient descent. The slowness principle may be introduced by adding a term to the cost function of the form

$$\lambda \sum_t L(f(\boldsymbol{x}^{(t+1)}), f(\boldsymbol{x}^{(t)})),$$

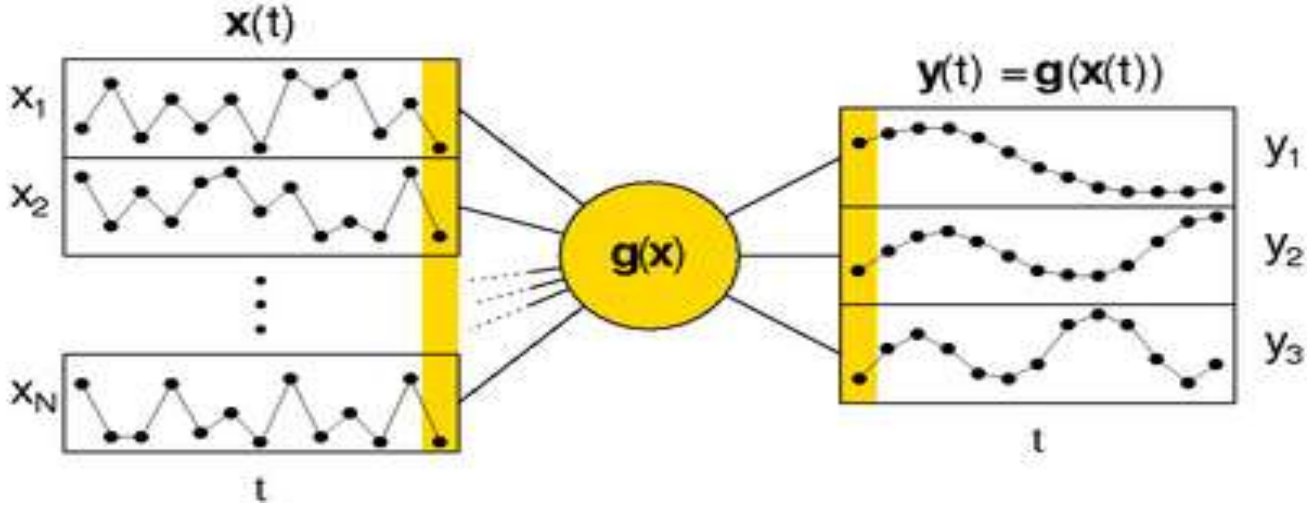
where λ is a hyper-parameter determining the strength of the slowness regularization term, t is the index into a time sequence of examples, f is the feature extractor to be regularized, and L is a loss function.

Slow Feature Analysis

- Given a set of time-varying input signals, **x**(t), SFA learns instantaneous, non-linear functions **g**(**x**) that transform **x** into slowly-varying output signals, **y**(t).

- The optimization procedure guarantees that SFA returns the global optimum for **g** (i.e., the slowest output signal) in a given function space.

- As the transformations must be instantaneous, trivial solution like low-pass filtering are not possible.



Slow Feature Analysis

- If we restrict the nonlinear functions $g_j(\cdot)$ to a finite dimensional function space, such as all polynomials of degree two, we can transform the variational problem into a more conventional optimization over the coefficients of the basis of the function space (e.g., all monomials of degree 1 and 2).

- In this way, the problem becomes simpler to solve, and one can use algebraic methods, which are the basis of the slow feature analysis algorithm.

Slow Feature Analysis

- Consider as a simple example the two-dimensional input signal $x_1(t) = \sin(t) + \cos(11t)^2$ and $x_2(t) = \cos(11t)$.

- Both components are quickly varying but hidden in the signal is the slowly varying $y(t) = x_1(t) - x_2^2(t)$, which can be extracted with a polynomial of degree two, namely $g(\mathbf{x}) = x_1 - x_2^2$.

- **To find the function g(x) that extracts the slow feature from the input signal, one proceeds as follows.**

- **The non-linear problem is transformed to a linear one by expanding the input into the space of nonlinear functions under consideration.**

- **Subsequently we seek for a linear transform** f(x; θ).

Slow Feature Analysis

- SFA algorithm (Wiskott and Sejnowski, 2002) then finds the linear transformation f(x; θ) , then solving the optimization problem

$$\min_{\boldsymbol{\theta}} \mathbb{E}_t(f(\boldsymbol{x}^{(t+1)})_i - f(\boldsymbol{x}^{(t)})_i)^2$$

Subject to

$$\mathbb{E}_t f(\boldsymbol{x}^{(t)})_i = 0$$

and

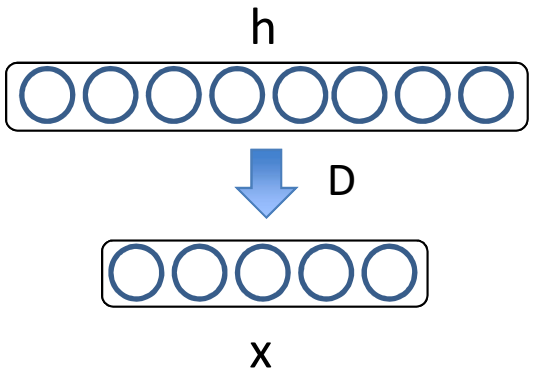
$$\mathbb{E}_t[f(\boldsymbol{x}^{(t)})_i^2] = 1.$$

- Question: Why are all these constraints necessary?
- We will provide examples in HW.

Sparse Coding

- Sparse coding (Olshausen & Field, 1996). Originally developed to explain early visual processing in the brain (edge detection).

- For each input $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can good reconstruct the original input $\mathbf{x}^{(t)}$



Sparse Coding

- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can good reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

Reconstruction: $\widehat{\mathbf{x}}^{(t)}$

Sparsity vs. reconstruction control

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

Reconstruction error

Sparsity penalty

Sparse Coding

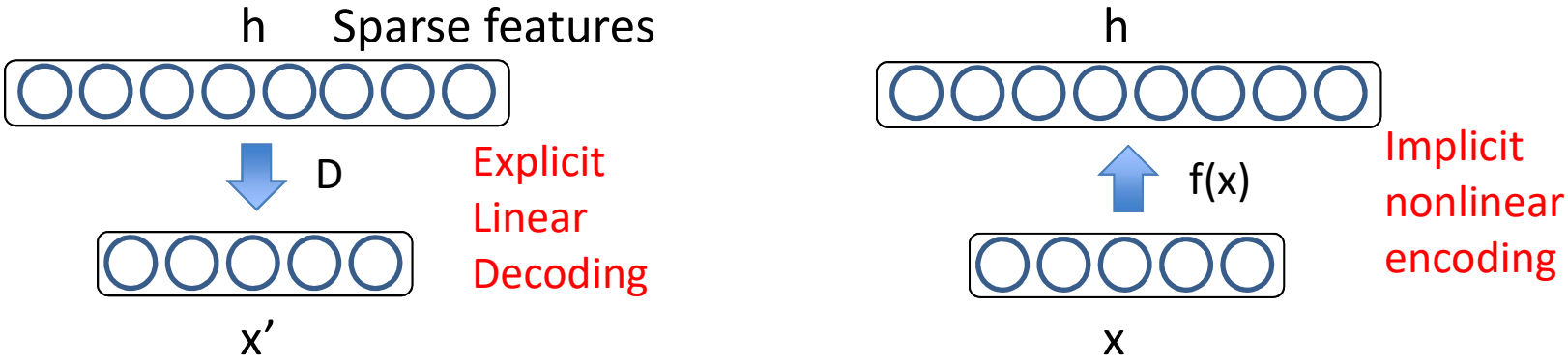
- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - it is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - we can good reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

Interpreting Sparse Coding

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

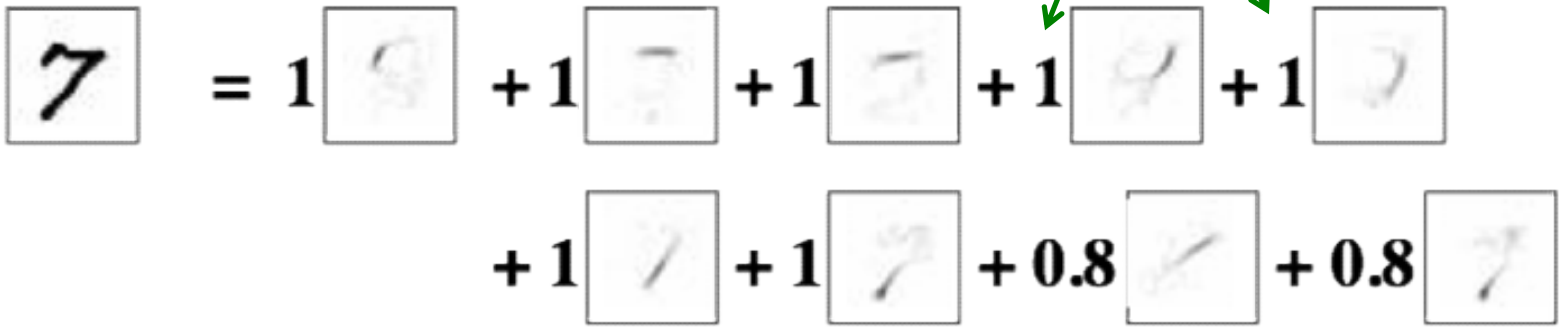


- Sparse, over-complete representation **h.**
- Encoding **h** = f(**x**) is implicit and nonlinear function of **x**.
- Reconstruction (or decoding) **x'** = Dh is linear and explicit.

Sparse Coding

- We can also write:

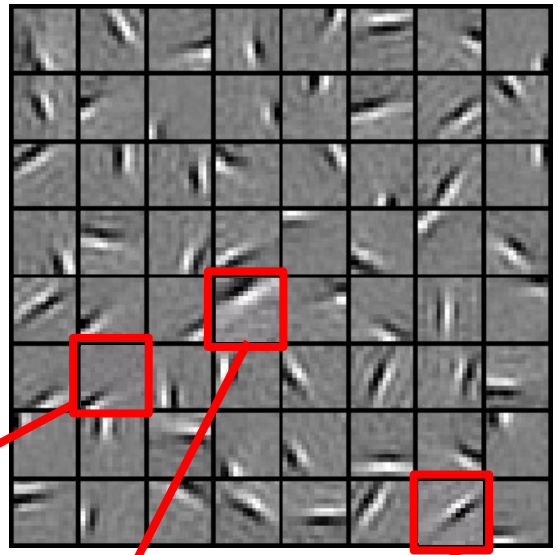
$$\hat{\mathbf{x}}^{(t)} = \mathbf{D}\,\mathbf{h}(\mathbf{x}^{(t)}) = \sum_{k \text{ s.t. } h(\mathbf{x}^{(t)})_k \neq 0} \mathbf{D}_{.,k}\, h(\mathbf{x}^{(t)})_k$$



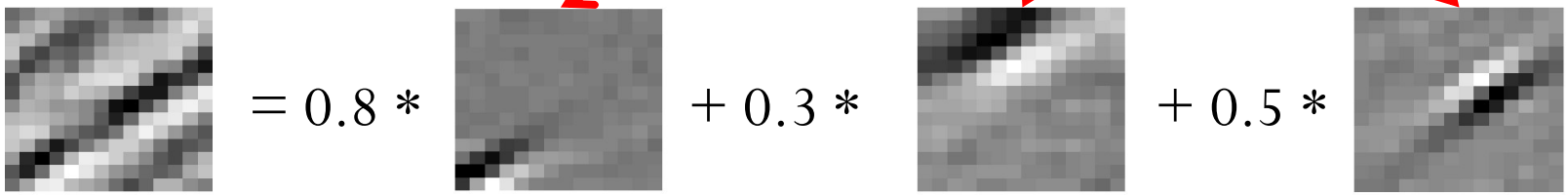
Sparse Coding

Natural Images

Learned bases: "Edges"



New example



$$x = 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{65}$$

[0, 0, … **0.8**, …, **0.3**, …, **0.5**, …] = coefficients (feature representation)

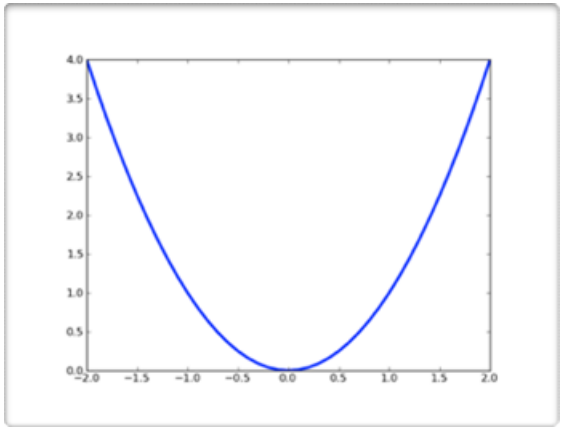
Slide Credit: Honglak Lee

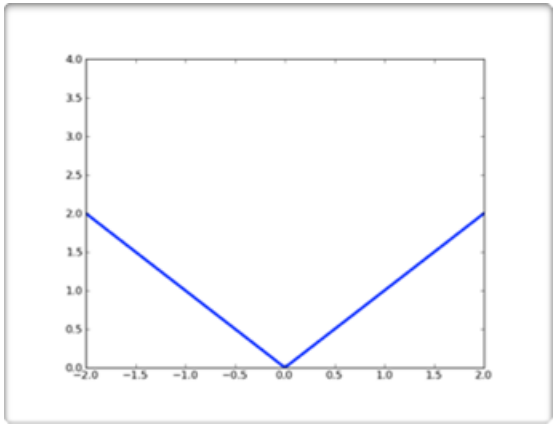
Inference

- How do we compute $\mathbf{h}(\mathbf{x}^{(t)})$?
  - We need to optimize:

$$l(\mathbf{x}^{(t)}) = \frac{1}{2}||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda||\mathbf{h}^{(t)}||_1$$

  - This is Lasso.





  - We could use a gradient descent method:

$$\nabla_{\mathbf{h}^{(t)}} l(\mathbf{x}^{(t)}) = \mathbf{D}^\top(\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\,\mathrm{sign}(\mathbf{h}^{(t)})$$

Inference

- For a single hidden unit:

$$\frac{\partial}{\partial h_k^{(t)}} l(\mathbf{x}^{(t)}) = (\mathbf{D}_{\cdot,k})^\top (\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\,\mathrm{sign}(h_k^{(t)})$$

  - issue: L_1 norm not differentiable at 0
  - very unlikely for gradient descent to "land" on $h_k^{(t)} = 0$ (even if it's the solution)

- Solution: if $h_k^{(t)}$ changes sign because of L_1 norm gradient, clamp to 0.

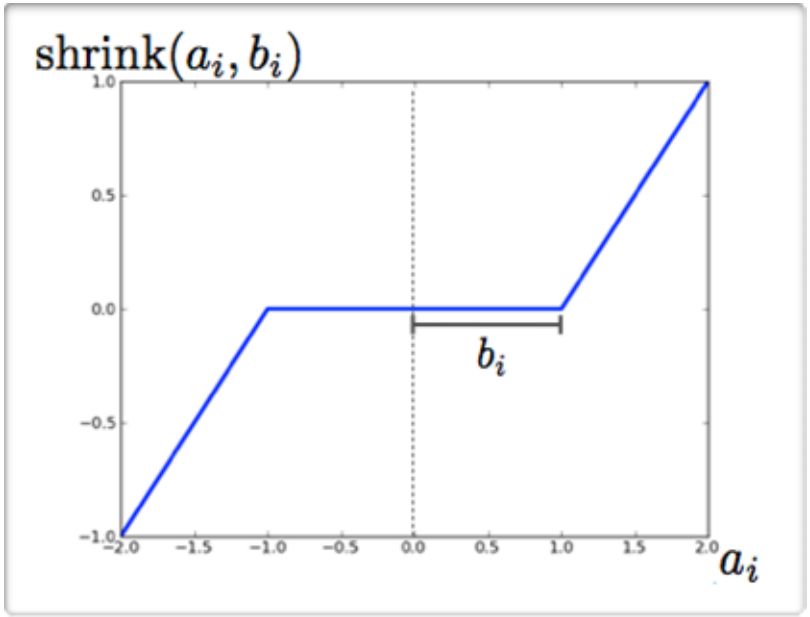
ISTA Algorithm

• This process corresponds to the ISTA (Iterative Shrinkage and Thresholding) Algorithm:

- Initialize $\mathbf{h}^{(t)}$ (for example to 0)
- While $\mathbf{h}^{(t)}$ has not converged

$$\mathbf{h}^{(t)} \Longleftarrow \mathbf{h}^{(t)} - \alpha\, \mathbf{D}^{\top}(\mathbf{D}\, \mathbf{h}^{(t)} - \mathbf{x}^{(t)})$$

$$\mathbf{h}^{(t)} \Longleftarrow \text{shrink}(\mathbf{h}^{(t)}, \alpha\, \lambda)$$



where

$$\text{shrink}(\mathbf{a}, \mathbf{b}) = [\ldots, \text{sign}(a_i)\ \max(|a_i| - b_i, 0), \ldots]$$

• Will converge if $\frac{1}{\alpha}$ is bigger than the largest eigenvalue of $\mathbf{D}^{\top}\mathbf{D}$

Sparse Coding and Dictionary Learning

Vahid Tarokh

ECE 685D, Fall 2025

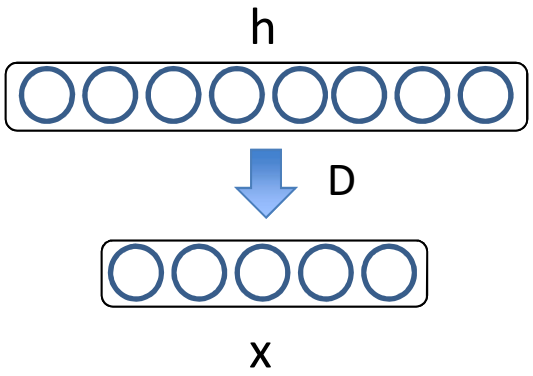
Unsupervised Learning

- Unsupervised learning: we only use the inputs $\mathbf{x}^{(t)}$ for learning
  - automatically extract meaningful features for your data
  - leverage the availability of unlabeled data
  - add a data-dependent regularizer to training (in some cases)

- We will consider 3 models for unsupervised learning that will form the basic building blocks for deeper models:
  - Autoencoders
  - Sparse Coding
  - Restricted Boltzmann Machines

Sparse Coding

- Sparse coding was originally developed in AI literature to explain early visual processing in the brain (edge detection) [Olshausen & Field, 1996].
- For each input $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - The representation is sparse: the vector $\mathbf{h}^{(t)}$ has many zeros
  - It can be used to well reconstruct the original input $\mathbf{x}^{(t)}$



Sparse Coding

- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - The representation is sparse: the vector $\mathbf{h}^{(t)}$ has many zeroes
  - It can be used to well reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

Reconstruction: $\widehat{\mathbf{x}}^{(t)}$

Sparsity vs. reconstruction control

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

Reconstruction error

Sparsity penalty

Sparse Coding

- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - The representation is sparse: the vector $\mathbf{h}^{(t)}$ has many zeroes
  - It can be used to well reconstruct the original input $\mathbf{x}^{(t)}$
- In other words:

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

  - we also constrain the columns of D to be of norm 1
  - otherwise, D could grow big while h becomes small to satisfy the L_1 constraint

Sparse Coding

- For each $\mathbf{x}^{(t)}$ find a latent representation $\mathbf{h}^{(t)}$ such that:
  - The representation is sparse: the vector $\mathbf{h}^{(t)}$ has many zeroes
  - It can be used to well reconstruct the original input $\mathbf{x}^{(t)}$

- In other words:

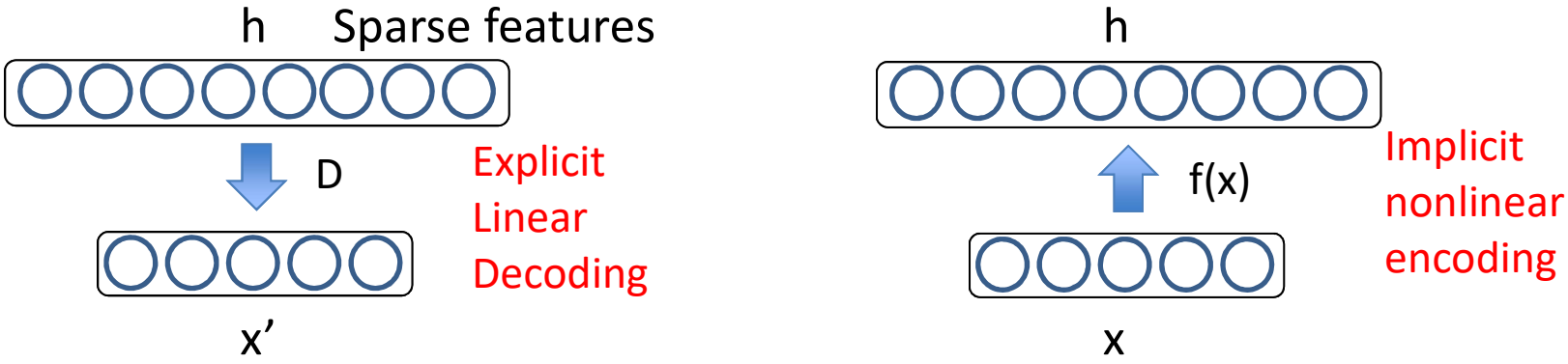
$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

  - D is equivalent to the autoencoder output weight matrix
  - However, $\mathbf{h}(\mathbf{x}^{(t)})$ is now a complicated function of $\mathbf{x}^{(t)}$
  - Encoder is the minimization problem:

$$\mathbf{h}(\mathbf{x}^{(t)}) = \arg\min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

Interpreting Sparse Coding

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda ||\mathbf{h}^{(t)}||_1$$

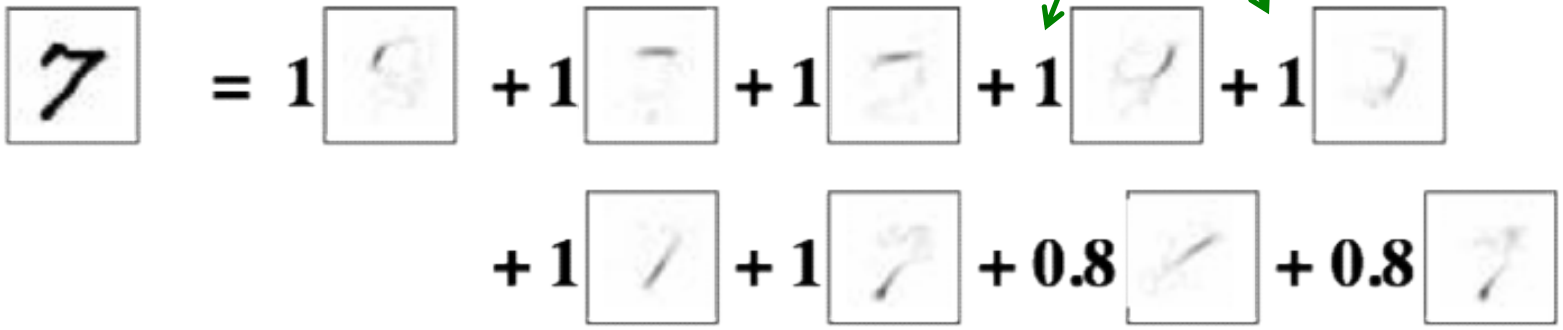


- Sparse, over-complete representation **h.**
- Encoding **h** = f(**x**) is implicit and nonlinear function of **x**.
- Reconstruction (or decoding) **x'** = Dh is linear and explicit.

Sparse Coding

- We can also write:

$$\hat{\mathbf{x}}^{(t)} = \mathbf{D}\,\mathbf{h}(\mathbf{x}^{(t)}) = \sum_{\substack{k \text{ s.t.} \\ h(\mathbf{x}^{(t)})_k \neq 0}} \mathbf{D}_{.,k}\, h(\mathbf{x}^{(t)})_k$$

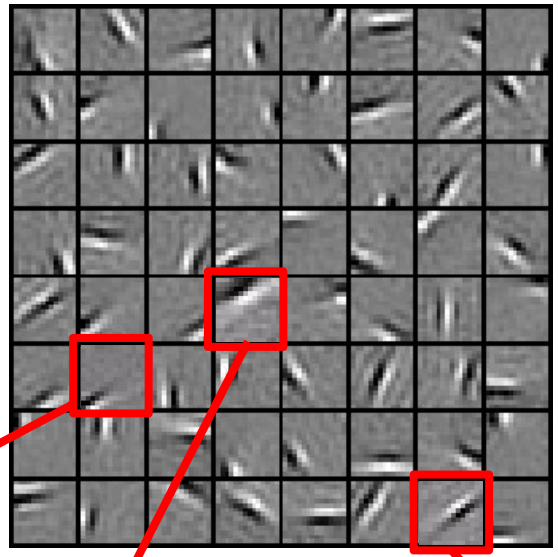


- D is often referred to as Dictionary
- In certain applications, we know what dictionary matrix to use
- In many cases, we have to learn it

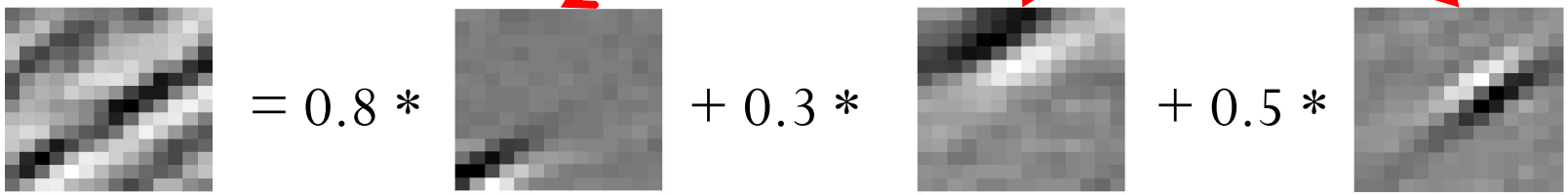
Sparse Coding

Natural Images

Learned bases: "Edges"



New example



$$x = 0.8 * \phi_{36} + 0.3 * \phi_{42} + 0.5 * \phi_{65}$$

[0, 0, … **0.8**, …, **0.3**, …, **0.5**, …] = coefficients (feature representation)

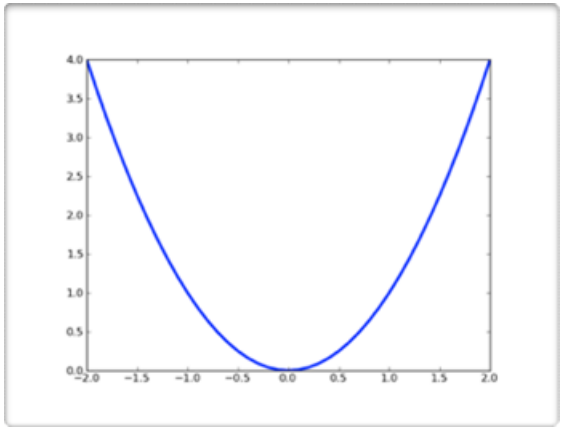
Slide Credit: Honglak Lee

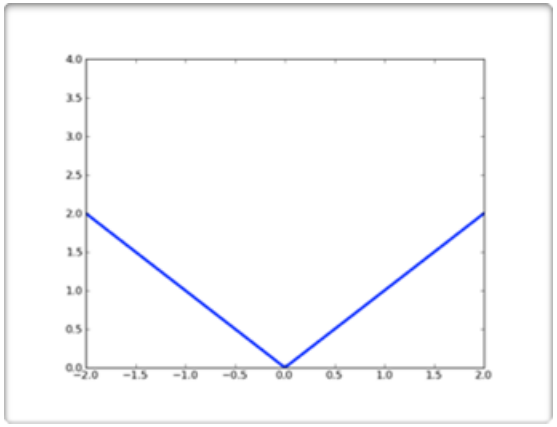
Inference

- Given dictionary D , how do we compute $\mathbf{h}(\mathbf{x}^{(t)})$?
  - We need to optimize:

$$l(\mathbf{x}^{(t)}) = \frac{1}{2}||\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}^{(t)}||_2^2 + \lambda||\mathbf{h}^{(t)}||_1$$

  - This is Lasso.





  - We could use a gradient descent method:

$$\nabla_{\mathbf{h}^{(t)}} l(\mathbf{x}^{(t)}) = \mathbf{D}^\top(\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\,\mathrm{sign}(\mathbf{h}^{(t)})$$

Inference

- For a single hidden unit:

$$\frac{\partial}{\partial h_k^{(t)}} l(\mathbf{x}^{(t)}) = (\mathbf{D}_{\cdot,k})^\top (\mathbf{D}\,\mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\,\mathrm{sign}(h_k^{(t)})$$

  - issue: L_1 norm not differentiable at 0
  - very unlikely for gradient descent to "land" on $h_k^{(t)} = 0$ (even if it's the solution)

- Solution: if $h_k^{(t)}$ changes sign because of L_1 norm gradient, clamp to 0.

Inference

- For a single hidden unit:

$$\frac{\partial}{\partial h_k^{(t)}} l(\mathbf{x}^{(t)}) = (\mathbf{D}_{\cdot,k})^\top (\mathbf{D}\ \mathbf{h}^{(t)} - \mathbf{x}^{(t)}) + \lambda\, \mathrm{sign}(h_k^{(t)})$$

- Solution: if $h_k^{(t)}$ changes sign because of L1 norm gradient, clamp to 0.

- Each hidden unit update would be performed as follows:

  - $h_k^{(t)} \Longleftarrow h_k^{(t)} - \alpha (\mathbf{D}_{\cdot,k})^\top (\mathbf{D}\ \mathbf{h}^{(t)} - \mathbf{x}^{(t)})$ ← Update from reconstruction
  - If $\mathrm{sign}(h_k^{(t)}) \neq \mathrm{sign}(h_k^{(t)} - \alpha\ \lambda\, \mathrm{sign}(h_k^{(t)}))$ then $h_k^{(t)} \Longleftarrow 0$
  - Else $h_k^{(t)} \Longleftarrow h_k^{(t)} - \alpha\ \lambda\, \mathrm{sign}(h_k^{(t)})$ ← Update sparsity term

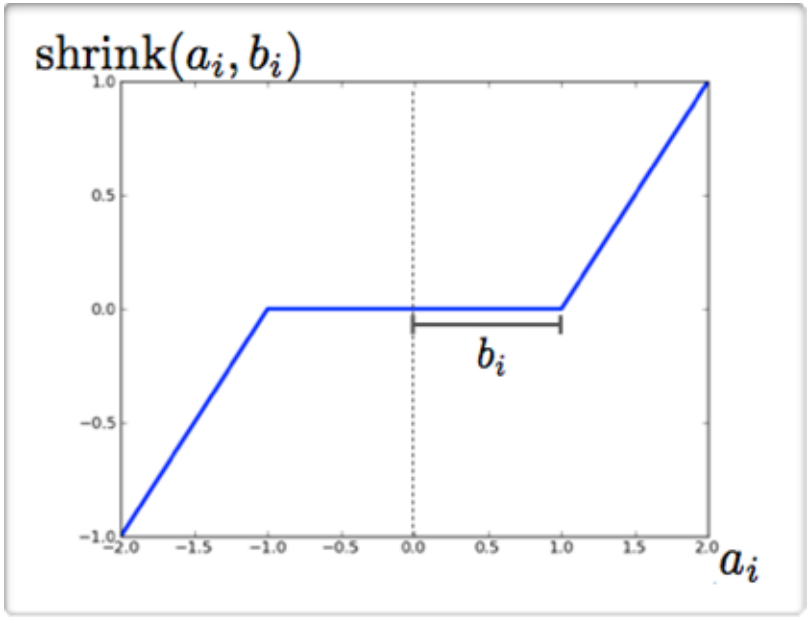
ISTA Algorithm

• This process corresponds to the ISTA (Iterative Shrinkage and Thresholding) Algorithm:

- Initialize $\mathbf{h}^{(t)}$ (for example to 0)
- While $\mathbf{h}^{(t)}$ has not converged

$$\mathbf{h}^{(t)} \Longleftarrow \mathbf{h}^{(t)} - \alpha\, \mathbf{D}^{\top}(\mathbf{D}\, \mathbf{h}^{(t)} - \mathbf{x}^{(t)})$$

$$\mathbf{h}^{(t)} \Longleftarrow \text{shrink}(\mathbf{h}^{(t)}, \alpha\, \lambda)$$



where

$$\text{shrink}(\mathbf{a}, \mathbf{b}) = [\ldots, \text{sign}(a_i)\ \max(|a_i| - b_i, 0), \ldots]$$

• Will converge if $\frac{1}{\alpha}$ is bigger than the largest eigenvalue of $\mathbf{D}^{\top}\mathbf{D}$

Dictionary Learning I

- Remember our optimization problem:

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} l(\mathbf{x}^{(t)}) = \min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}(\mathbf{x}^{(t)})||_2^2 + \lambda ||\mathbf{h}(\mathbf{x}^{(t)})||_1$$

- Let us first assume that $\mathbf{h}(\mathbf{x}^{(t)})$ does not depend on D
  - We then minimize:

$$\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{2} ||\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}(\mathbf{x}^{(t)})||_2^2$$

  - we must also constrain the columns of D to be of unit norm

column norm constraints: ‖D:,j‖2=1, ∀j

Dictionary Learning I

- We can use projected gradient descent algorithm.

- While D has not converged:
  - Perform gradient update of D

$$\mathbf{D} \Longleftarrow \mathbf{D} + \alpha \frac{1}{T} \sum_{t=1}^{T} (\mathbf{x}^{(t)} - \mathbf{D}\, \mathbf{h}(\mathbf{x}^{(t)}))\, \mathbf{h}(\mathbf{x}^{(t)})^{\top}$$

  - Renormalize the columns of D
  - For each column of D:

$$\mathbf{D}_{\cdot,j} \Longleftarrow \frac{\mathbf{D}_{\cdot,j}}{||\mathbf{D}_{\cdot,j}||_2}$$

Dictionary Learning II

- An alternative method is to solve for each column $\mathbf{D}_{\cdot,j}$ in cycle.
  - setting the gradient for $\mathbf{D}_{\cdot,j}$ to zero, we have

$$
\begin{aligned}
0 &= \frac{1}{T}\sum_{t=1}^{T}(\mathbf{x}^{(t)} - \mathbf{D}\,\mathbf{h}(\mathbf{x}^{(t)}))\,h(\mathbf{x}^{(t)})_j \\
0 &= \sum_{t=1}^{T}\left(\mathbf{x}^{(t)} - \left(\sum_{i\neq j}\mathbf{D}_{\cdot,i}\,h(\mathbf{x}^{(t)})_i\right) - \mathbf{D}_{\cdot,j}\,h(\mathbf{x}^{(t)})_j\right)\,h(\mathbf{x}^{(t)})_j \\
\sum_{t=1}^{T}\mathbf{D}_{\cdot,j}h(\mathbf{x}^{(t)})_j^2 &= \sum_{t=1}^{T}\left(\mathbf{x}^{(t)} - \left(\sum_{i\neq j}\mathbf{D}_{\cdot,i}\,h(\mathbf{x}^{(t)})_i\right)\right)\,h(\mathbf{x}^{(t)})_j \\
\mathbf{D}_{\cdot,j} &= \frac{1}{\sum_{t=1}^{T}h(\mathbf{x}^{(t)})_j^2}\sum_{t=1}^{T}\left(\mathbf{x}^{(t)} - \left(\sum_{i\neq j}\mathbf{D}_{\cdot,i}\,h(\mathbf{x}^{(t)})_i\right)\right)\,h(\mathbf{x}^{(t)})_j
\end{aligned}
$$

  - Note that we don't need to specify a learning rate to update D.

Dictionary Learning II

- An alternative method is to solve for each column $\mathbf{D}_{\cdot,j}$ in cycle.
  - We can rewrite

$$\begin{aligned}
\mathbf{D}_{\cdot,j} &= \frac{1}{\sum_{t=1}^{T} h(\mathbf{x}^{(t)})_j^2} \sum_{t=1}^{T} \left(\mathbf{x}^{(t)} - \left(\sum_{i \neq j} \mathbf{D}_{\cdot,i}\, h(\mathbf{x}^{(t)})_i \right) \right) h(\mathbf{x}^{(t)})_j \\
&= \frac{1}{\sum_{t=1}^{T} h(\mathbf{x}^{(t)})_j^2} \left(\left(\sum_{t=1}^{T} \mathbf{x}^{(t)} h(\mathbf{x}^{(t)})_j \right) - \sum_{i \neq j} \mathbf{D}_{\cdot,i} \left(\sum_{t=1}^{T} h(\mathbf{x}^{(t)})_i h(\mathbf{x}^{(t)})_j \right) \right)
\end{aligned}$$

$$= \frac{1}{A_{j,j}} (\mathbf{B}_{\cdot,j} - \mathbf{D}\ \mathbf{A}_{\cdot,j} + \mathbf{D}_{\cdot,j} A_{j,j})$$

  - this way, we only need to store:

$$\mathbf{A} \Longleftarrow \sum_{t=1}^{T} \mathbf{h}(\mathbf{x}^{(t)})\, \mathbf{h}(\mathbf{x}^{(t)})^\top$$

$$\mathbf{B} \Longleftarrow \sum_{t=1}^{T} \mathbf{x}^{(t)}\, \mathbf{h}(\mathbf{x}^{(t)})^\top$$

Dictionary Learning II

- This leads to the following algorithm

  - While D has not converged:
    - for each column $\mathbf{D}_{.,j}$ perform updates

$$\mathbf{D}_{.,j} \Longleftarrow \frac{1}{A_{j,j}} \left(\mathbf{B}_{.,j} - \mathbf{D}\ \mathbf{A}_{.,j} + \mathbf{D}_{.,j}\ A_{j,j}\right)$$

$$\mathbf{D}_{.,j} \Longleftarrow \frac{\mathbf{D}_{.,j}}{||\mathbf{D}_{.,j}||_2}$$

- This is referred to as a block-coordinate descent algorithm
  - a different block of variables are updated at each step
  - the "blocks" are the columns $\mathbf{D}_{.,j}$

Learning Sparse Coding Model

• Putting it all together, we have the following algorithm, where learning alternates between inference and dictionary learning.

• While D has not converged:

- find the sparse codes $\mathbf{h}(\mathbf{x}^{(t)})$ for all $\mathbf{x}^{(t)}$ in the training set with ISTA
- Update the dictionary:

$$\mathbf{A} \Longleftarrow \sum_{t=1}^{T} \mathbf{x}^{(t)}\ \mathbf{h}(\mathbf{x}^{(t)})^{\top}$$

$$\mathbf{B} \Longleftarrow \sum_{t=1}^{T} \mathbf{h}(\mathbf{x}^{(t)})\ \mathbf{h}(\mathbf{x}^{(t)})^{\top}$$

$$\mathbf{D}_{\cdot,j} \Longleftarrow \frac{1}{A_{j,j}} \left(\mathbf{B}_{\cdot,j} - \mathbf{D}\ \mathbf{A}_{\cdot,j} + \mathbf{D}_{\cdot,j}\ A_{j,j}\right)$$

$$\mathbf{D}_{\cdot,j} \Longleftarrow \frac{\mathbf{D}_{\cdot,j}}{||\mathbf{D}_{\cdot,j}||_2}$$

Instead of computing A and B from scratch over all data, we maintain running averages that update with each new sample.

Online Learning

- This algorithm is "batch" (i.e. not online)
  - single update of the dictionary per pass on the training set
  - for large datasets, we'd like to update D after visiting each $\mathbf{x}^{(t)}$
- Solution: for each input $\mathbf{x}^{(t)}$
  - perform inference of $\mathbf{h}(\mathbf{x}^{(t)})$ for the current input
  - update running averages of the quantities required to update D:

$$\mathbf{B} \Longleftarrow \beta\, \mathbf{B} + (1-\beta)\, \mathbf{x}^{(t)}\, \mathbf{h}(\mathbf{x}^{(t)})^\top$$

$$\mathbf{A} \Longleftarrow \beta\, \mathbf{A} + (1-\beta)\, \mathbf{h}(\mathbf{x}^{(t)})\, \mathbf{h}(\mathbf{x}^{(t)})^\top$$

Online Learning

- While D has not converged:
  - For each $\mathbf{x}^{(t)}$
    - Infer code $\mathbf{h}(\mathbf{x}^{(t)})$
    - Update the dictionary:

$$\mathbf{A} \Longleftarrow \beta\ \mathbf{A} + (1-\beta)\ \mathbf{h}(\mathbf{x}^{(T+1)})\ \mathbf{h}(\mathbf{x}^{(T+1)})^\top$$

$$\mathbf{B} \Longleftarrow \beta\ \mathbf{B} + (1-\beta)\ \mathbf{x}^{(t)}\ \mathbf{h}(\mathbf{x}^{(t)})^\top$$

    - while D hasn't converged
      - for each column of D perform gradient update

$$\mathbf{D}_{\cdot,j} \Longleftarrow \frac{1}{A_{j,j}}\left(\mathbf{B}_{\cdot,j} - \mathbf{D}\ \mathbf{A}_{\cdot,j} + \mathbf{D}_{\cdot,j}\ A_{j,j}\right)$$

$$\mathbf{D}_{\cdot,j} \Longleftarrow \frac{\mathbf{D}_{\cdot,j}}{||\mathbf{D}_{\cdot,j}||_2}$$

Online Dictionary Learning for Sparse Coding. Mairal, Bach, Ponce and Sapiro, 2009.

Whitening

- Before running a sparse coding algorithm, it is beneficial to remove "obvious" structure from the data
  - normalize such that mean is 0 and covariance is the identity (whitening)
  - this will remove 1st and 2nd order statistical structure

- preprocessing
  - let the empirical mean be $\widehat{\boldsymbol{\mu}}$ and the empirical covariance matrix be $\widehat{\Sigma} = \mathbf{U}\Lambda\mathbf{U}^\top$(in its eigenvalue/eigenvector representation)
  - ZCA transforms each input as follows:

$$\mathbf{x} \Longleftarrow \mathbf{U}\ \Lambda^{-\frac{1}{2}}\ \mathbf{U}^\top(\mathbf{x} - \widehat{\boldsymbol{\mu}})$$

Whitening

- After this transformation
  - the empirical mean is 0

$$
\begin{aligned}
& \frac{1}{T}\sum_t \mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top(\mathbf{x}^{(t)}-\widehat{\boldsymbol{\mu}}) \\
= \quad & \mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\left(\left(\frac{1}{T}\sum_t \mathbf{x}^{(t)}\right)-\widehat{\boldsymbol{\mu}}\right) \\
= \quad & \mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top(\widehat{\boldsymbol{\mu}}-\widehat{\boldsymbol{\mu}}) \\
= \quad & 0
\end{aligned}
$$

Whitening

- After this transformation
  - the empirical covariance matrix is the identity

$$\frac{1}{T-1}\sum_t \left(\mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top(\mathbf{x}^{(t)}-\widehat{\boldsymbol{\mu}})\right)\left(\mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top(\mathbf{x}^{(t)}-\widehat{\boldsymbol{\mu}})\right)^\top$$

$$\begin{aligned}
&= \mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\left(\frac{1}{T-1}\sum_t(\mathbf{x}^{(t)}-\widehat{\boldsymbol{\mu}})(\mathbf{x}^{(t)}-\widehat{\boldsymbol{\mu}})\right)^\top\mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\\
&= \mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\,\widehat{\Sigma}\,\mathbf{U}\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\\
&= \mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\,\mathbf{U}\,\Lambda\,\mathbf{U}^\top\,\mathbf{U}\,\Lambda^{-\frac{1}{2}}\,\mathbf{U}^\top\\
&= \mathbf{I}
\end{aligned}$$

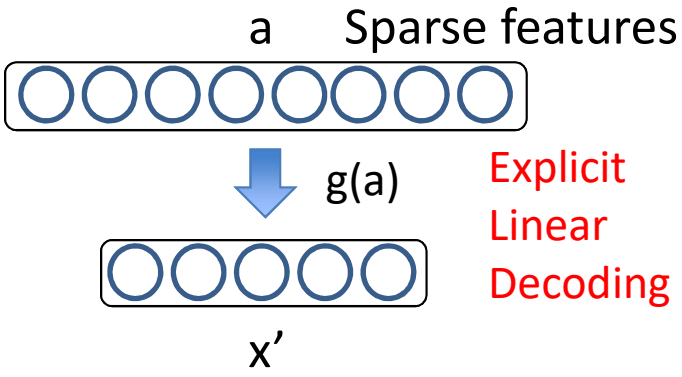
Feature Learning

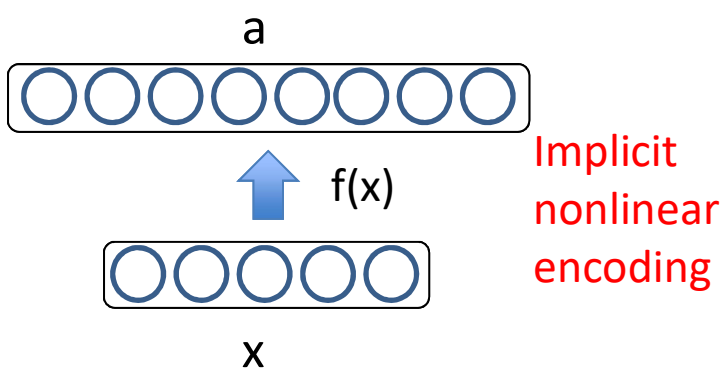
- A sparse coding model can be used to extract features
  - given a labeled training set $\{(\mathbf{x}^{(t)}, y^{(t)})\}$
  - train sparse coding dictionary only on training inputs $\{\mathbf{x}^{(t)}\}$
  - this yields a dictionary from which to infer sparse codes
  - train your favorite classifier on transformed training set $\{(\mathbf{h}(\mathbf{x}^{(t)}), y^{(t)})\}$

- When classifying test input x
  - infer its sparse representation: $\mathbf{h}(\mathbf{x})$
  - feed it to the classifier

Interpreting Sparse Coding

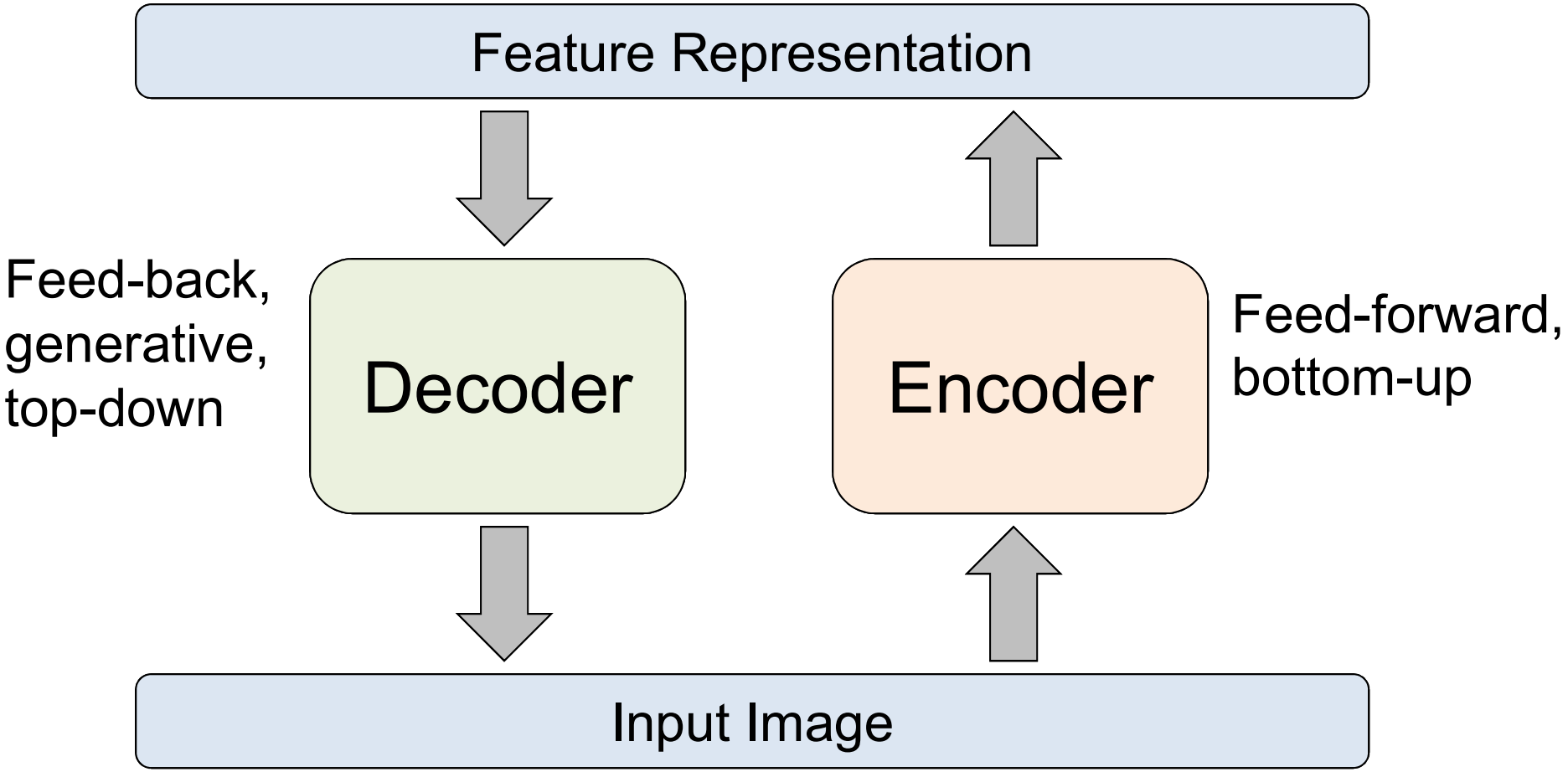
$$\min_{\mathbf{a},\phi} \sum_{n=1}^{N} \left|\left| \mathbf{x}_n - \sum_{k=1}^{K} a_{nk}\phi_k \right|\right|_2^2 + \lambda \sum_{n=1}^{N}\sum_{k=1}^{K} |a_{nk}|$$





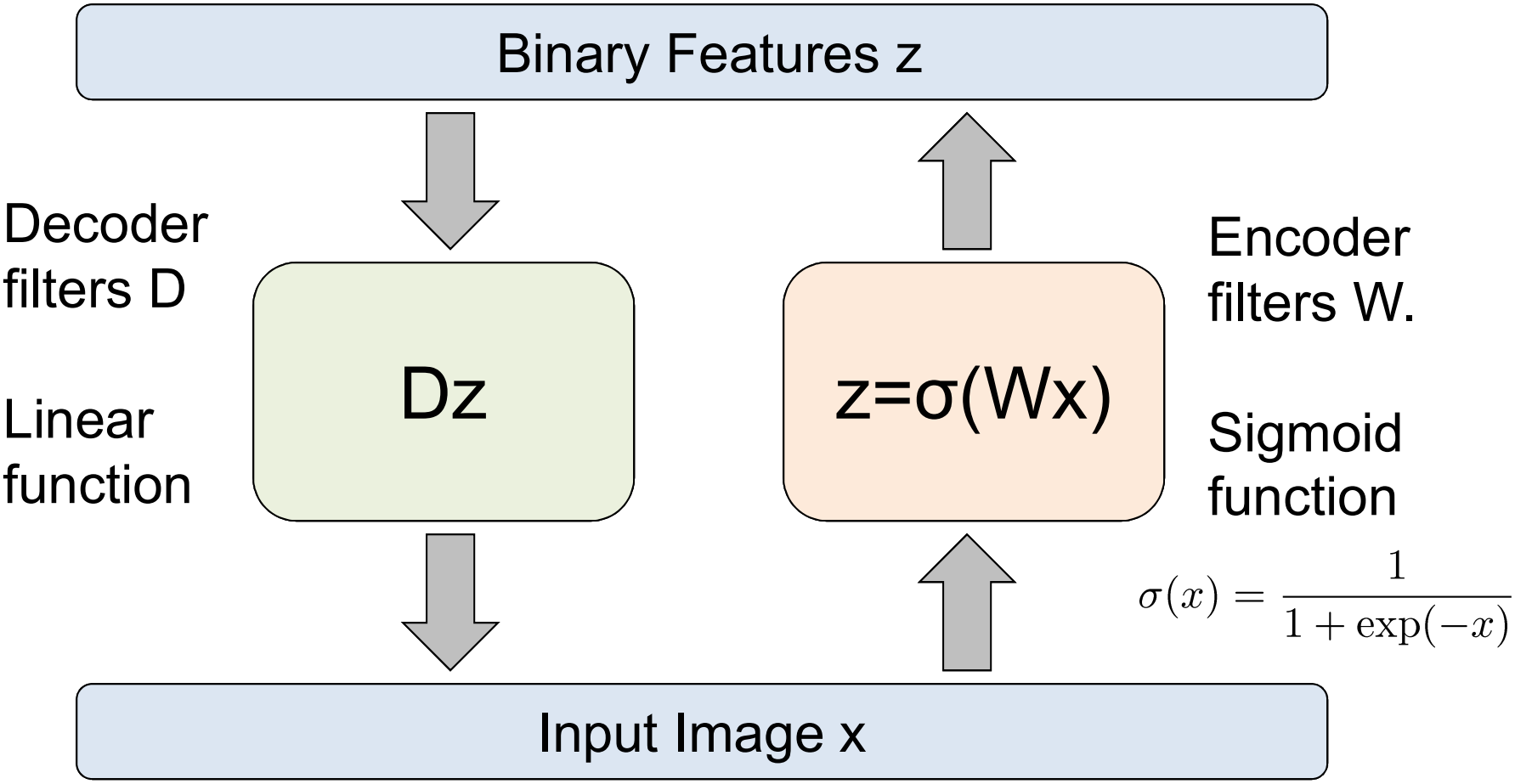
- Sparse, over-complete representation **a.**
- Encoding **a** = f(**x**) is implicit and nonlinear function of **x**.
- Reconstruction (or decoding) **x'** = g(**a**) is linear and explicit.

Autoencoder

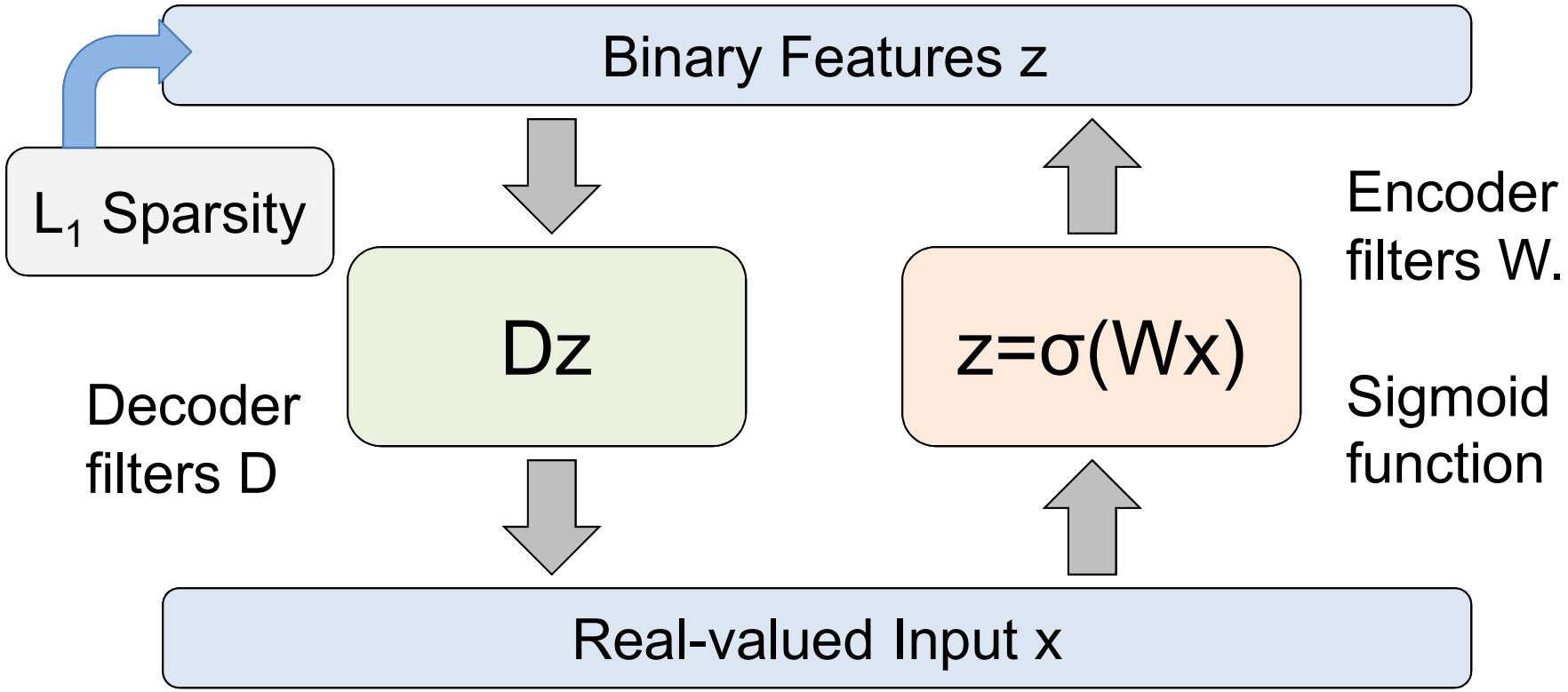


- Details of what goes insider the encoder and decoder matter!
- Need constraints to avoid learning an identity.

Autoencoder



Predictive Sparse Decomposition

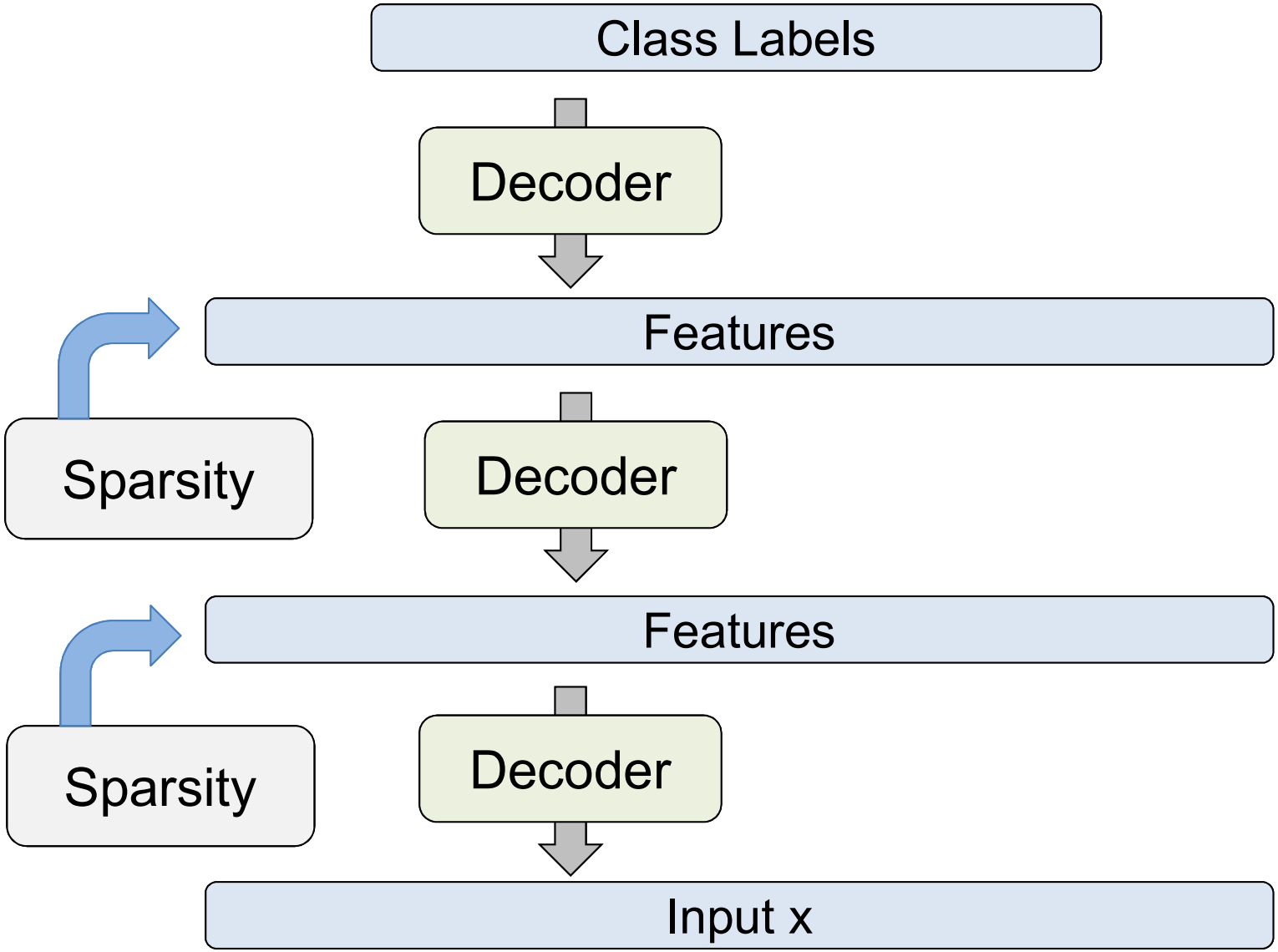


At training time

$$\min_{D,W,\mathbf{z}} \underbrace{||D\mathbf{z}-\mathbf{x}||_2^2+\lambda|\mathbf{z}|_1}_{\text{Decoder}} + \underbrace{||\sigma(W\mathbf{x})-\mathbf{z}||_2^2}_{\text{Encoder}}$$

Kavukcuoglu et al., '09

Stacked Sparse Coding?



Graphical Models

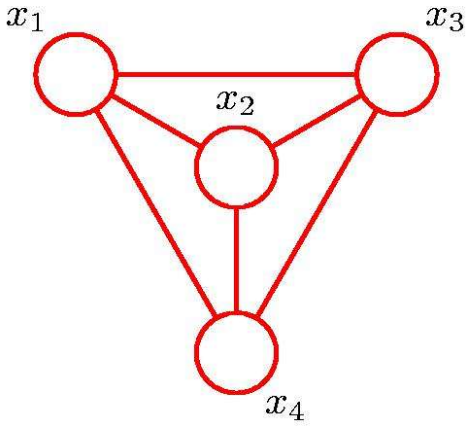
Vahid Tarokh
ECE 685D, Fall 2025

Graphical Models

- Probabilistic graphical models provide a powerful framework for representing dependency structure between random variables.
- Graphical models offer several useful properties:
  - They provide a simple way to visualize the structure of a probabilistic model and can be used to motivate new models.
  - They provide various insights into the properties of the model, including conditional independence.
  - Complex computations (e.g. inference and learning in sophisticated models) can be expressed in terms of graphical manipulations.

Graphical Models

- A graph contains a set of nodes (vertices) connected by links (edges or arcs)



- In a probabilistic graphical model, each node represents a random variable, and links represent probabilistic dependencies between random variables.
- The graph specifies the way in which the joint distribution over all random variables decomposes into a product of factors, where each factor depends on a subset of the variables.

- Types of graphical models:
  - Bayesian networks, also known as Directed Graphical Models (the links have a particular directionality indicated by the arrows)
  - Markov Random Fields, also known as Undirected Graphical Models (the links do not carry arrows and have no directional significance).

- Hybrid graphical models that combine directed and undirected graphical models, such as Deep Belief Networks.

Bayesian Networks

- Directed Graphs are useful for expressing causal relationships between random variables.
- We consider an arbitrary joint distribution $p(a, b, c)$ over three random variables a, b and c.
- Note that at this point, we do not need to specify anything else about these variables
- By application of the product rule of probability (twice):

$$p(a, b, c) = p(c|a, b)p(a, b) = p(c|a, b)p(b|a)p(a)$$

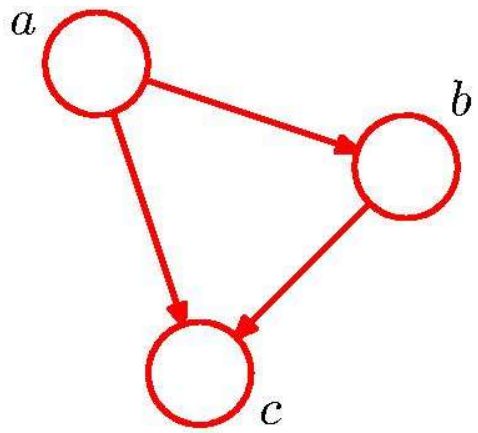
- This decomposition holds for any choice of the joint distribution.

Bayesian Networks

• By application of the product rule of probability (twice), we get

$$p(a,b,c) = p(c|a,b)p(a,b) = p(c|a,b)p(b|a)p(a)$$

• Represent the joint distribution in terms of a simple graphical model:



- Introduce a node for each of the random variables.
- Associate each node with the corresponding conditional distribution in above equation.
- For each conditional distribution we add directed links to the graph from the nodes corresponding to the variables on which the distribution is conditioned.

• Hence for the factor $p(c|a,b)$, there will be links from nodes a and b to node c.

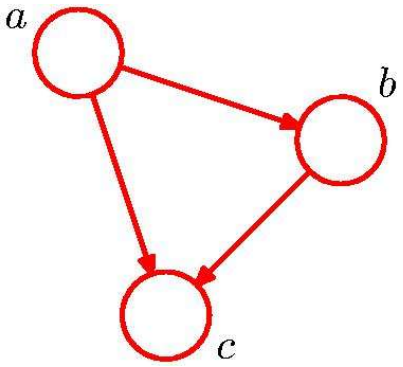
• For the factor $p(a)$, there will be no incoming links.

Bayesian Networks

• By application of the product rule of probability (twice), we get

$$p(a,b,c) = p(c|a,b)p(a,b) = p(c|a,b)p(b|a)p(a)$$

• If there is a link going from node a to node b, then we say that:



- node a is a parent of node b.
- node b is a child of node a.

• For the decomposition, we choose a specific ordering of the random variables: a,b,c.

• If we chose a different ordering, we would get a different graphical representation (we will come back to that point later).

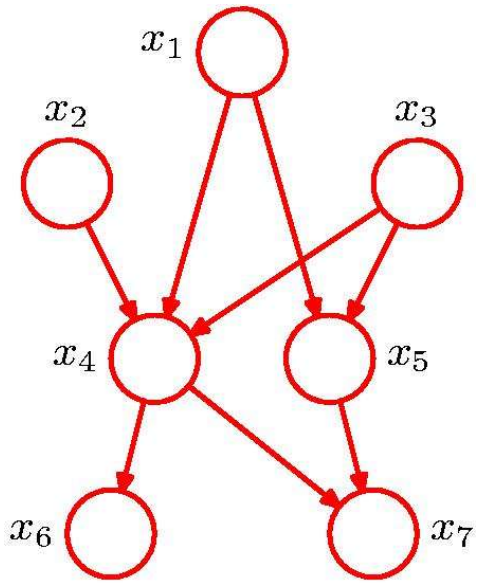
• The joint distribution over K variables factorizes:

$$p(x_1,\ldots,x_K) = p(x_K|x_1,\ldots,x_{K-1})\ldots p(x_2|x_1)p(x_1)$$

• If each node has incoming links from all lower numbered nodes, then the graph is fully connected; there is a link between all pairs of nodes.

Bayesian Networks

• Absence of links conveys certain information about the properties of the class of distributions that the graph conveys.



• Note that this graph is not fully connected (e.g. there is no link from x_1 to x_2).

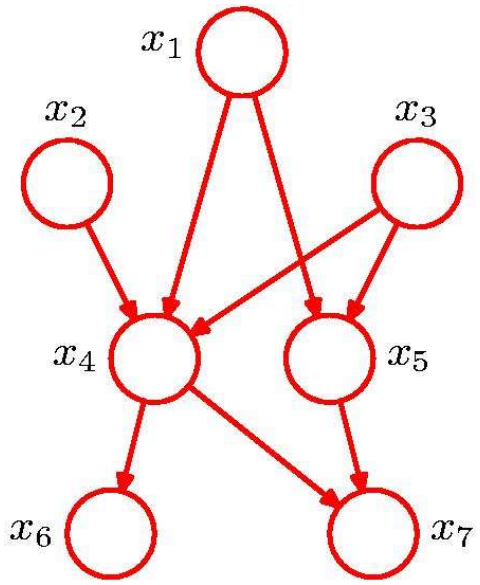
• The joint distribution over $x_1,\ldots,x_7$ can be written as a product of a set of conditional distributions.

$$p(x_1,\ldots,x_7) = p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)$$
$$p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$$

• Note that according to the graph, x_5 will be conditioned only on x_1 and x_3.

Factorization Property

• The joint distribution defined by the graph is given by the product of a conditional distribution for each node conditioned on its parents:



$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$

• Where pa_k denotes a set of parents for the node x_k.

• This equation expresses a key factorization property of the joint distribution for a ***directed*** graphical model.

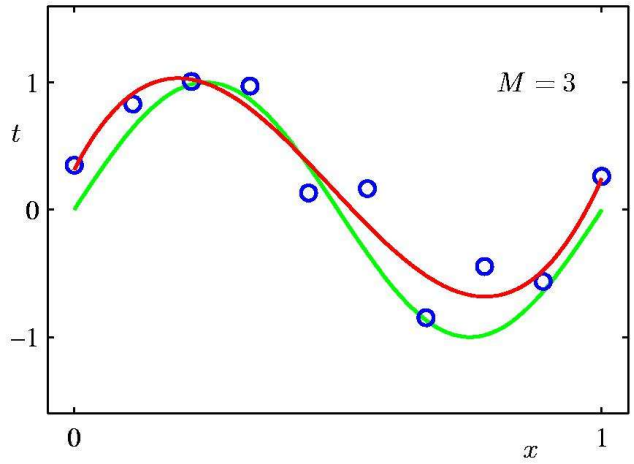
• Important restriction: There must be **no directed cycles!**

• Such graphs are also called directed acyclic graphs (DAGs).

Bayesian Curve Fitting

• As an example, consider Bayesian polynomial regression model:

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j$$



• We are given inputs $\mathbf{X} = \{x_1, x_2, ..., x_N\}$ and target values $\mathbf{t} = [t_1, t_2, ..., t_N]^T$.

• Given the prior over parameters, the joint distribution is given by:

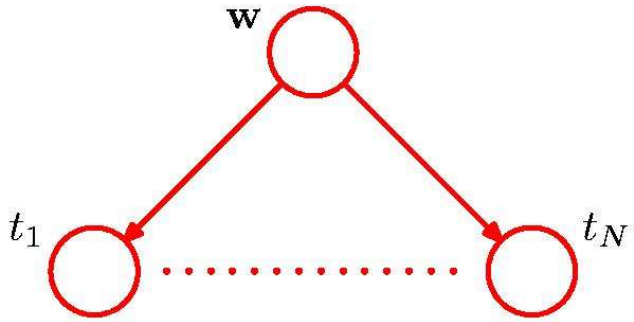
$$p(\mathbf{t}, \mathbf{w}|\mathbf{X}) = p(\mathbf{w}) \prod_{n=1}^{N} p(t_n|y(\mathbf{w}, x_n)).$$

Prior term Likelihood term

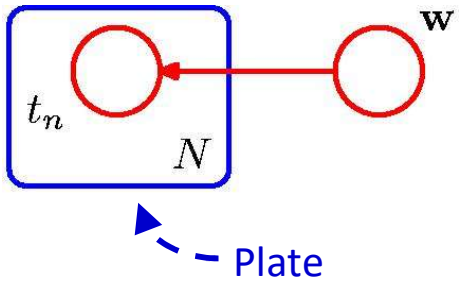
Graphical Representation

$$p(\mathbf{t}, \mathbf{w}|\mathbf{X}) = p(\mathbf{w}) \prod_{n=1}^{N} p(t_n|y(\mathbf{w}, x_n)).$$

• This distribution can be represented as a graphical model.



• Same representation using plate notation.



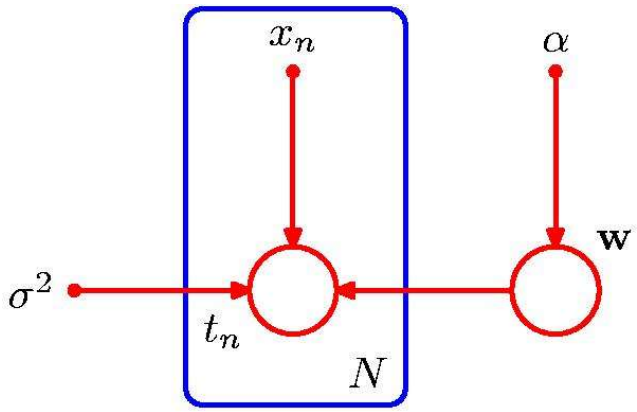
• Compact representation: we introduce a plate that represents N nodes of which only a single example t_n is shown explicitly.

• Note that $\boldsymbol{w}$ and $\mathbf{t} = [t_1, t_2, ..., t_N]^T$ represent random variables.

Graphical Representation

• It will often be useful to make the parameters of the model as well as random variables be explicit.

$$p(\mathbf{t}, \mathbf{w}|\mathbf{x}, \alpha, \sigma^2) = p(\mathbf{w}|\alpha) \prod_{n=1}^{N} p(t_n|\mathbf{w}, x_n, \sigma^2).$$



$$p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha \boldsymbol{I}),$$

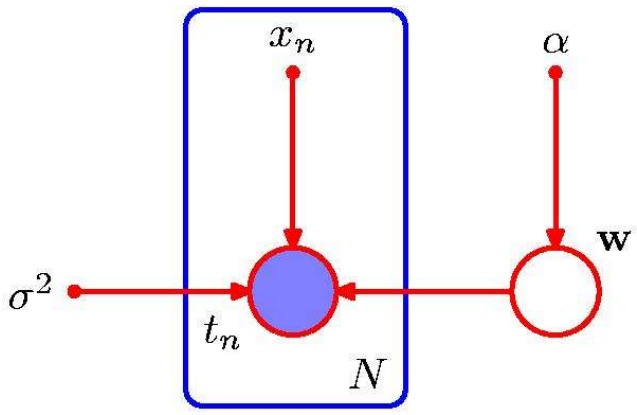
$$p(t_n|\mathbf{w}, x_n, \sigma^2) = \mathcal{N}(t_n|y(\mathbf{w}, x_n), \sigma^2),$$

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j x^j$$

• Random variables will be denoted by **open circles** and deterministic parameters will be denoted by **smaller solid circles**.

Graphical Representation

• When we apply a graphical model to a problem in machine learning, we will set some of the variables to specific observed values (e.g. condition on the data).



$$p(\mathbf{w}|\mathbf{t}) \propto p(\mathbf{w}) \prod_{n=1}^{N} p(t_n|\mathbf{w})$$

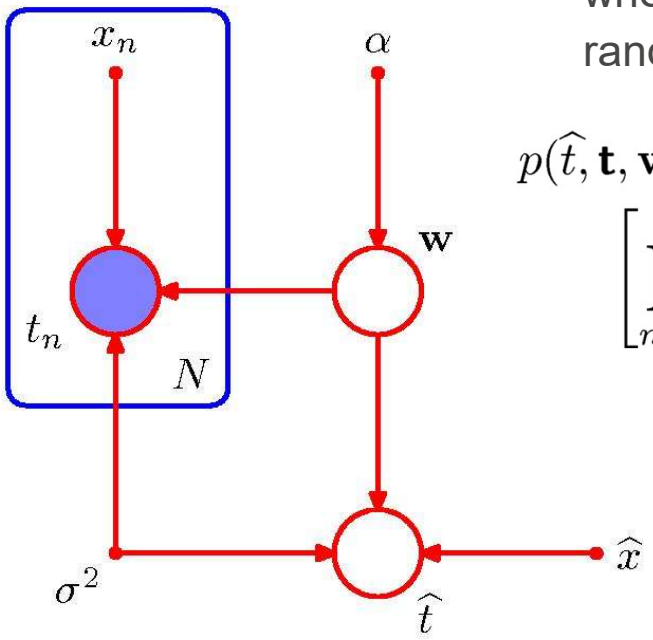
• For example, having observed the values of the targets $\{t_n\}$ on the training data, we wish to infer the posterior distribution over parameters w.

• In this example, we conditioned on observed data $\mathbf{t} = [t_1, t_2, ..., t_N]^T$ by shadowing the corresponding nodes.

Predictive Distribution

• We may also be interested in making predictions for a new input value $\widehat{x}$.

$$p(\widehat{t}|\widehat{x}, \mathbf{x}, \mathbf{t}, \alpha, \sigma^2) \propto \int p(\widehat{t}, \mathbf{t}, \mathbf{w}|\widehat{x}, \mathbf{x}, \alpha, \sigma^2)\, \mathrm{d}\mathbf{w}$$



- where the joint distribution of all the random variables is given by:

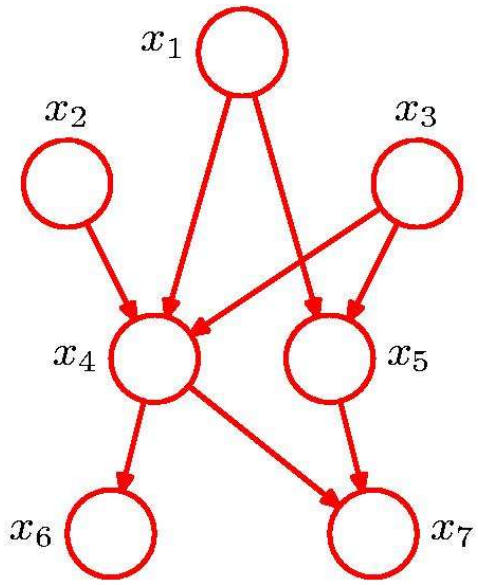
$$p(\widehat{t}, \mathbf{t}, \mathbf{w}|\widehat{x}, \mathbf{x}, \alpha, \sigma^2) = \left[\prod_{n=1}^{N} p(t_n|x_n, \mathbf{w}, \sigma^2)\right] p(\mathbf{w}|\alpha) p(\widehat{t}|\widehat{x}, \mathbf{w}, \sigma^2)$$

• Here we are setting the random variables in **t** to the specific values observed in the data.

Ancestral Sampling

• Consider a joint distribution over K random variables $p(x_1, x_2, ..., x_K)$ that factorizes as:

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|\mathrm{pa}_k)$$



• Our goal is draw a sample from this distribution.

• Start at the top and sample in order.

$$\hat{x}_1 \sim p(x_1)$$
$$\hat{x}_2 \sim p(x_2)$$
$$\hat{x}_3 \sim p(x_3)$$
$$\hat{x}_4 \sim p(x_4|\hat{x}_1, \hat{x}_2, \hat{x}_3)$$
$$\hat{x}_5 \sim p(x_5|\hat{x}_1, \hat{x}_3)$$

The parent variables are set to their sampled values

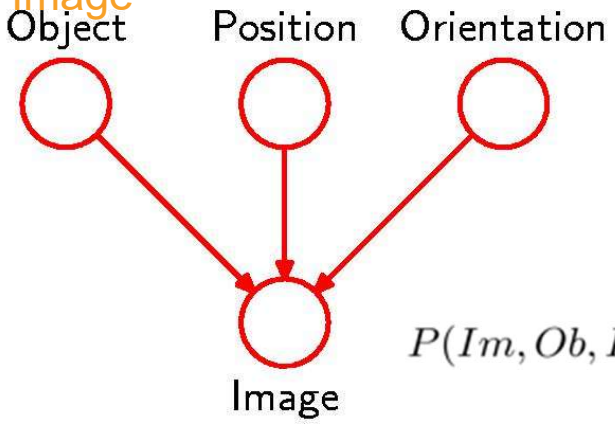
• To obtain a sample from the marginal distribution, e.g. $p(x_2, x_5)$, we sample from the full joint distribution, retain $\hat{x}_2, \hat{x}_5$, and discard the remaining values.

Generative Models

• Higher-level nodes will typically represent latent (hidden) random variables.

• The primary role of the latent variables is to allow a complicated distribution over observed variables to be constructed from simpler (typically exponential family) conditional distributions.

Generative Model of an Image



• Object identity, position, and orientation have independent prior probabilities.

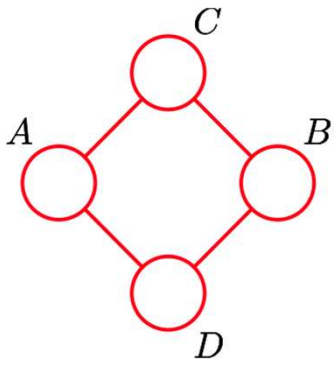
• The image has a probability distribution that depends on the object identity, position, and orientation (likelihood function).

$$P(Im, Ob, Po, Or) = \underbrace{P(Im|Ob, Po, Or)}_{\text{Likelihood}} \underbrace{P(Ob)P(Po)P(Or)}_{\text{Prior}}$$

• The graphical model captures the causal process, by which the observed data was generated (hence the name generative models).

Undirected Graphical Models

Directed graphs are useful for expressing causal relationships between random variables, whereas undirected graphs are useful for expressing soft constraints between random variables



• The joint distribution defined by the graph is given by the product of non-negative potential functions over the maximal cliques (connected subset of nodes).

$$p(\mathbf{x}) = \frac{1}{\mathcal{Z}} \prod_C \phi_C(x_C) \qquad \mathcal{Z} = \sum_{\mathbf{x}} \prod_C \phi_C(x_C)$$

where the normalizing constant $\mathcal{Z}$ is called a partition function.

• For example, the joint distribution factorizes:

$$p(A, B, C, D) = \frac{1}{\mathcal{Z}} \phi(A, C)\phi(C, B)\phi(B, D)\phi(A, D)$$

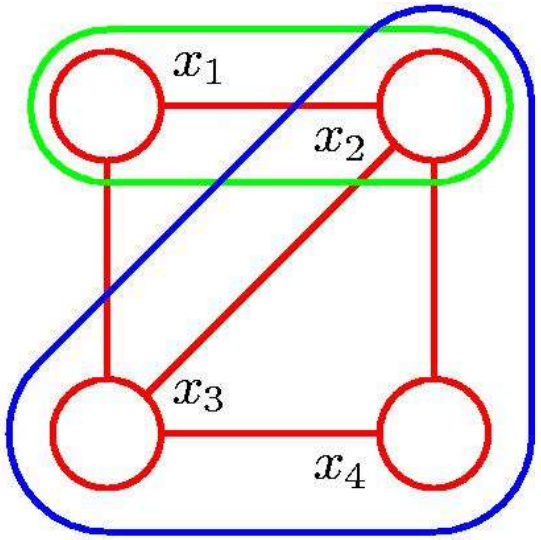
• Let us look at the definition of cliques.

Cliques

• The subsets that are used to define the potential functions are represented by maximal cliques in the undirected graph.

• Clique: a subset of nodes such that there exists a link between all pairs of nodes in a subset.

• Maximal Clique: a clique such that it is not possible to include any other nodes in the set without it ceasing to be a clique.



• This graph has 5 cliques:

$\{x_1, x_2\}, \{x_2, x_3\}, \{x_3, x_4\},$
$\{x_4, x_2\}, \{x_1, x_3\}.$

• Two maximal cliques:

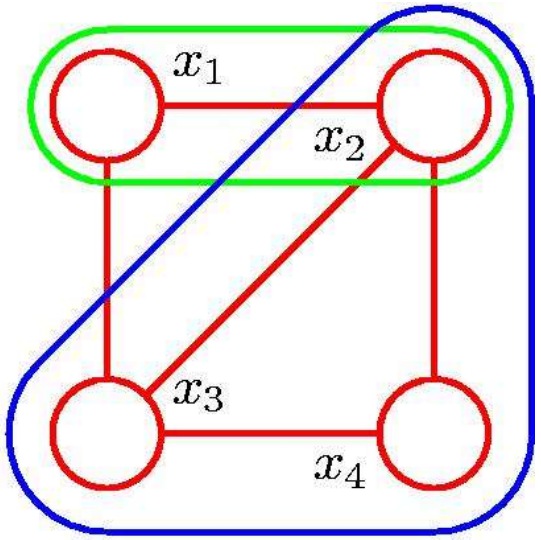
$\{x_1, x_2, x_3\}, \{x_2, x_3, x_4\}.$

Using Cliques to Represent Subsets

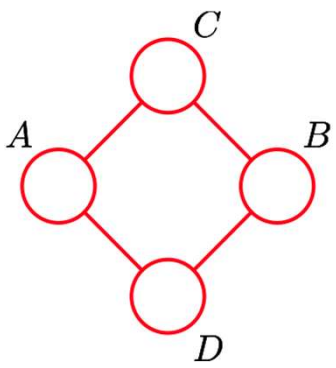
• If the potential functions only involve two nodes, an undirected graph has a nice representation.

• If the potential functions involve more than two nodes, using a different factor graph representation is much more useful.

• For now, let us consider only potential functions that are defined over two nodes.



Markov Random Fields (MRFs)



$$p(\mathbf{x}) = \frac{1}{\mathcal{Z}} \prod_C \phi_C(x_C)$$

- Each potential function is a mapping from the joint configurations of random variables in a clique to non-negative real numbers.
- The choice of potential functions is not restricted to having specific probabilistic interpretations.

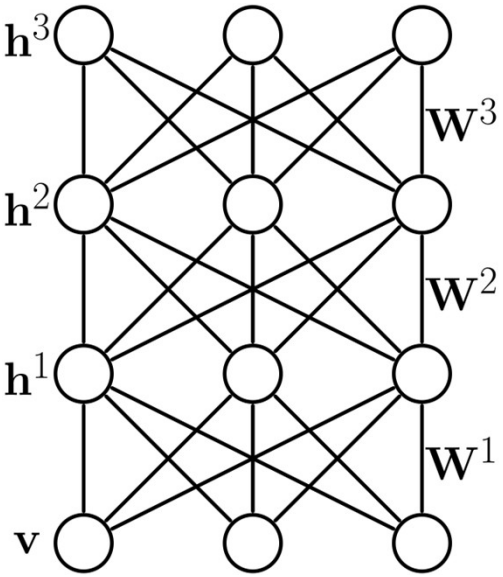
Potential functions are often represented as exponentials:

$$p(\mathbf{x}) = \frac{1}{\mathcal{Z}} \prod_C \phi_C(x_C) = \frac{1}{\mathcal{Z}} \exp(-\sum_C E(x_c)) = \underbrace{\frac{1}{\mathcal{Z}} \exp(-E(\mathbf{x}))}_{\text{Boltzmann distribution}}$$

where $\boldsymbol{E}(\boldsymbol{x})$ is called an energy function.

MRFs with Hidden Variables

For many interesting real-world problems, we need to introduce hidden or latent variables.



• Our random variables will contain both visible and hidden variables x=(v,h).

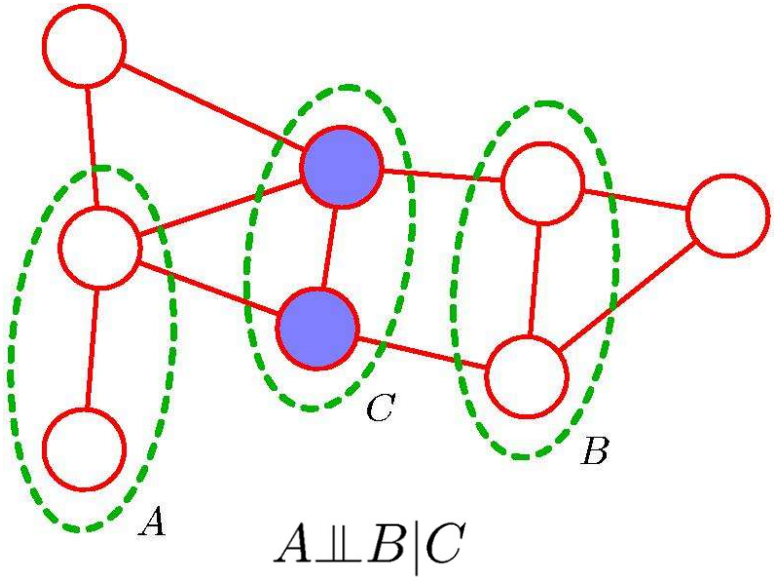
$$p(\mathbf{v}) = \frac{1}{\mathcal{Z}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$$

• In general, computing both partition function and summation over hidden variables will be intractable, except for special cases.

• Parameter learning becomes a very challenging task.

Conditional Independence

• Conditional Independence is easier compared to directed models:



$A \perp\!\!\!\perp B | C$

A is conditionally independent of B given C

• Observation blocks a node.

• Two sets of nodes are conditionally independent if the observations block all paths between them.

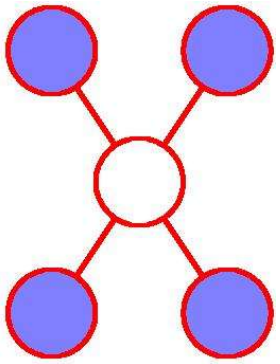
iff every path between A and B is blocked by nodes in C.

A node becomes a “blocker” when it is observed.

Markov Blanket

• The Markov blanket of a node is simply all the directly connected nodes.

Markov Blanket



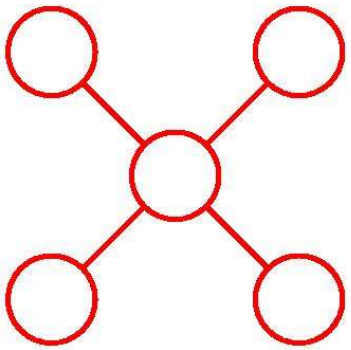
• This is simpler than in directed models, since there is no explaining away.

• The conditional distribution of x_i conditioned on all the variables in the graph is dependent only on the variables in the Markov blanket.

Markov blanket in MRF = the set of neighboring nodes

Conditional Independence and Factorization

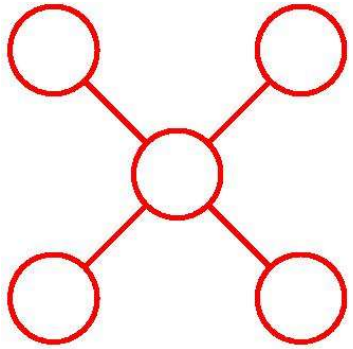
- Consider two sets of distributions:
  - The set of distributions consistent with the conditional independence relationships defined by the undirected graph.
  - The set of distributions consistent with the factorization defined by potential functions on maximal cliques of the graph.
- The Hammersley-Clifford theorem states that these two sets of distributions are the same.



$$p(\mathbf{x}) = \frac{1}{Z} \prod_{C} \phi_C(x_C)$$

Interpreting Potentials

• In contrast to directed graphs, the potential functions **do not have a specific probabilistic interpretation**.



$$p(\mathbf{x}) = \frac{1}{\mathcal{Z}} \prod_C \phi_C(x_C) = \frac{1}{\mathcal{Z}} \exp(-\sum_C E(x_c))$$

• This gives us greater flexibility in choosing the potential functions.

• We can view the potential function as expressing which configuration of the local variables are preferred to others.

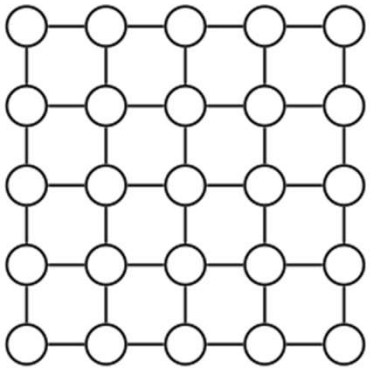
• Global configurations with relatively high probabilities are those that find a good balance in satisfying the (possibly conflicting) influences of the clique potentials.

• So far we did not specify the nature of random variables, discrete or continuous.

Discrete MRFs

• MRFs with all discrete variables are widely used in many applications.

• MRFs with binary variables are sometimes called Ising models in statistical mechanics, and Boltzmann machines in machine learning literature.



• Denoting the binary valued variable at node j by $x_j \in \{0, 1\}$, the Ising model for a graph $G(V, E)$ and for the joint probabilities is given by:

$$P_\theta(\mathbf{x}) = \frac{1}{\mathcal{Z}(\theta)} \exp\Big(\sum_{ij \in E} x_i x_j \theta_{ij} + \sum_{i \in V} x_i \theta_i \Big)$$

• The conditional distribution is given by logistic:

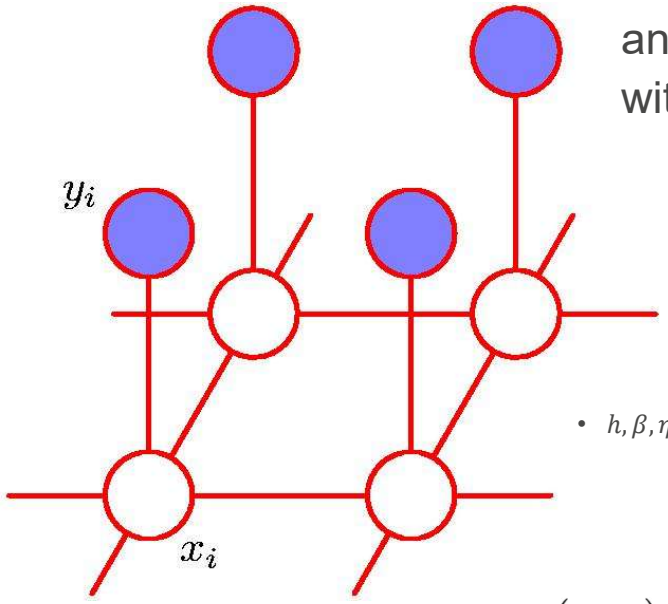
$$P_\theta(x_i = 1 | \mathbf{x}_{-i}) = \frac{1}{1 + \exp(-\theta_i - \sum_{ij \in E} x_j \theta_{ij})},$$

where x_{-i} denotes all nodes except for i.

Hence the parameter θ_{ij} measures the dependence of x_i on x_j, conditional on the other nodes.

Example: Image Denoising

- Let us look at the example of noise removal from a binary image.
- Let the observed noisy image be described by an array of binary pixel values: $y_j \in \{-1, +1\}$, i=1,…,D.
- We take a noise-free image $x_j \in \{-1, +1\}$, and randomly flip the sign of pixels with some small probability.



Bias term

Neighboring pixels are likely to have the same sign

$$E(\mathbf{x}, \mathbf{y}) = h \sum_i x_i - \beta \sum_{\{i,j\}} x_i x_j - \eta \sum_i x_i y_i$$

- h, β, η *are model parameters*

Noisy and clean pixels are likely to have the same sign

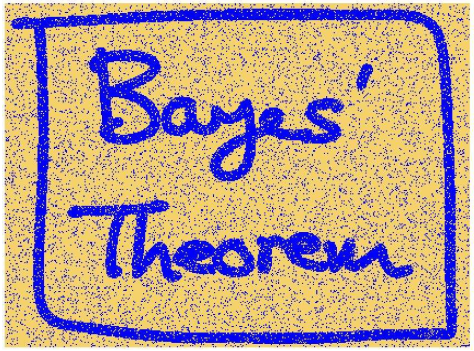
$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \exp\{-E(\mathbf{x}, \mathbf{y})\}$$

Iterated Conditional Modes

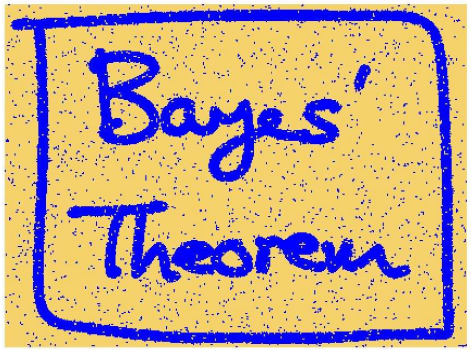
- Iterated conditional modes: coordinate-wise gradient descent.

- Visit the unobserved nodes sequentially and set each x to whichever of its two values has the lowest energy.
  - This only requires us to look at the Markov blanket, i.e. the connected nodes.
  - Markov blanket of a node is simply all the directly connected nodes.

Original Image



Noisy Image



ICM

Restricted Boltzmann Machines

Vahid Tarokh

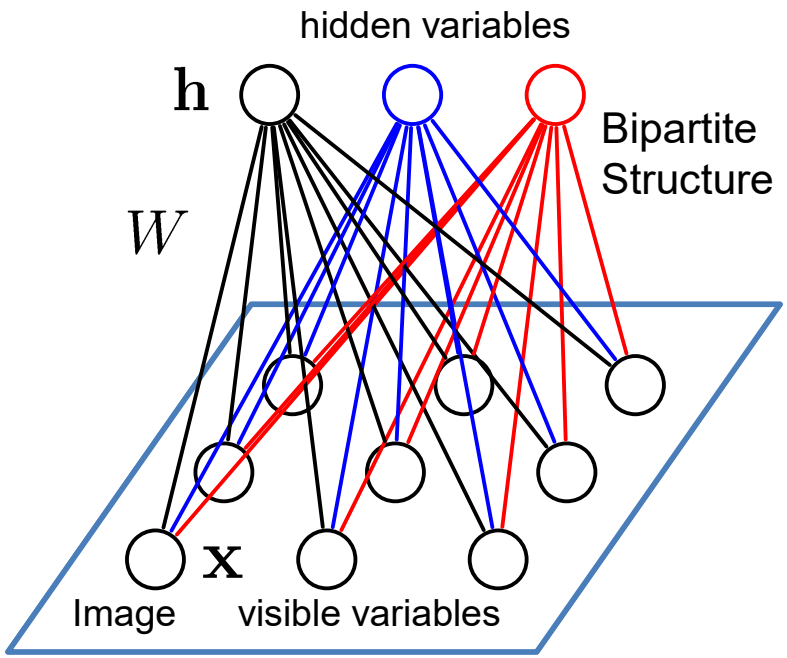
ECE 685D, Fall 2025

Unsupervised Learning

- Unsupervised learning: we only use the inputs $\mathbf{x}^{(t)}$ for learning
  - automatically extract meaningful features for your data
  - leverage the availability of unlabeled data

- We will consider 3 models for unsupervised learning that will form the basic building blocks for deeper models:
  - Autoencoders
  - Sparse coding models
  - Restricted Boltzmann Machines

Restricted Boltzmann Machines



- Undirected bipartite graphical model
- Stochastic binary visible variables:

$$\mathbf{x} \in \{\mathbf{0}, \mathbf{1}\}^{\mathbf{D}}$$

- Stochastic binary hidden variables:

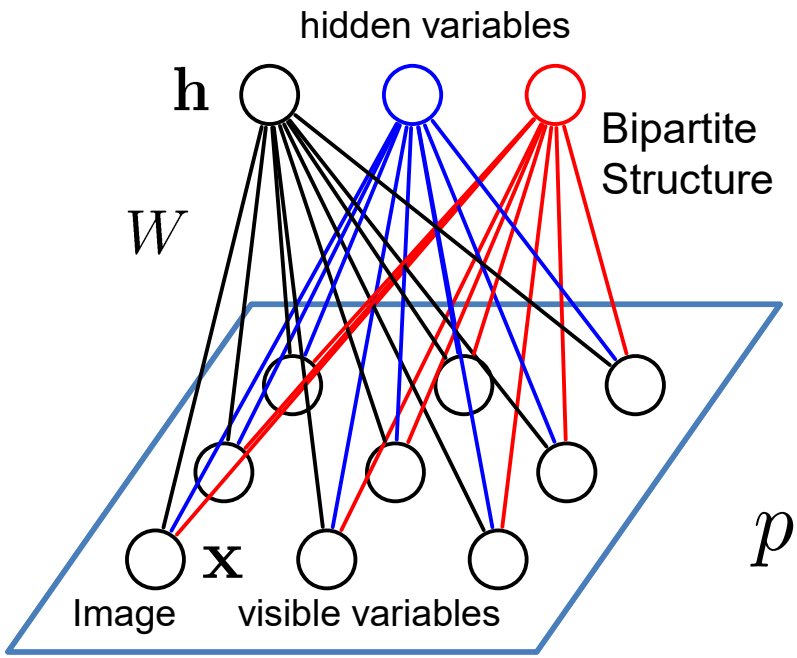
$$\mathbf{h} \in \{0, 1\}^{F}$$

- The energy of the joint configuration:

$$\begin{aligned} E(\mathbf{x}, \mathbf{h}) &= -\mathbf{h}^\top \mathbf{W}\mathbf{x} - \mathbf{c}^\top \mathbf{x} - \mathbf{b}^\top \mathbf{h} \\ &= -\sum_j \sum_k W_{j,k} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \end{aligned}$$

Markov random fields, Boltzmann machines, log-linear models.

Restricted Boltzmann Machines



• Probability of the joint configuration is given by the Boltzmann distribution:

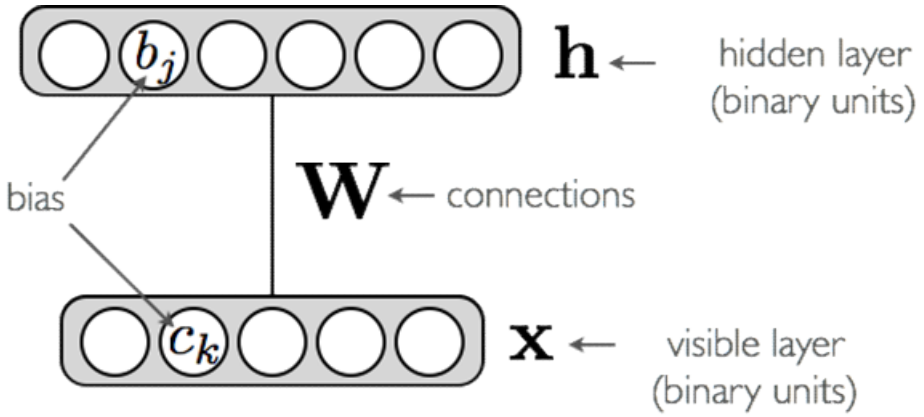
$$p(\mathbf{x}, \mathbf{h}) = \exp(-E(\mathbf{x}, \mathbf{h}))/Z$$

Partition function (intractable)

$$Z = \sum_{\mathbf{x},\mathbf{h}} \exp\left(-E(\mathbf{x}, \mathbf{h})\right)$$

Markov random fields, Boltzmann machines, log-linear models.

Restricted Boltzmann Machines

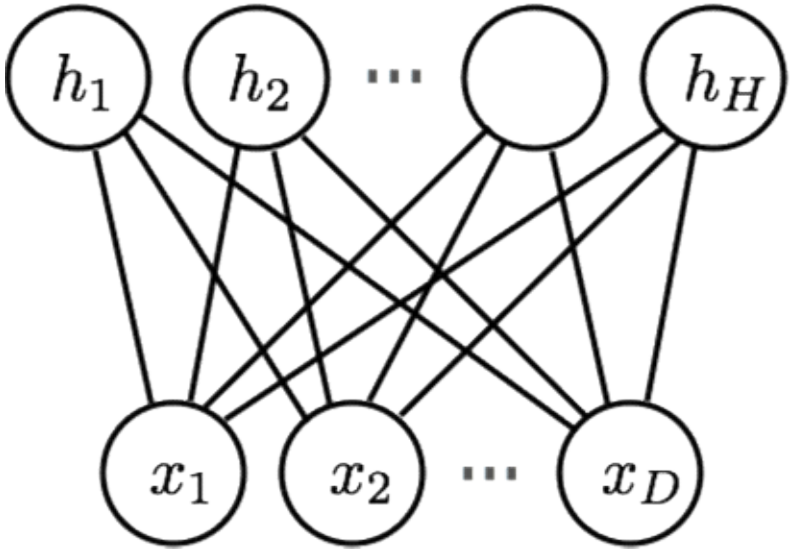


$$
\begin{aligned}
p(\mathbf{x}, \mathbf{h}) &= \exp(-E(\mathbf{x}, \mathbf{h}))/Z \\
&= \exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h})/Z \\
&= \exp(\mathbf{h}^\top \mathbf{W}\mathbf{x}) \exp(\mathbf{c}^\top \mathbf{x}) \exp(\mathbf{b}^\top \mathbf{h})/Z
\end{aligned}
$$

Factors

- The notation based on an energy function is simply an alternative to the representation as the product of factors

Restricted Boltzmann Machines



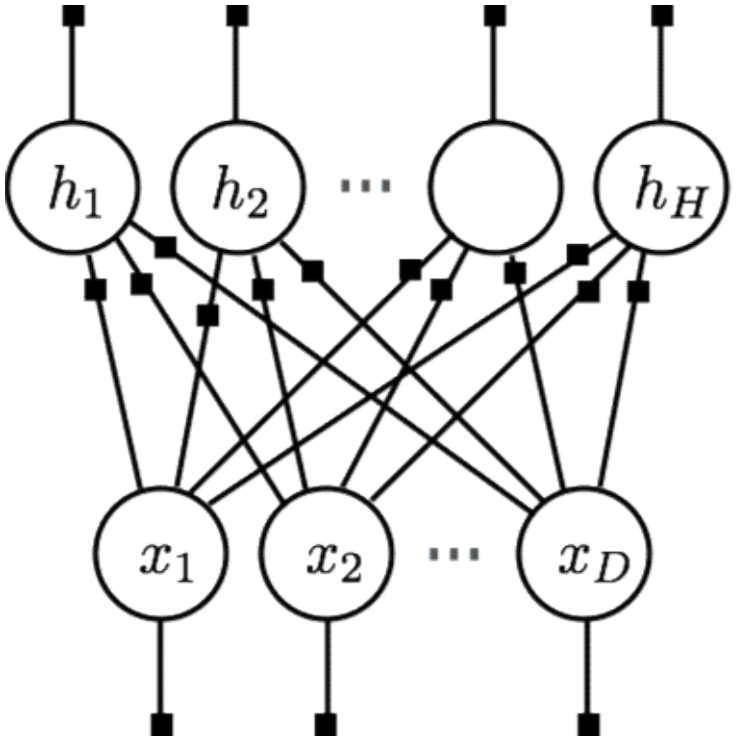
Pair-wise factors

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \prod_j \prod_k \exp(W_{j,k} h_j x_k) \prod_k \exp(c_k x_k) \prod_j \exp(b_j h_j)$$

Unary factors

- The scalar visualization is more informative of the structure within the vectors.

Factor Graph View

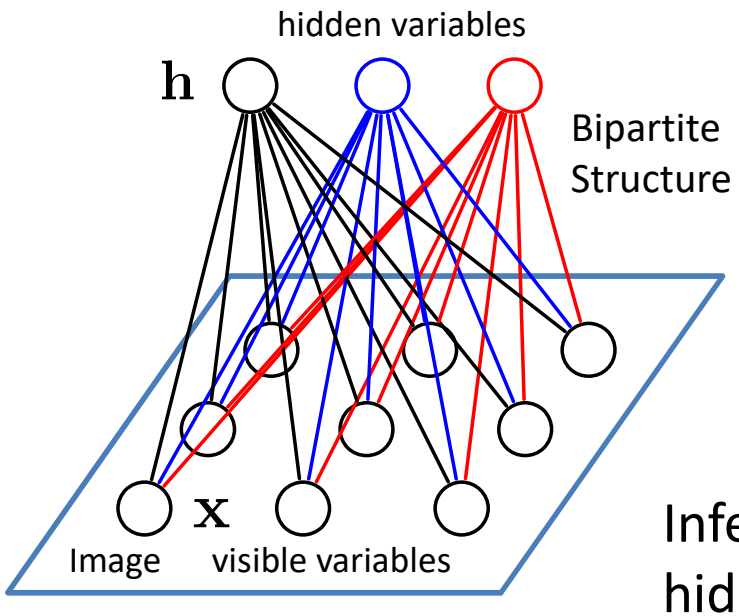


Pair-wise factors

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \prod_j \prod_k \exp(W_{j,k} h_j x_k) \prod_k \exp(c_k x_k) \prod_j \exp(b_j h_j)$$

Unary factors

Inference



Restricted: **No interaction between hidden variables**

Inferring the distribution over the hidden variables is easy:

$$p(\mathbf{h}|\mathbf{x}) = \underbrace{\prod_j p(h_j|\mathbf{x})}$$

Factorizes: Easy to compute

Similarly:

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h})$$

Markov random fields, Boltzmann machines, log-linear models.

In RBM:

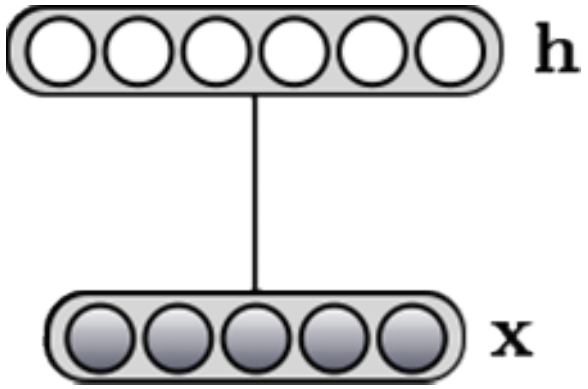
1. Every visible node xk connects to all hidden nodes
2. No visible-visible or hidden-hidden edges

Inference

- Conditional Distributions:

A visible unit's neighbors = all hidden units

A hidden unit's neighbors = all visible units

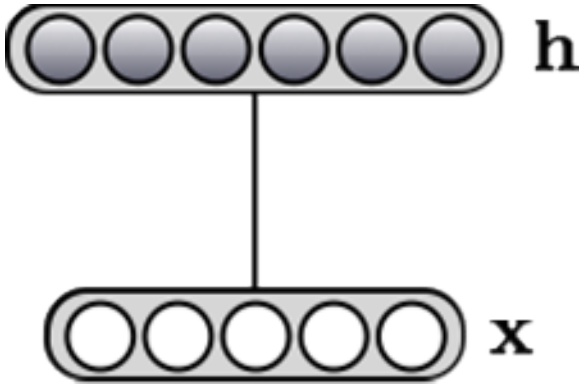


$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$$

$$p(h_j = 1|\mathbf{x}) = \frac{1}{1 + \exp(-(b_j + \mathbf{W}_j.\mathbf{x}))}$$

$$= \text{sigm}(b_j + \mathbf{W}_j.\mathbf{x})$$

j^{th} row of W



$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h})$$

$$p(x_k = 1|\mathbf{h}) = \frac{1}{1 + \exp(-(c_k + \mathbf{h}^\top \mathbf{W}_{.k}))}$$

$$= \text{sigm}(c_k + \mathbf{h}^\top \mathbf{W}_{.k})$$

k^{th} column of W

Local Markov Property

- In general, we have the following property:

$$
\begin{aligned}
p(z_i|z_1,\ldots,z_V) &= p(z_i|\mathrm{Ne}(z_i)) \\
&= \frac{p(z_i,\mathrm{Ne}(z_i))}{\sum_{z_i'} p(z_i',\mathrm{Ne}(z_i))} \\
&= \frac{\prod_{\substack{f \text{ involving } z_i \\ \text{and any } \mathrm{Ne}(z_i)}} \Psi_f(z_i,\mathrm{Ne}(z_i))}{\sum_{z_i'} \prod_{\substack{f \text{ involving } z_i \\ \text{and any } \mathrm{Ne}(z_i)}} \Psi_f(z_i',\mathrm{Ne}(z_i))}
\end{aligned}
$$

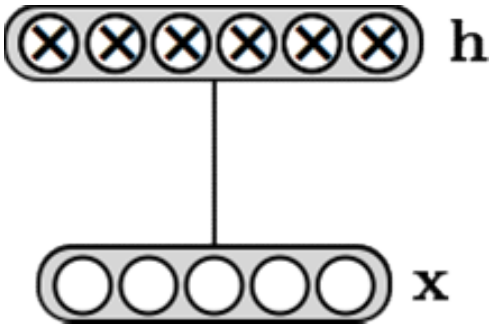
  - z_i is any variable in the Markov network (x_k or h_j in an RBM)
  - $\mathrm{Ne}(z_i)$ are the neighbors of z_i in the Markov network

Free Energy

- What about computing marginal $p(\mathbf{x})$?

$$
\begin{aligned}
p(\mathbf{x}) &= \sum_{\mathbf{h}\in\{0,1\}^H} p(\mathbf{x},\mathbf{h}) = \sum_{\mathbf{h}\in\{0,1\}^H} \exp(-E(\mathbf{x},\mathbf{h}))/Z \\
&= \exp\left(\mathbf{c}^\top\mathbf{x} + \sum_{j=1}^{H} \log(1+\exp(b_j + \mathbf{W}_j.\mathbf{x}))\right)/Z \\
&= \exp(-F(\mathbf{x}))/Z
\end{aligned}
$$

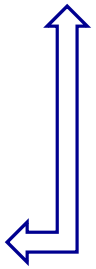
Free Energy



Free Energy

- What about computing marginal $p(\mathbf{x})$?

$$
\begin{aligned}
p(\mathbf{x}) &= \sum_{\mathbf{h}\in\{0,1\}^H} \exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{c}^\top \mathbf{x} + \mathbf{b}^\top \mathbf{h})/Z \\
&= \exp(\mathbf{c}^\top \mathbf{x}) \sum_{h_1\in\{0,1\}} \cdots \sum_{h_H\in\{0,1\}} \exp\left(\sum_j h_j \mathbf{W}_{j.}\mathbf{x} + b_j h_j\right)/Z \\
&= \exp(\mathbf{c}^\top \mathbf{x}) \left(\sum_{h_1\in\{0,1\}} \exp(h_1 \mathbf{W}_{1.}\mathbf{x} + b_1 h_1)\right) \ldots \left(\sum_{h_H\in\{0,1\}} \exp(h_H \mathbf{W}_{H.}\mathbf{x} + b_H h_H)\right)/Z \\
&= \exp(\mathbf{c}^\top \mathbf{x}) \left(1 + \exp(b_1 + \mathbf{W}_{1.}\mathbf{x})\right) \ldots \left(1 + \exp(b_H + \mathbf{W}_{H.}\mathbf{x})\right)/Z \\
&= \exp(\mathbf{c}^\top \mathbf{x}) \exp(\log(1 + \exp(b_1 + \mathbf{W}_{1.}\mathbf{x}))) \ldots \exp(\log(1 + \exp(b_H + \mathbf{W}_{H.}\mathbf{x})))/Z \\
&= \exp\left(\mathbf{c}^\top \mathbf{x} + \sum_{j=1}^{H} \log(1 + \exp(b_j + \mathbf{W}_{j.}\mathbf{x}))\right)/Z
\end{aligned}
$$



- Also known as Product of Experts model.

Free Energy

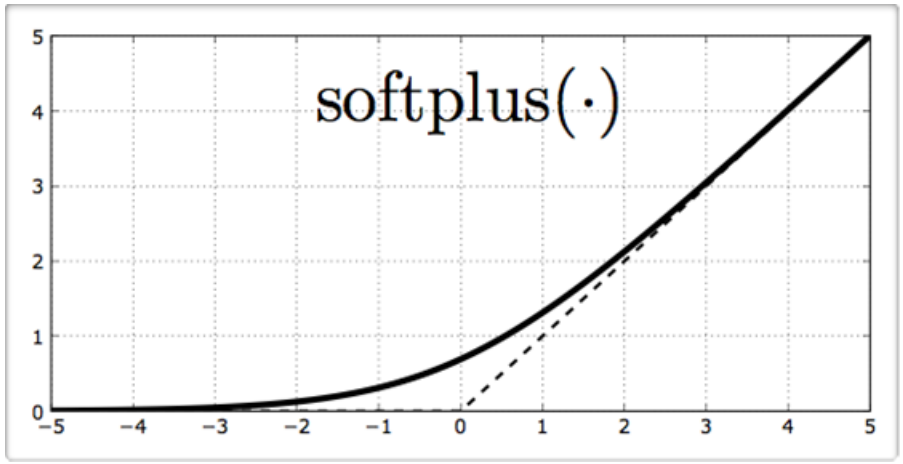
$$p(\mathbf{x}) = \exp\left(\mathbf{c}^\top \mathbf{x} + \sum_{j=1}^{H} \log(1 + \exp(b_j + \mathbf{W}_j.\mathbf{x}))\right) /Z$$

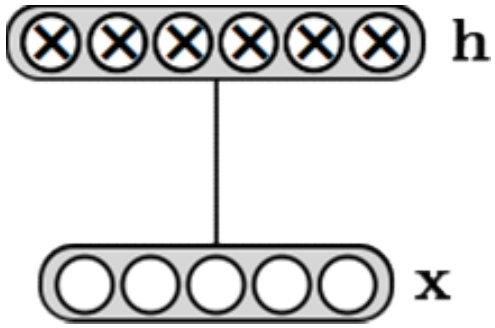
$$= \exp\left(\mathbf{c}^\top \mathbf{x} + \sum_{j=1}^{H} \text{softplus}(b_j + \mathbf{W}_j.\mathbf{x})\right) /Z$$

bias the probability of each x_i

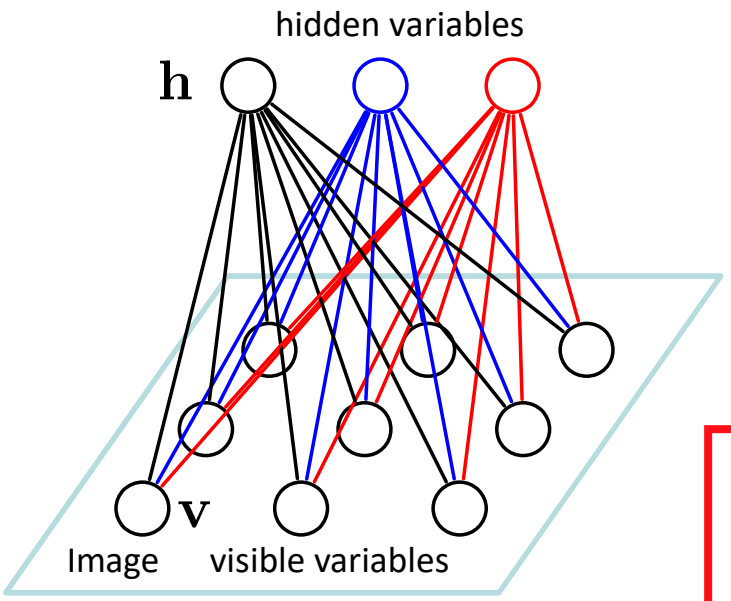
bias of each feature

feature expected in x





Model Learning



• Given a set of *i.i.d.* training examples we want to minimize the average negative log-likelihood (NLL):

$$\frac{1}{T}\sum_t l(f(\mathbf{x}^{(t)})) = \frac{1}{T}\sum_t -\log p(\mathbf{x}^{(t)})$$

Remember:

$$p(\mathbf{x},\mathbf{h}) \quad = \quad \exp(-E(\mathbf{x},\mathbf{h}))/Z$$

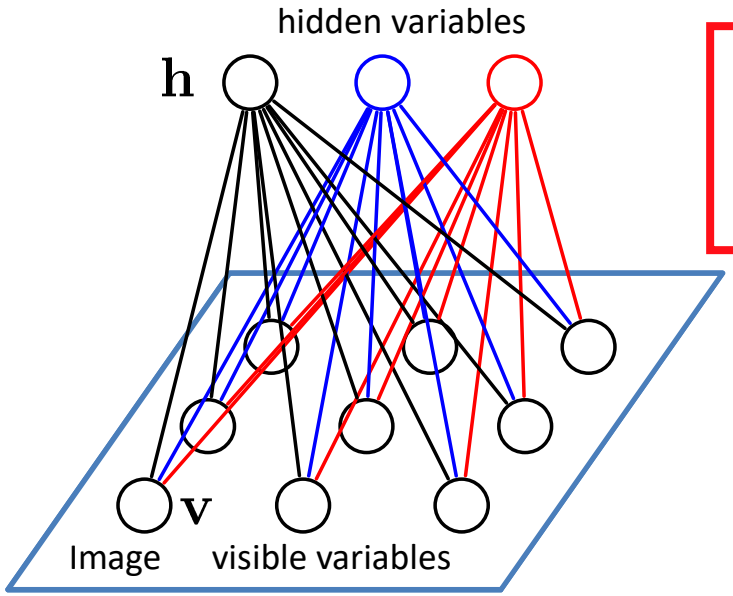
• Derivative of the negative log-likelihood objective:

$$\frac{\partial -\log p(\mathbf{x}^{(t)})}{\partial\theta} = \mathrm{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}^{(t)},\mathbf{h})}{\partial\theta}\,\middle|\,\mathbf{x}^{(t)}\right] - \mathrm{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial\theta}\right]$$

Positive Phase

Negative Phase
Hard to compute

An Important Calculation



Remember:

$$p(\mathbf{x},\mathbf{h}) = \exp(-E(\mathbf{x},\mathbf{h}))/Z$$

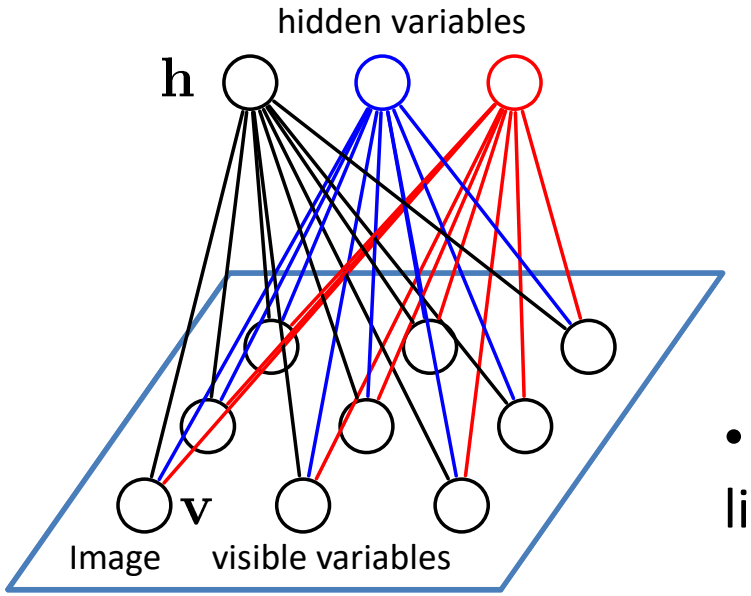
$$Z = \Sigma_{x,h} \exp(-E(x,h))$$

$$\ln(Z) = \ln(\Sigma_{x,h} \exp(-E(x,h)))$$

$$\frac{\partial l\ (Z)}{\partial \theta} = -\frac{1}{Z} \Sigma_{x,h} \exp(-E(x,h)) \frac{\partial E(x,h)}{\partial \theta}$$

$$\frac{\partial ln(Z)}{\partial \theta} = -\Sigma_{x,h} p(x,h) \frac{\partial E(x,h)}{\partial \theta}$$

Model Learning



$$p(\mathbf{x}, \mathbf{h}) = \exp(-E(\mathbf{x}, \mathbf{h}))/Z$$

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$$

• Derivative of the negative log-likelihood objective:

$$\frac{\partial - \log p(\mathbf{x}^{(t)})}{\partial \theta} = \mathrm{E}_{\mathbf{h}} \left[\frac{\partial E(\mathbf{x}^{(t)}, \mathbf{h})}{\partial \theta} \Big| \mathbf{x}^{(t)} \right] - \mathrm{E}_{\mathbf{x},\mathbf{h}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right]$$

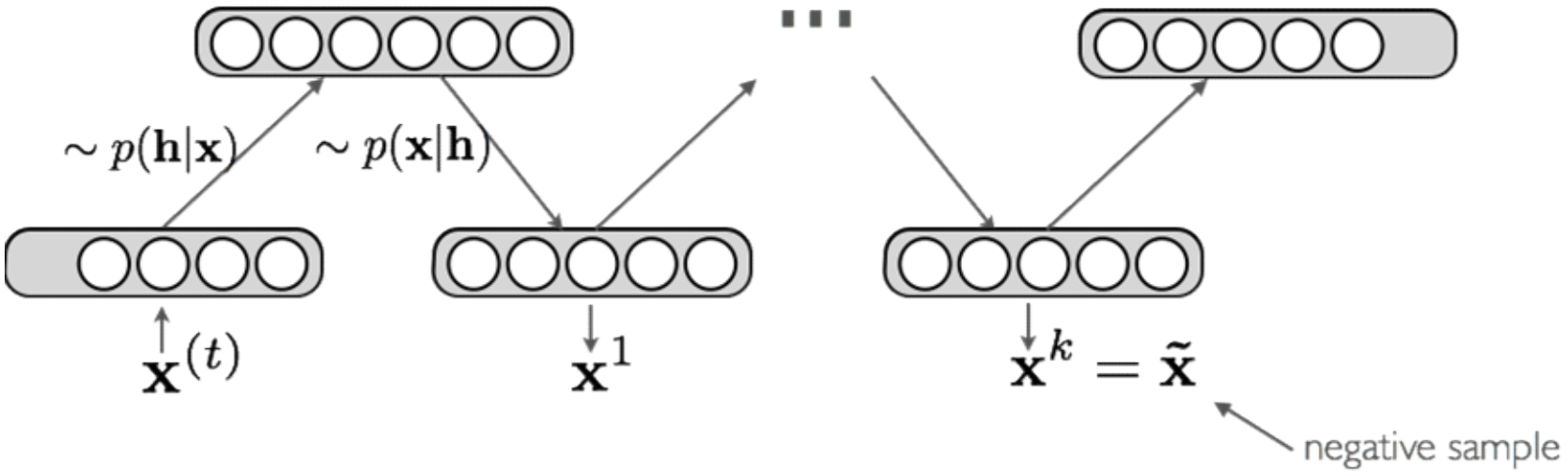
Data-Dependent
Expectations w.r.t P(h|x)

Model: Expectation
w.r.t joint P(x,h)

• Second term: intractable due to exponential number of configurations.

Contrastive Divergence

- Key idea behind Contrastive Divergence:
  - Replace the expectation by a point estimate at $\tilde{\mathbf{x}}$
  - Obtain the point $\tilde{\mathbf{x}}$ by Gibbs sampling
  - Start sampling chain at $\mathbf{x}^{(t)}$



Hinton, Neural Computation, 2002

Deriving Learning Rule

- Let us look at derivative of $\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta}$ for $\theta = W_{jk}$

$$\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \left(-\sum_{jk} W_{jk} h_j x_k - \sum_k c_k x_k - \sum_j b_j h_j \right)$$

$$= -\frac{\partial}{\partial W_{jk}} \sum_{jk} W_{jk} h_j x_k$$

$$= -h_j x_k$$

Remember:

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^\top \mathbf{W} \mathbf{x} - \mathbf{c}^\top \mathbf{x} - \mathbf{b}^\top \mathbf{h}$$

- Hence:

$$\nabla_{\mathbf{W}} E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}\, \mathbf{x}^\top$$

Deriving Learning Rule

- Let us now derive $\mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \theta}\middle|\mathbf{x}\right]$

$$\mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial W_{jk}}\middle|\mathbf{x}\right] = \mathbb{E}_{\mathbf{h}}\left[-h_j x_k\middle|\mathbf{x}\right] = \sum_{h_j\in\{0,1\}} -h_j x_k p(h_j|\mathbf{x})$$

$$= -x_k p(h_j = 1|\mathbf{x})$$

- Hence:

$$\mathrm{E}_{\mathbf{h}}\left[\nabla_{\mathbf{W}} E(\mathbf{x},\mathbf{h})\,|\mathbf{x}\right] = -\mathbf{h}(\mathbf{x})\,\mathbf{x}^\top$$

$$\mathbf{h}(\mathbf{x}) \stackrel{\text{def}}{=} \begin{pmatrix} p(h_1=1|\mathbf{x}) \\ \cdots \\ p(h_H=1|\mathbf{x}) \end{pmatrix}$$

$$= \mathrm{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$

Deriving Learning Rule

- Hence:

$$\mathrm{E}_{\mathbf{h}}\left[\nabla_{\mathbf{W}} E(\mathbf{x},\mathbf{h})\,|\mathbf{x}\right] = -\mathbf{h}(\mathbf{x})\,\mathbf{x}^{\top}$$

$$\mathbf{h}(\mathbf{x}) \overset{\text{def}}{=} \begin{pmatrix} p(h_1=1|\mathbf{x}) \\ \cdots \\ p(h_H=1|\mathbf{x}) \end{pmatrix} = \mathrm{sigm}(\mathbf{b}+\mathbf{W}\mathbf{x})$$

$$\frac{\partial -\log p(\mathbf{x}^{(t)})}{\partial \theta} = \mathrm{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}^{(t)},\mathbf{h})}{\partial \theta}\,\middle|\,\mathbf{x}^{(t)}\right] - \mathrm{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \theta}\right]$$

$$\sum_{\mathbf{x},\mathbf{h}} -\mathbf{h}(\mathbf{x})\mathbf{x}^{\top}\mathbf{P}(\mathbf{x},\mathbf{h})$$

Easy to compute exactly

Difficult to compute: exponentially many Configurations.

Approximate Learning

- An approximation to the gradient of the log-likelihood objective:

$$\frac{\partial - \log p(\mathbf{x}^{(t)})}{\partial \theta} = \mathrm{E}_{\mathbf{h}} \left[\frac{\partial E(\mathbf{x}^{(t)}, \mathbf{h})}{\partial \theta} \middle| \mathbf{x}^{(t)} \right] - \mathrm{E}_{\mathbf{x},\mathbf{h}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right]$$

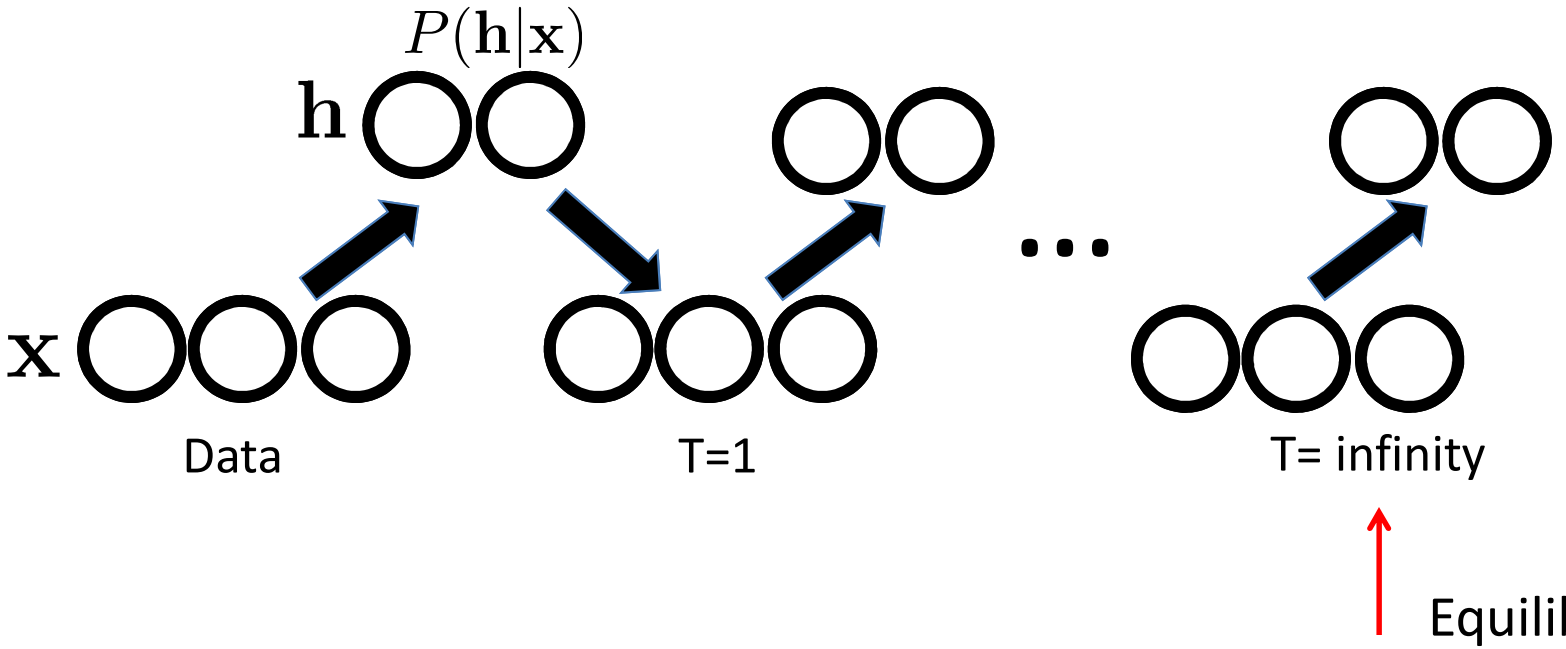
$$\sum_{\mathbf{x},\mathbf{h}} -\mathbf{h}(\mathbf{x})\mathbf{x}^{\top}\mathbf{P}(\mathbf{x}, \mathbf{h})$$

- Replace the average over all possible input configurations by samples.
- Run MCMC chain (Gibbs sampling) starting from the observed examples.

- Initialize $x^0 = x$
- Sample h^0 from $P(h \mid x^0)$
- For t=1:T
  - Sample x^t from $P(x \mid h^{t-1})$
  - Sample h^t from $P(h \mid x^t)$

Approximate ML Learning for RBMs

Run Markov chain (alternating Gibbs Sampling):

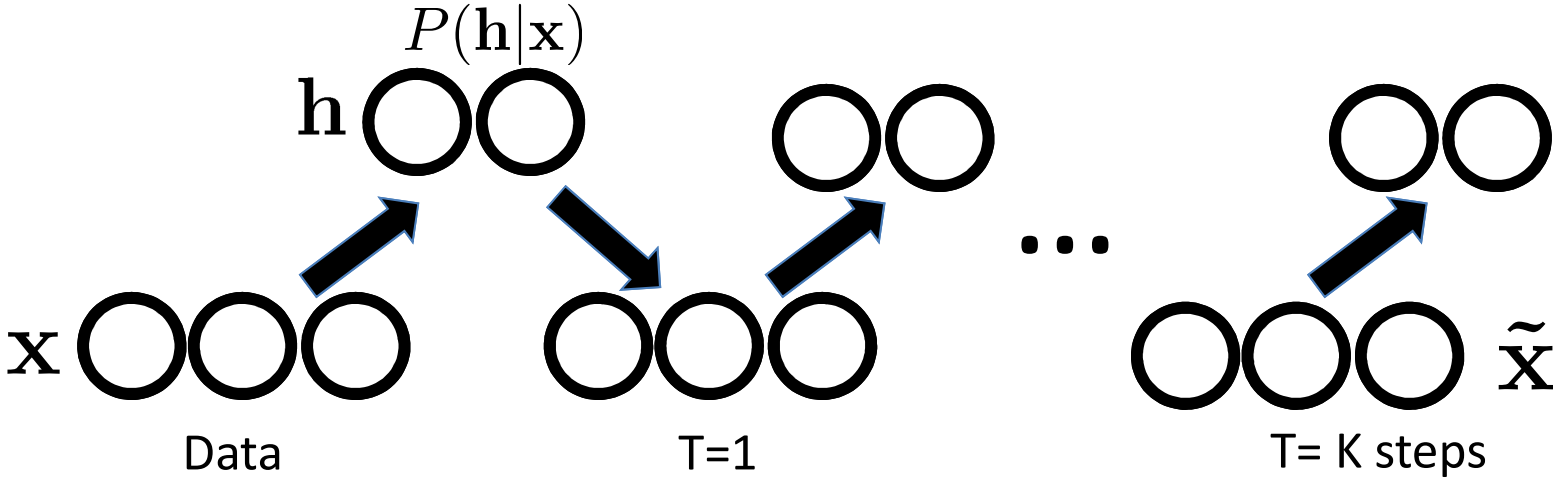


$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$$

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h})$$

Contrastive Divergence

Run Markov chain (alternating Gibbs Sampling):



- K is typically set to 1.

$$p(\mathbf{h}|\mathbf{x}) = \prod_j p(h_j|\mathbf{x})$$

$$p(\mathbf{x}|\mathbf{h}) = \prod_k p(x_k|\mathbf{h})$$

Deriving Learning Rule

$\mathbf{x}^{(t)}$ $\tilde{\mathbf{x}}$ $\theta = \mathbf{W}$

$$
\begin{aligned}
\mathbf{W} &\Longleftarrow \mathbf{W} - \alpha \left(\nabla_{\mathbf{W}} - \log p(\mathbf{x}^{(t)}) \right) \\
&\Longleftarrow \mathbf{W} - \alpha \left(\mathrm{E}_{\mathbf{h}} \left[\nabla_{\mathbf{W}} E(\mathbf{x}^{(t)}, \mathbf{h}) \middle| \mathbf{x}^{(t)} \right] - \mathrm{E}_{\mathbf{x},\mathbf{h}} \left[\nabla_{\mathbf{W}} E(\mathbf{x}, \mathbf{h}) \right] \right) \\
&\Longleftarrow \mathbf{W} - \alpha \left(\mathrm{E}_{\mathbf{h}} \left[\nabla_{\mathbf{W}} E(\mathbf{x}^{(t)}, \mathbf{h}) \middle| \mathbf{x}^{(t)} \right] - \mathrm{E}_{\mathbf{h}} \left[\nabla_{\mathbf{W}} E(\tilde{\mathbf{x}}, \mathbf{h}) \middle| \tilde{\mathbf{x}} \right] \right) \\
&\Longleftarrow \mathbf{W} + \alpha \left(\mathbf{h}(\mathbf{x}^{(t)}) \, \mathbf{x}^{(t)^\top} - \mathbf{h}(\tilde{\mathbf{x}}) \, \tilde{\mathbf{x}}^\top \right)
\end{aligned}
$$

Learning rate

CD-k Algorithm

- For each training example $\mathbf{x}^{(t)}$
  - Generate a negative sample $\tilde{\mathbf{x}}$ using k steps of Gibbs sampling, starting at the data point $\mathbf{x}^{(t)}$
  - Update model parameters:

$$\mathbf{W} \Longleftarrow \mathbf{W} + \alpha \left(\mathbf{h}(\mathbf{x}^{(t)})\, \mathbf{x}^{(t)\top} - \mathbf{h}(\tilde{\mathbf{x}})\, \tilde{\mathbf{x}}^{\top} \right)$$

$$\mathbf{b} \Longleftarrow \mathbf{b} + \alpha \left(\mathbf{h}(\mathbf{x}^{(t)}) - \mathbf{h}(\tilde{\mathbf{x}}) \right)$$

$$\mathbf{c} \Longleftarrow \mathbf{c} + \alpha \left(\mathbf{x}^{(t)} - \tilde{\mathbf{x}} \right)$$

  - Go back to 1 until stopping criteria

CD-k Algorithm

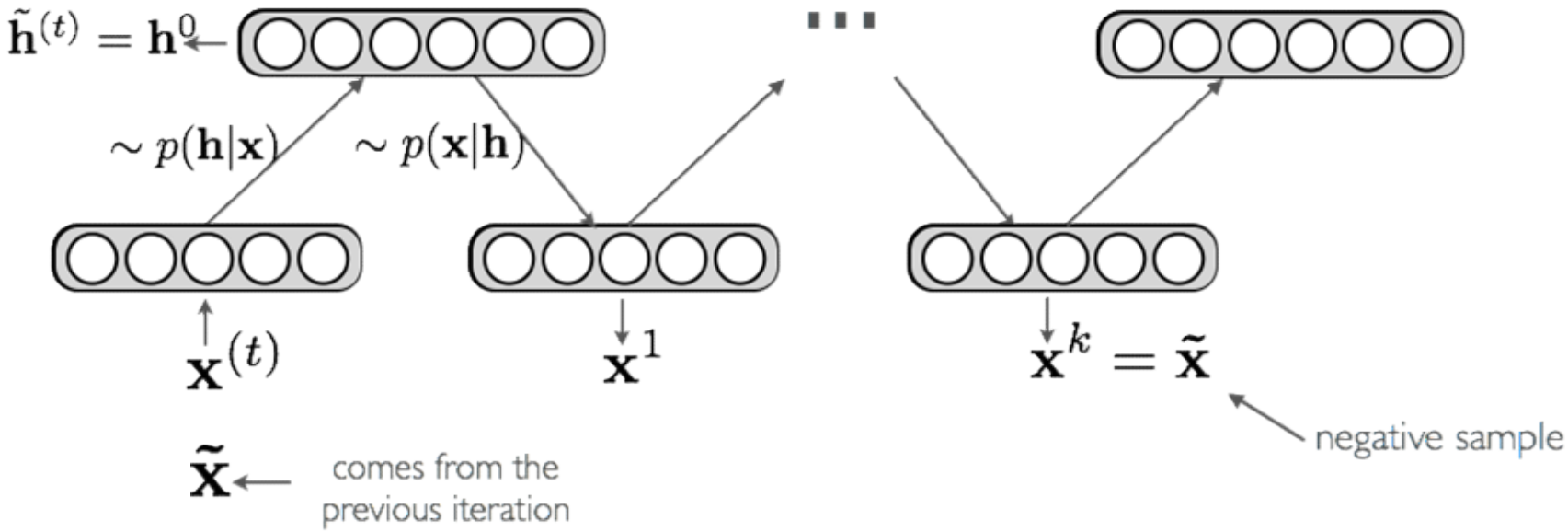
- CD-k: contrastive divergence with k iterations of Gibbs sampling

- In general, the bigger k is, the less biased the estimate of the gradient will be

- In practice, k=1 works well for learning good features and for pre-training

Persistent CD: Stochastic ML Estimator

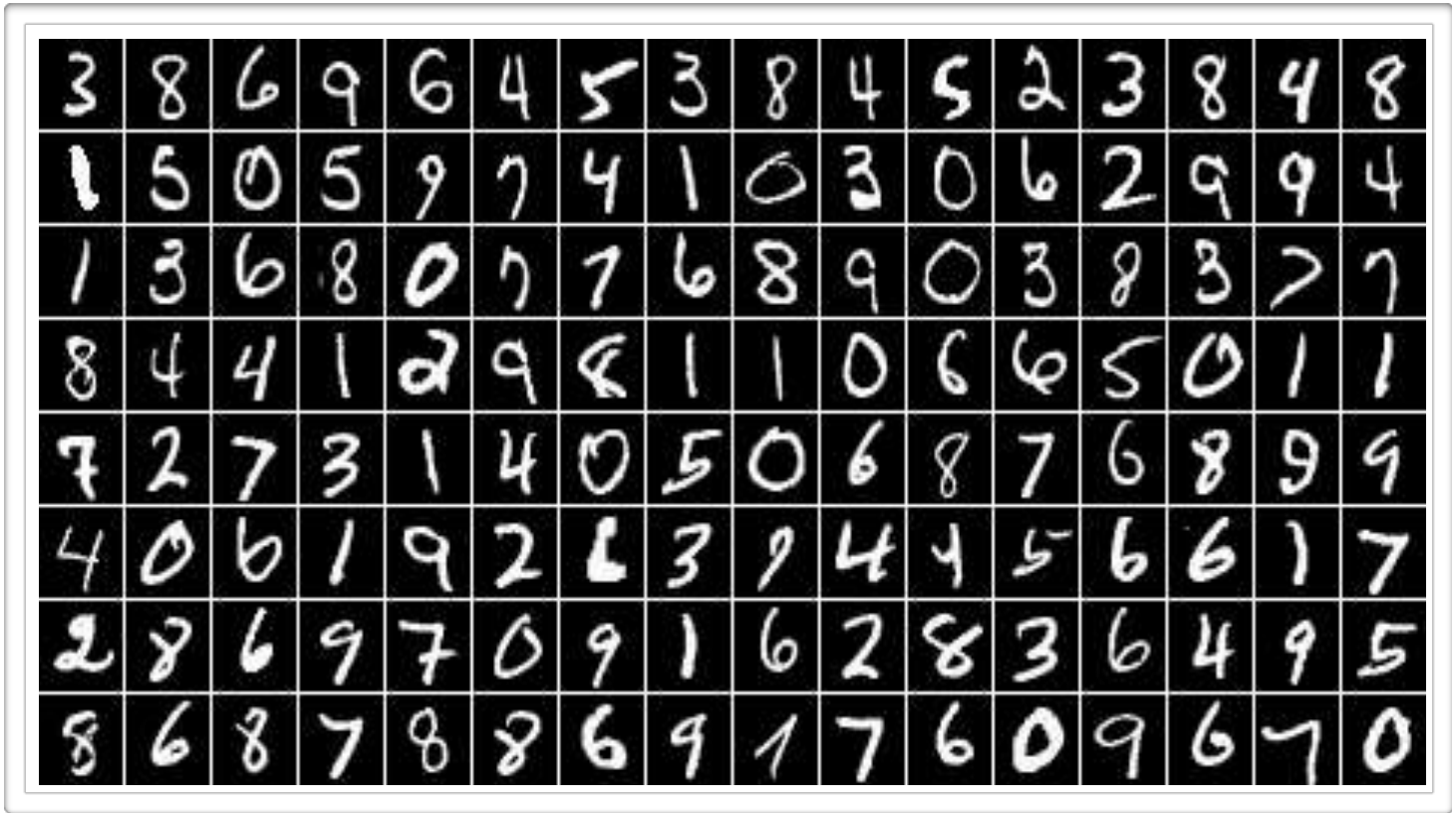
• Idea: instead of initializing the chain to $\mathbf{x}^{(t)}$, initialize the chain to the negative sample of the last iteration



Tieleman, ICML, 2008

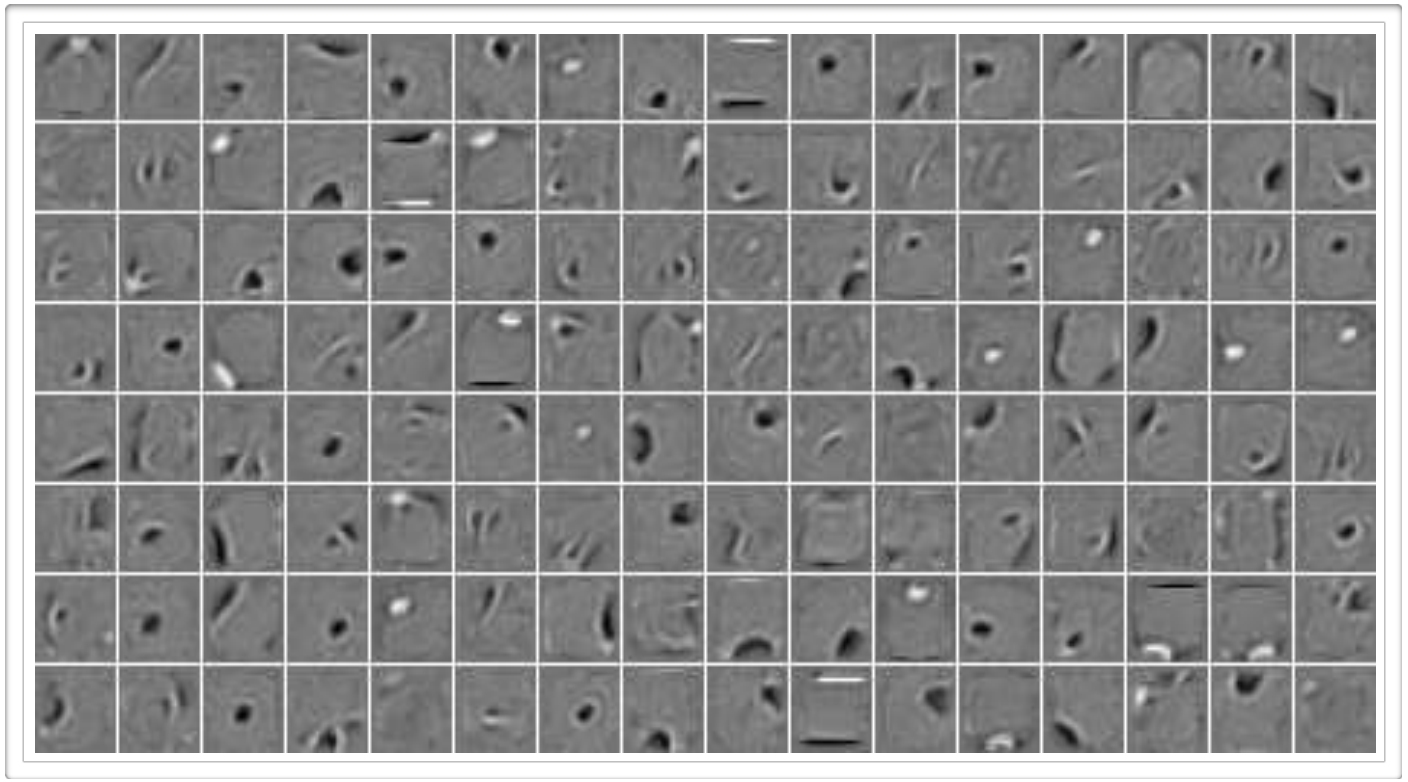
Example: MNIST

- MNIST dataset:



Learned Features

- MNIST dataset:



(Larochelle et al., JMLR 2009)

Gaussian Bernoulli RBMs

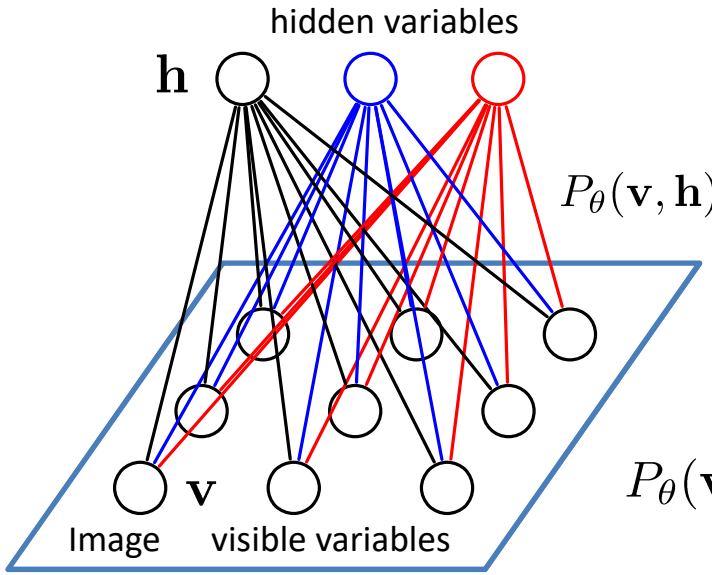
- Let x represent a real-valued (unbounded) input.

  - add a quadratic term to the energy function

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^\top \mathbf{W}\mathbf{x} - \mathbf{c}^\top \mathbf{x} - \mathbf{b}^\top \mathbf{h} + \tfrac{1}{2}\mathbf{x}^\top \mathbf{x}$$

  - In this case $p(\mathbf{x}|\mathbf{h})$ becomes a Gaussian distribution with mean $\boldsymbol{\mu} = \mathbf{c} + \mathbf{W}^\top \mathbf{h}$ and identity covariance matrix
  - recommend to normalize the training set by:
    - subtracting the mean off each input
    - dividing each input by the training set standard deviation
  - should use a smaller learning rate than in the regular RBM

Gaussian Bernoulli RBMs



Pair-wise **Unary**

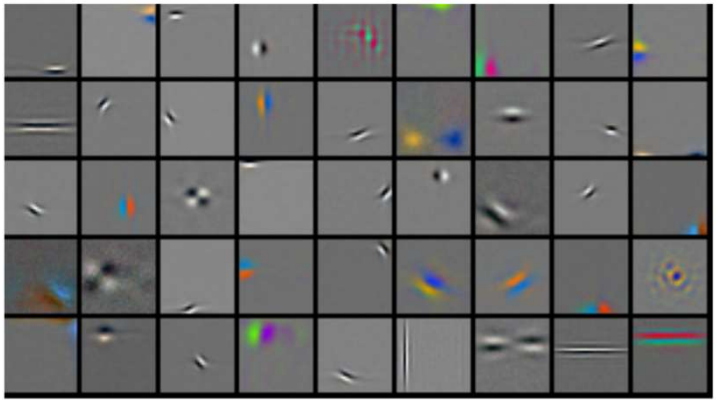
$$P_\theta(\mathbf{v},\mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(\sum_{i=1}^{D}\sum_{j=1}^{F} W_{ij} h_j \frac{v_i}{\sigma_i} + \sum_{i=1}^{D} \frac{(v_i - b_i)^2}{2\sigma_i^2} + \sum_{j=1}^{F} a_j h_j\right)$$

$$\theta = \{W, a, b\}$$

$$P_\theta(\mathbf{v}|\mathbf{h}) = \prod_{i=1}^{D} P_\theta(v_i|\mathbf{h}) = \prod_{i=1}^{D} \mathcal{N}\left(b_i + \sum_{j=1}^{F} W_{ij} h_j, \sigma_i^2\right)$$

4 million **unlabelled** images

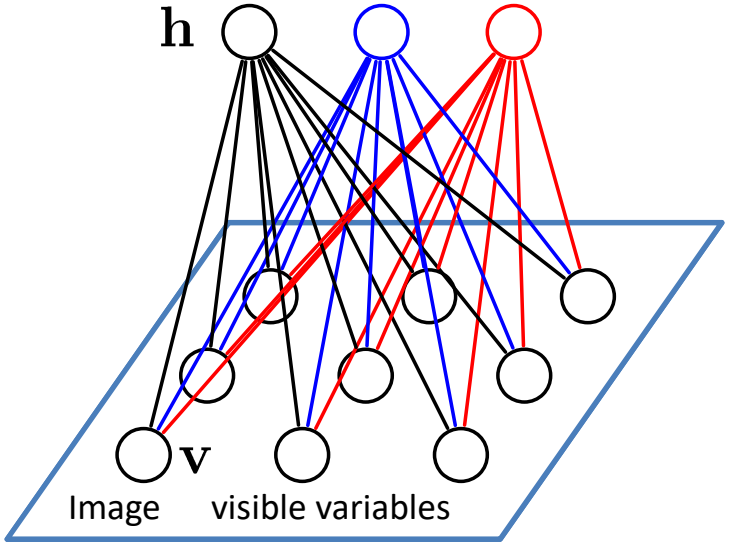
Learned features (out of 10,000)



(Notation: vector x is replaced with v).

Gaussian Bernoulli RBMs

Gaussian-Bernoulli RBM:



Interpretation: Mixture of exponentially growing number of Gaussians

$$P_\theta(\mathbf{v}) = \sum_{\mathbf{h}} P_\theta(\mathbf{v}|\mathbf{h}) P_\theta(\mathbf{h}),$$

where

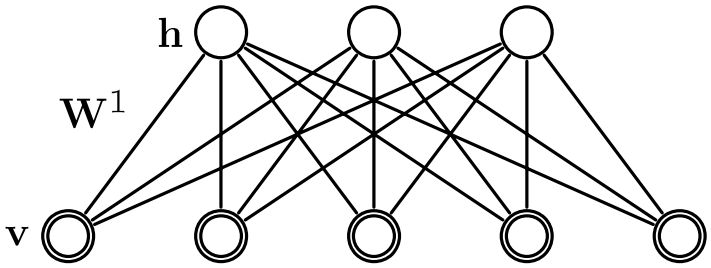
$$P_\theta(\mathbf{h}) = \int_{\mathbf{v}} P_\theta(\mathbf{v}, \mathbf{h}) d\mathbf{v}$$ is an implicit prior, and

$$P(v_i = x|\mathbf{h}) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x - b_i - \sigma_i \sum_j W_{ij} h_j)^2}{2\sigma_i^2}\right)$$ Gaussian

Example: Collaborative Filtering

$$P_\theta(\mathbf{v}, \mathbf{h}) = \frac{1}{\mathcal{Z}(\theta)} \exp\left(\sum_{ijk} W_{ij}^k v_i^k h_j + \sum_{ik} b_i^k v_i^k + \sum_j a_j h_j \right)$$

Binary hidden: user preferences



Multinomial visible: user ratings

Netflix dataset:
480,189 users
17,770 movies
Over 100 million ratings

Learned features: ``genre''

Fahrenheit 9/11	Independence Day
Bowling for Columbine	The Day After Tomorrow
The People vs. Larry Flynt	Con Air
Canadian Bacon	Men in Black II
La Dolce Vita	Men in Black
Friday the 13th	Scary Movie
The Texas Chainsaw Massacre	Naked Gun
Children of the Corn	Hot Shots!
Child's Play	American Pie
The Return of Michael Myers	Police Academy

State-of-the-art performance on the Netflix dataset.

(Salakhutdinov, Mnih, Hinton, ICML 2007)

Variational Autoencoders

Vahid Tarokh

ECE 685D, Fall 2025

Generative Models-Review

- We discussed about two learning paradigms : generative and discriminative
- In generative paradigm the learner tries to learn a joint distribution over all the variables.
- A generative model simulates how the data is generated in the real world.
- Why do we need generative model:
  - Encoding the laws of physics and other constraints into the generative process
  - Expresses causal relations of the world, so we can generalize better to new situations than mere correlations.
  - Building useful abstractions of the world
  - Useful for unsupervised representation learning

VAE-Overview

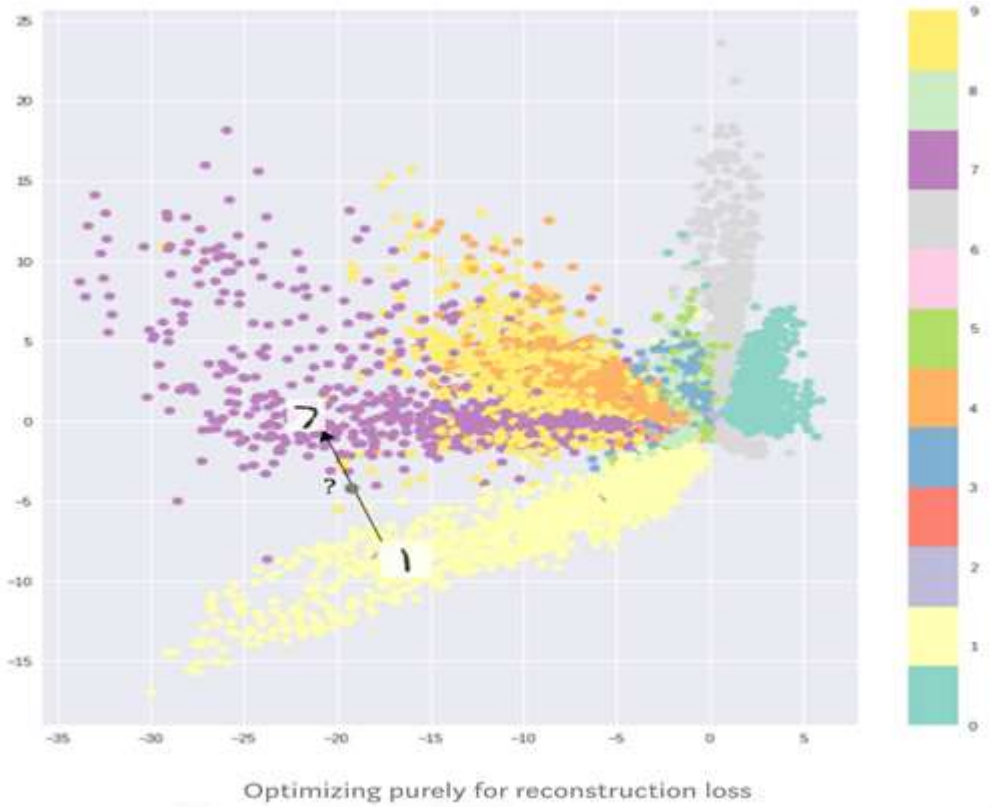
- Variational Autoencoders:
  - Allow us to design complex generative models of data and fit them to large datasets.
  - Can generate fake data/images (e.g., fake celebrity faces) and fake high-resolution digital artwork.

VAE-Overview

- These models also yield state-of-the-art machine learning results in image generation.
- VAEs bridges graphical models and the deep learning
- The general construction is based on generative probability models and does not really need to be implemented using Neural Networks.
- Variational Autoencoders (VAEs) based on Deep Learning were proposed in 2013.
- But since this is a course on Neural Networks, we will mostly implement them using Neural Networks.

Ideal Generative Models

- The fundamental problem with autoencoders:
  - ➢latent encoded set where their encoded vectors lie, may not be contiguous, or allow easy interpolation.



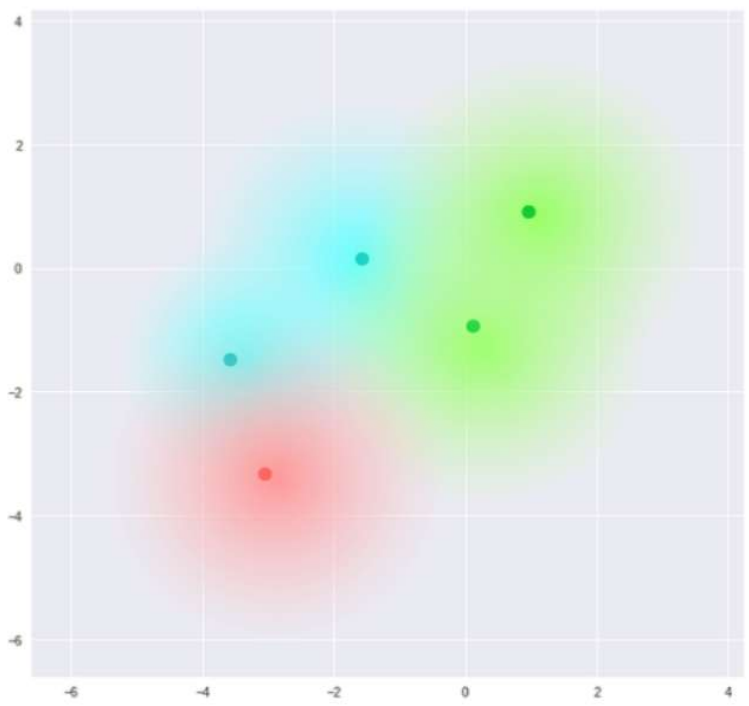
- Training an auto-encoder on the MNIST dataset
- Visualizing the encodings from a 2D latent space
- The formation of distinct clusters.

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

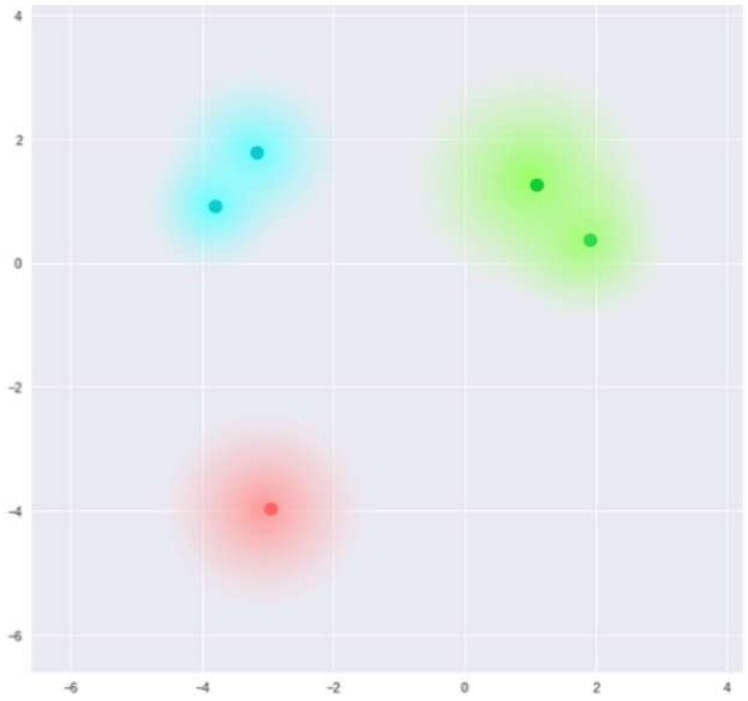
Optimizing only based on the maximum likelihood (reconstruction loss)

Ideal Generative Models

- What we may ideally want: the encodings are close to being "contiguous" (intuitively speaking) while still being distinct,
- This allows smooth interpolation and enables construction of *new* samples.



Desired encoding



Undesired encoding

Parameterization of conditional distributions with Neural Networks

- Probabilistic models based on neural networks are computationally scalable
  - Using stochastic gradient-based optimization
  - Scaling to large models and large datasets
- An example:
  - Parameterizing a categorical distribution (cat), $P_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ using neural networks (NN)
    - y : class label, conditioned on
    - $\boldsymbol{x}$: input image
    - θ : the parameter vector of the distribution

$$\theta = NN(\boldsymbol{x})$$
$$p_\theta(y|\boldsymbol{x})$$

$\boldsymbol{x}$

$NN(\boldsymbol{x})$

$$p(y = dog|\boldsymbol{x}) = 0.979$$
$$p(y = cat|\boldsymbol{x}) = 0.02$$
$$p(y = deer|\boldsymbol{x}) = 0.001$$
$$\vdots$$

Deep Latent-Variable Model (DLVM)

- DLVM denote a latent variable model $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ where distributions are parameterized by neural networks.
- Advantage of DLVM:
  - Even having simple factors (prior or conditional distribution) such as conditional Gaussian, the **marginal likelihood**, $p_\theta(\boldsymbol{x})$ can be arbitrarily complex

DLVM is attractive for approximating complicated underlying distributions

Example:
DLVM for multivariate Bernoulli data

- An example from (Kingma and Welling, 2014) for binary data $\boldsymbol{x}$ and with a spherical Gaussian latent space,

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I})$$ **Latent variable distribution**

$$\mathbf{p} = \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\mathbf{z})$$ **Parametrized Bernoulli with NN**

$$\log p(\mathbf{x}|\mathbf{z}) = \sum_{j=1}^{D} \log p(x_j|\mathbf{z}) = \sum_{j=1}^{D} \log \text{Bernoulli}(x_j; p_j)$$

$$= \sum_{j=1}^{D} x_j \log p_j + (1 - x_j) \log(1 - p_j)$$

D denotes the dimension of x and $0 \leq p_j\text{'s} \leq 1$.

Disadvantage of DLVM

- Intractability of Marginal likelihood (evidence)

$$\int p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})\boldsymbol{dz}$$

  - So, we cannot differentiate it w.r.t. its parameters and optimize it
- This means the intractability of the posterior, $p(\boldsymbol{z}|\boldsymbol{x})$
- Solution: Estimate posterior distribution using Variational Inference (variational Bayes)

VAE Framework

- Using Variational Autoencoders (VAEs), we can efficiently optimize DLVMs jointly with a corresponding inference model using SGD.
- In the probability model framework, a variational auto-encoder (VAE) is assumed to be modeled by a specific probability model of data $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$.
- We can write the joint probability of the model as $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})$
- The generative process can be written as follows:
  - For each data-point i:
    - Draw latent variables $\boldsymbol{z_i} \sim p(\boldsymbol{z})$
    - Draw data-point $\boldsymbol{x_i} \sim p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$

The Goal

- To get around of directly computing the intractable posterior, one can introduce parametric *inference model* $q_\lambda(z|x)$ to approximate the true posterior:

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$$

  - This model is also called an *encoder* or *recognition model*.
  - The parameters of this inference model, λ also called the *variational parameters*.
  - Using the above approximation, we can optimize the marginal likelihood.

- For instance if $q_\lambda(z|x)$ are Gaussian, the λ is the mean vector and co-variance matrix.

VAE Framework

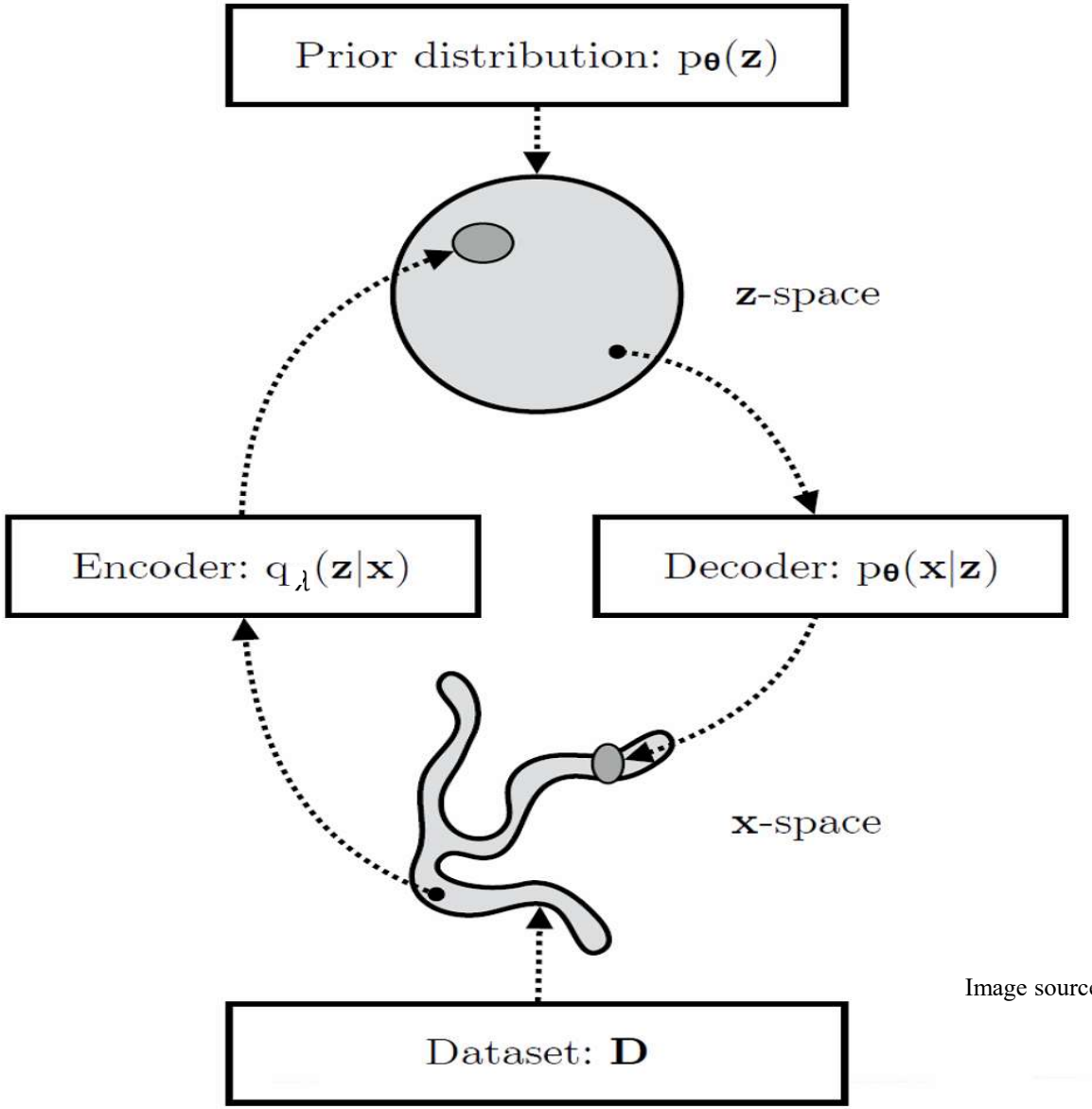


Image source: Kingma and Welling, 2019

Variational Parameters

- Parameterizing the distribution of approximate posterior, $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ by a deep neural network encoder:
  - The variational parameters include the weights and biases of the neural network:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = EncoderNN_\phi(\boldsymbol{x})$$

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$$

  - Using a single encoder neural network to perform posterior inference over all datapoints (shared parameters.
    - This is called *amortized variational inference*
  - In traditional approximate inference models, variational parameters are not shared

Question: How to *choose* λ?

- The optimization objective is reverse KL:
  - The *evidence lower bound*, abbreviated as ELBO (*variational lower bound*)

$$D(q_\lambda(\boldsymbol{z}|\boldsymbol{x})||p_\theta(\boldsymbol{z}|x))$$
$$= E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ] + \log\left(\mathrm{p}_\theta(\boldsymbol{x})\right)$$
$$- \ E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ p_\theta\ (\boldsymbol{x},\boldsymbol{z})\ ]$$

- Hence:

$$log\left(p(\boldsymbol{x})\right) = D(q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ||p_\theta(\ \boldsymbol{z}\ |\ \boldsymbol{x}\ )) + E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ p_\theta\ (\boldsymbol{x},\boldsymbol{z})\ ] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ]$$

Evidence Lower Bound (ELBO)

- This means that minimizing the KL-distance is equivalent to maximizing

$$ELBO = E_{q_\lambda(z|x)}[log\, p_\theta(x,z)] - E_{q_\lambda(z|x)}[log\, q_\lambda(z|x)]$$

- Assume that no two data points share their latent variables with each other then ELBO decomposes into the sum of

$$ELBO_i = E_{q_\lambda(z|x_i)}[log\, p_\theta(x,z)] - E_{q_\lambda(z|x_i)}[log\, q_\lambda(z|x_i)]$$

The Main Message

- This is equal to

$$ELBO_i = E_{q_\lambda(z|x_i)}[\log p_\theta(x_i \mid z)] - D(q_\lambda(z|x_i) \,||\, p(z))$$

Likelihood term (reconstruction part)

Closeness of encoding to p(z) (typically Gaussian)

- Typically $p(z)$ is selected to be standard normal distribution. Then if we have a parametric model class for $p_\theta(x_i \mid z)$, we can maximize the objective function $\sum_i ELBO_i$ over parameters θ and λ using Stochastic Gradient Ascent.

- This process is true regardless of if model classes $p_\theta(x_i|z)$ and $q_\lambda(z|x)$ are given by deep Neural Networks or not!

Computational Flow In a Variational Autoencoder

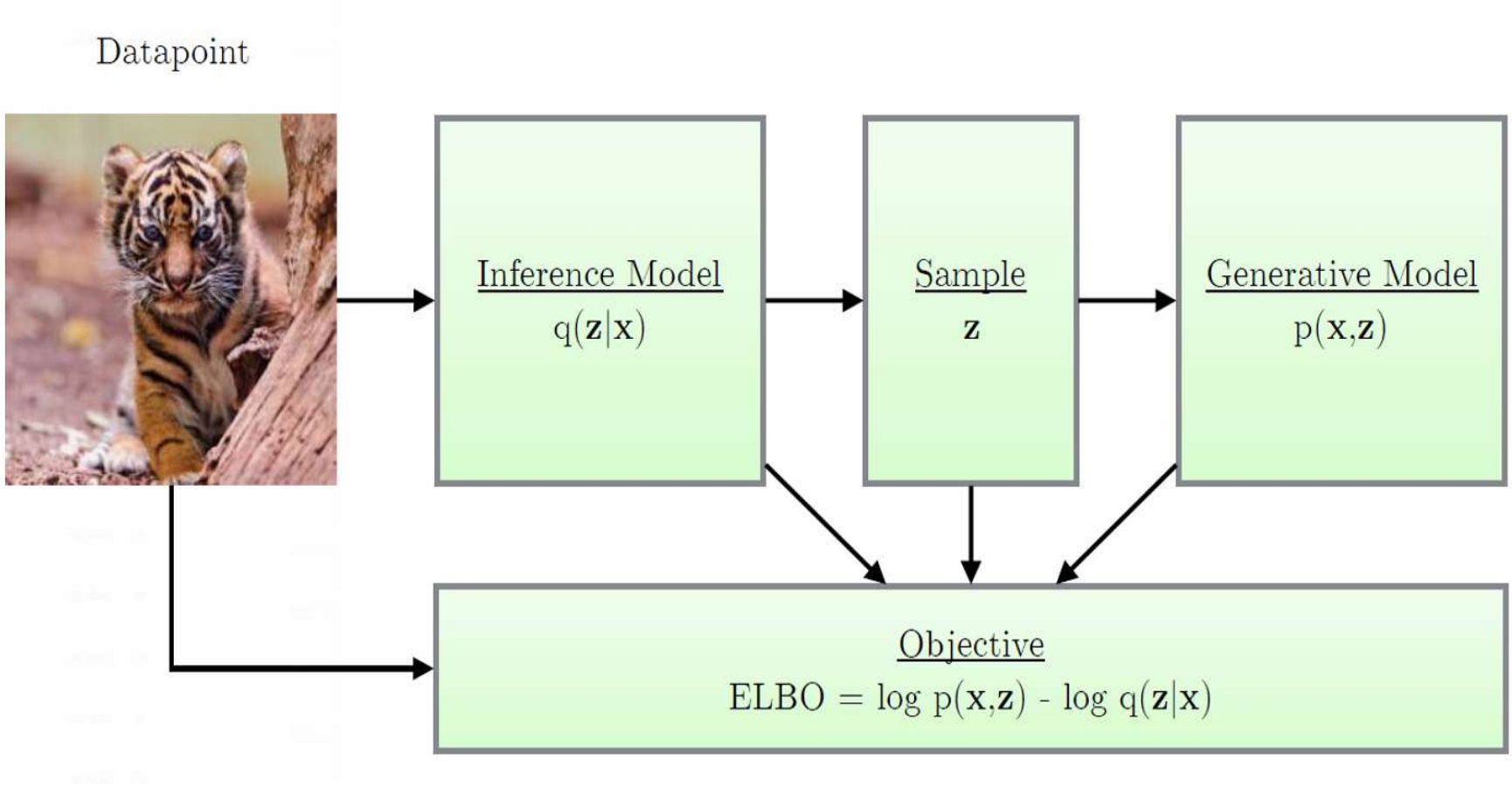


Image source: Kingma and Welling, 2019

Generation of New Data

- Let us introduce some names:
  - $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ is referred to as the encoder
  - $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is referred to as the decoder

- If we have an encoder and decoder designed (optimized as before) and have calculated all the optimized parameters, then to make a new value of x (generate data)

  - Generate z according to $p(\boldsymbol{z})$ (Typically standard Gaussian)
  - Generate x according to $p_\theta(\boldsymbol{x}|\mathbf{z})$

Connecting to Neural Networks

- **But this course is about Neural Networks**.

- Idea: Use DNNs somehow for encoders and Decoders.
  - Encoding Neural Network: Up on observing x, the neural network outputs parameters λ
  - Decoding Neural Network: Up on observing z, the neural network outputs parameters θ

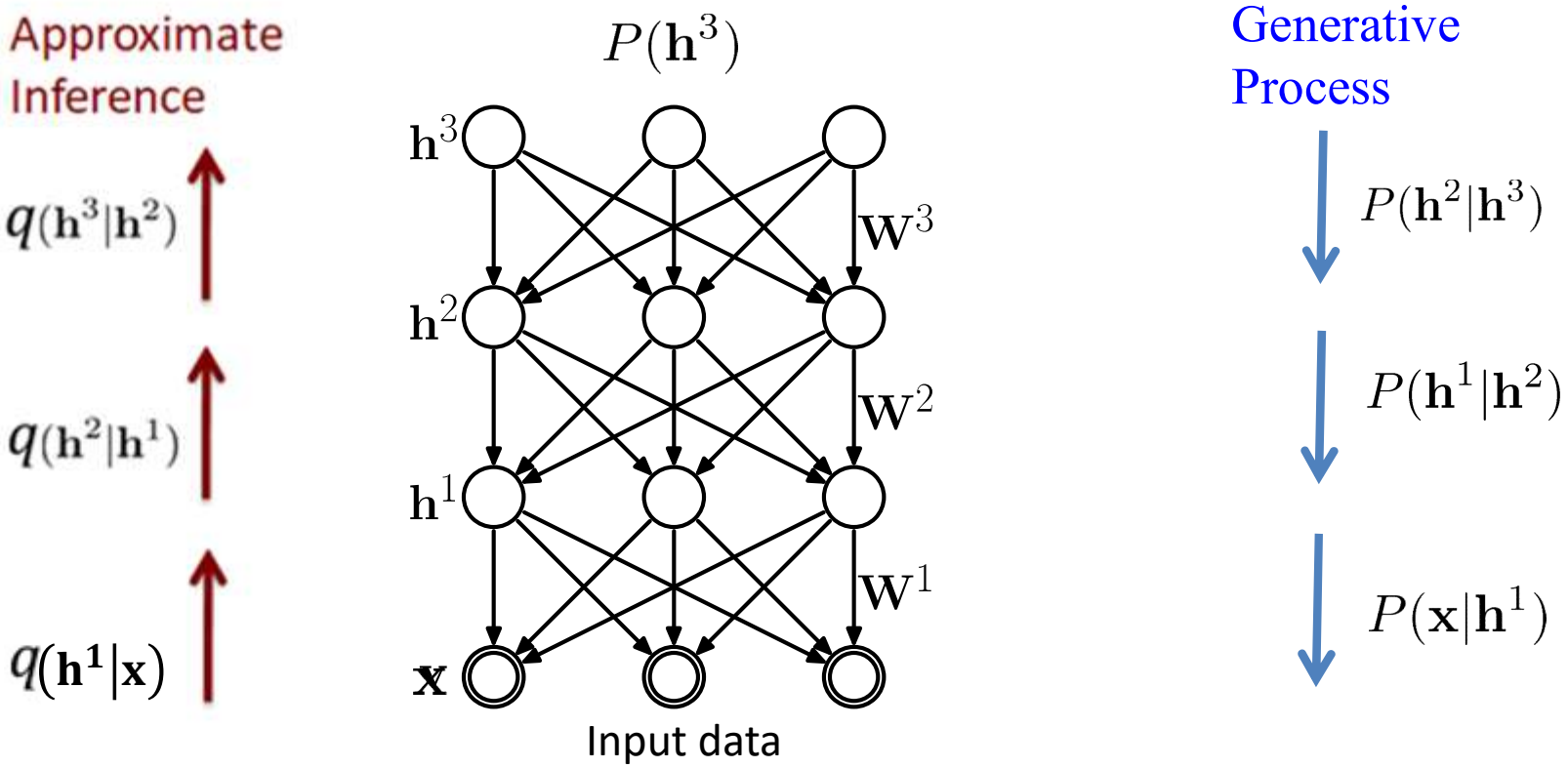
- Equivalently we will have to learn the weights λ and θ of encoding and decoding DNNs using SGD to minimize $-ELBO$.

- We can present this in a fancier way if we wish.

VAEs from Multilayer DNNs

Deep VAE

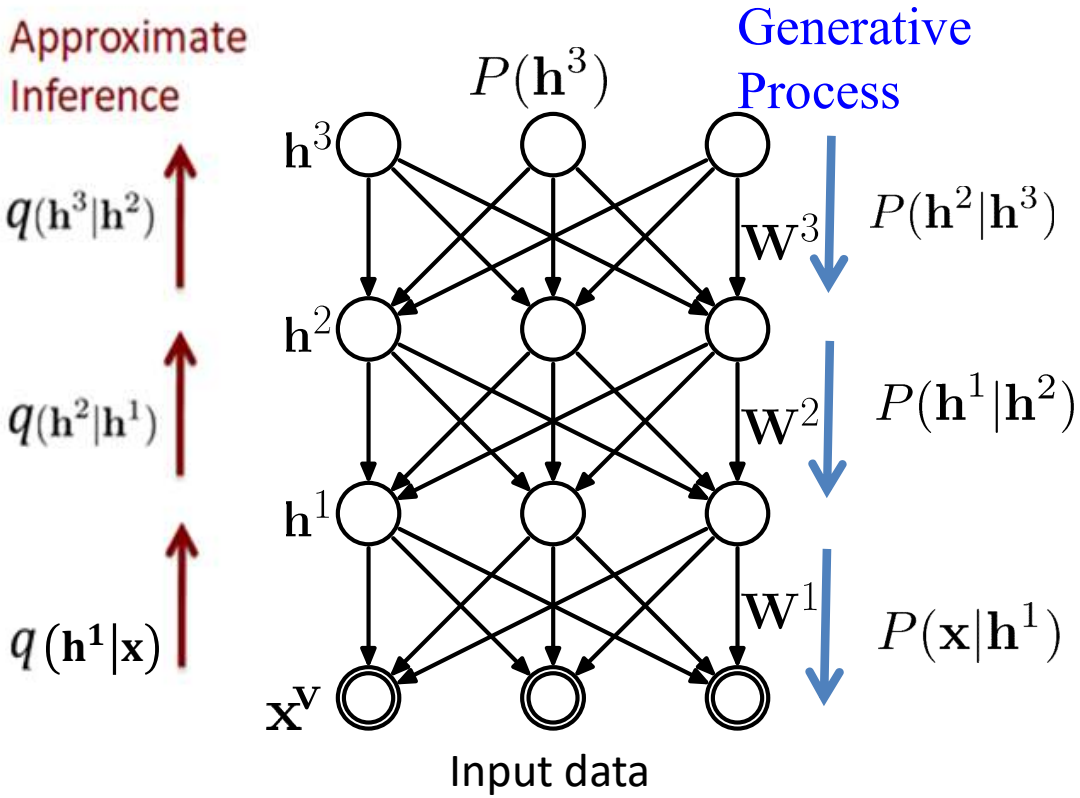
- **Consider multilevel features:**



Generative (Decoder) Network

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{\mathbf{h}^1,\ldots,\mathbf{h}^L} p(\mathbf{h}^L|\boldsymbol{\theta})p(\mathbf{h}^{L-1}|\mathbf{h}^L,\boldsymbol{\theta})\cdots p(\mathbf{x}|\mathbf{h}^1,\boldsymbol{\theta})$$

Each term may denote a complicated nonlinear relationship



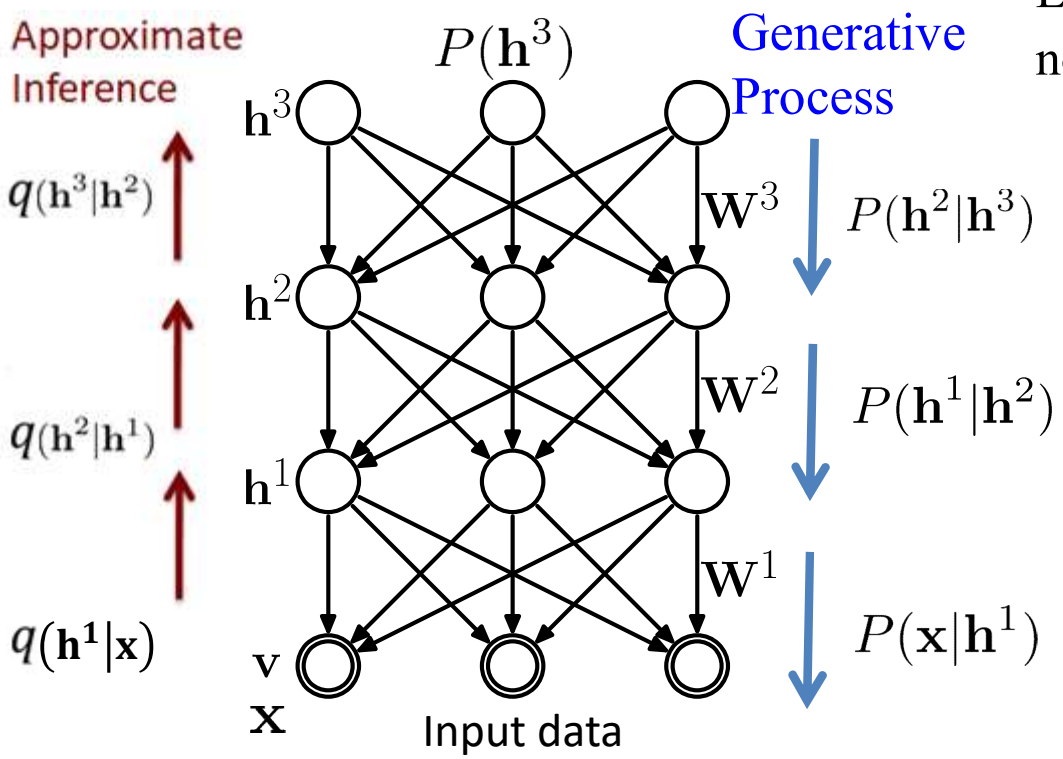
- $\boldsymbol{\theta}$ denotes parameters of decoder.
- L is the number of **stochastic** layers.
- Sampling and probability evaluation is tractable for each $p(\mathbf{h}^\ell|\mathbf{h}^{\ell+1})$.

Recognition (Encoder) Network

- The recognition model (**encoder**) is defined in terms of an analogous factorization:

$$q(\mathbf{h}|\mathbf{x},\boldsymbol{\lambda}) = q\left(\mathbf{h^1}\middle|\mathbf{x},\boldsymbol{\lambda}\right) q\left(\mathbf{h^2}\middle|\mathbf{h^1},\boldsymbol{\lambda}\right) \ldots q\left(\mathbf{h^L}\middle|\mathbf{h}^{\mathrm{L}-1},\boldsymbol{\lambda}\right)$$

Each term may denote a complicated nonlinear relationship



We assume that

$$\mathbf{h}^L \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

The conditionals:

$$p(\mathbf{h}^\ell|\ \mathbf{h}^{\ell+1})$$
$$q(\mathbf{h}^\ell|\mathbf{h}^{\ell-1})$$

Using SGD for training Both Networks

- We can train decoder and encoder networks using SGD algorithm for minimizing the $\mathcal{L}_{\theta,\lambda}(\boldsymbol{x}) \triangleq$ -ELBO ($\mathcal{D}$ denotes a mini-batch):

$$\mathcal{L}_{\theta,\lambda}(\boldsymbol{x}) = \sum_{\boldsymbol{x}\in\mathcal{D}} \mathcal{L}_{\theta,\lambda}(\boldsymbol{x})$$

$$\mathcal{L}_{\theta,\lambda}(\boldsymbol{x}) = E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})]$$

- Obtaining Unbiased gradient w.r.t the decoder parameters, θ is simple:

$$\begin{aligned}\nabla_\theta(\mathcal{L}_{\theta,\lambda}(\boldsymbol{x})) &= \nabla_\theta(E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})]) \\ &= E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[\nabla_\theta(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})] \\ &\simeq \nabla_\theta\,(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})) \\ &= \nabla_\theta(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}))\end{aligned}$$

Using SGD for training Both Networks

- Unbiased gradient w.r.t the variational parameters, λ is more difficult:
  - The Expectation is taken w.r.t the $q_\lambda(\mathbf{z}|\mathbf{x})$, which is a function of λ !!!

$$\nabla_\lambda(\mathcal{L}_{\theta,\lambda}(\boldsymbol{x})) = \nabla_\lambda(E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})])$$
$$\neq E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[\nabla_\lambda(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})]$$

- Fortunately, for the continuous r.v's, the unbiased estimator of the gradient can be obtained through the reparameterization trick.

Reparameterization Trick

- This trick is essentially the application of change of variables:
  - Let z be a r.v distributed as $q_\lambda(z|x)$
  - Assume, there is a differentiable and invertible function h such that

$$z = h(\epsilon, x, \lambda)$$

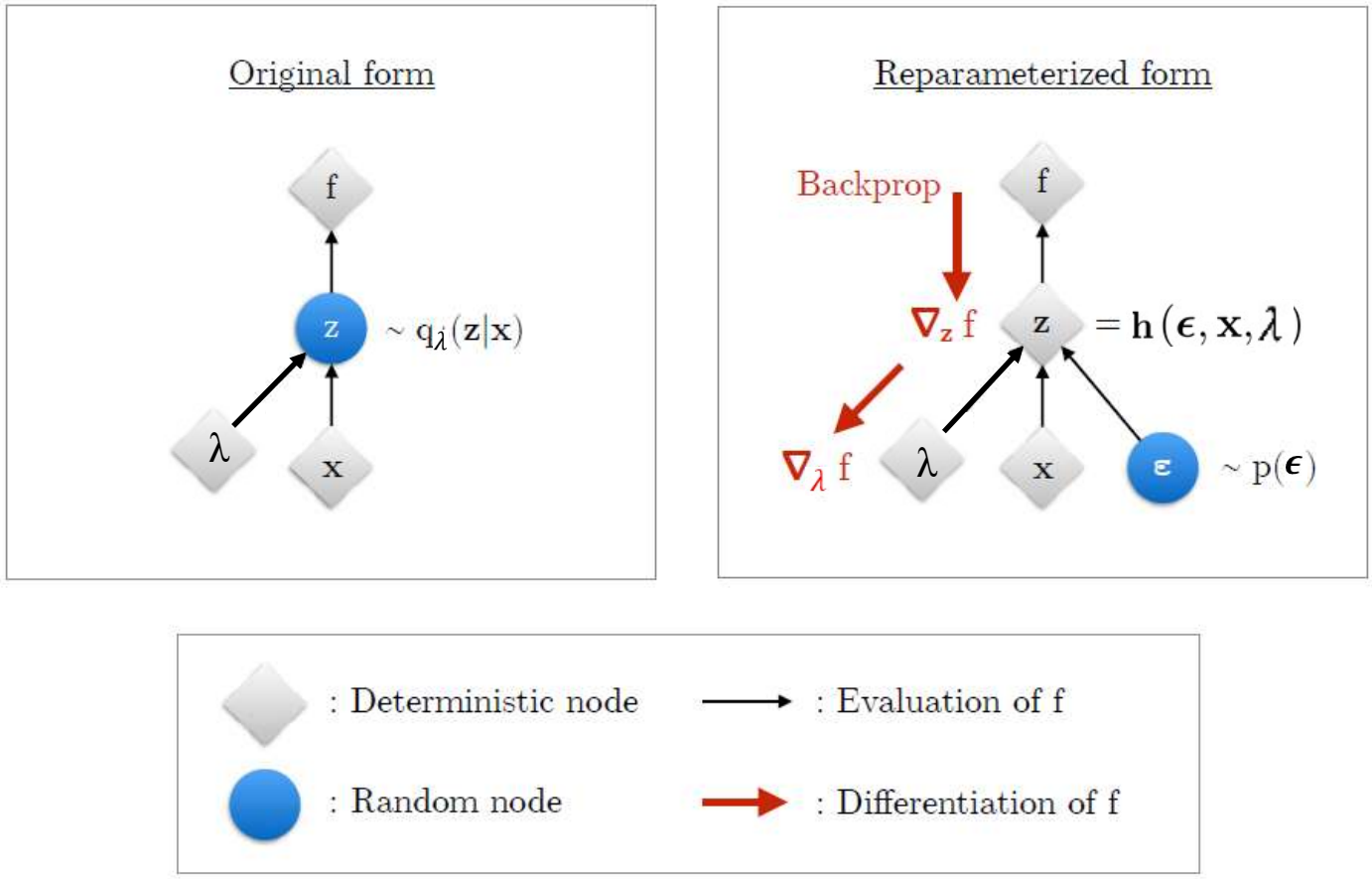
- Where $\varepsilon \sim p(\varepsilon)$ is another random variable which is independent of x and λ. Hence:

$$E_{q(z|x)}[f(z)] = E_{p(\varepsilon)}[f(z)]$$

- Now, we can safely exchange the gradient and expectation (f(z) $= log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})$):

$$\nabla_\lambda\, E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})\,] = E_{p(\varepsilon)}\,[\nabla_\lambda(log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x}))]$$
$$= \nabla_\lambda E_{p(\varepsilon)}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})] \simeq \nabla_\lambda(log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x}))$$

Illustration of Reparameterization Trick



- The reparametrized ELBO estimator is referred to as the **Stochastic Gradient Variational Bayes (SGVB)** estimator (Kingma and Welling, 2014)

Image source: Kingma and Welling, 2019

Computing the Distribution of The Approximate Posterior After Change of Variables

- The goal is to compute the distribution of $log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ by changing z as a function of ε .
- This is easy task as long as the function h is chosen appropriately
- Using the change of variable rule in probability:

$$log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x}) = \log p(\boldsymbol{\epsilon}) - \log\left|\det\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}\right)\right|$$

- $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}$ denotes the Jacobian matrix which is computed through $\boldsymbol{z} = h(\boldsymbol{\epsilon}, \boldsymbol{x}, \boldsymbol{\lambda})$.

Two common and simple choices for function h

1. Factorized Gaussian posterior:

- We can assume that $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ is given by

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2))$$

- In this case, variational parameters, and the approximate posterior, $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ are given by:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{EncoderNN}_\lambda(\mathbf{x})$$

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2)$$

Univariate Gaussian Distribution

- We take h as an affine function of $\boldsymbol{\varepsilon}$ as follows:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

Jacobian: $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = diag(\boldsymbol{\sigma}) \xrightarrow{\text{yields}} \log \left| \det \left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} \right) \right| = \sum_i log(\sigma_i)$

Two common and simple choices for function h

2. Full-covariance Gaussian posterior

- We can assume that $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ is given by

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$
$$\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^{\mathbf{T}}$$ Cholesky Decomposition

- $\mathbf{L}$ is a lower or upper triangle matrix with non-zero entries in the diagonal.

- In this case, variational parameters is given by:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}, \boldsymbol{L}') = \text{EncoderNN}_\lambda(\mathbf{x})$$
$$\boldsymbol{L} = \boldsymbol{L_{mask}} \odot \mathbf{L}' + \text{diag}(\boldsymbol{\sigma})$$

- L_{mask} is a 0-1 matrix with zero's on and above (lower) diagonal.

- We take h as an affine function of $\boldsymbol{\varepsilon}$ as follows:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}$$

Jacobian: $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \boldsymbol{L} \xrightarrow{\text{yields}} \log \left|\det\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}\right)\right| = \sum_i log|L_{ii}|$

Computing the Gradients

- The gradient w.r.t the parameters: both recognition and generative:

$$\nabla_{\theta,\lambda} \mathbb{E}_{\boldsymbol{z} \sim q(\boldsymbol{z}|\mathbf{x},\lambda)} \left[\log \frac{p(\mathbf{x}, \boldsymbol{z}|\boldsymbol{\theta})}{q(\boldsymbol{z}|\mathbf{x},\lambda)} \right]$$

$$= \nabla_{\theta,\lambda} \mathbb{E}_{\boldsymbol{\epsilon}^1,\ldots,\boldsymbol{\epsilon}^L \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})} \left[\log \frac{p(\mathbf{x}, h(\boldsymbol{\epsilon}, \mathbf{x}, \boldsymbol{\theta})|\boldsymbol{\theta})}{q(\mathbf{h}(\boldsymbol{\epsilon}, \mathbf{x}, \lambda)|\mathbf{x}, \lambda)} \right]$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}^1,\ldots,\boldsymbol{\epsilon}^L \sim \mathcal{N}(\mathbf{0},\boldsymbol{I})} \left[\nabla_{\theta,\lambda} \log \frac{p(\mathbf{x}, h(\boldsymbol{\epsilon}, \mathbf{x}, \boldsymbol{\theta})|\boldsymbol{\theta})}{q(\mathbf{h}(\boldsymbol{\epsilon}, \mathbf{x}, \lambda)|\mathbf{x}, \lambda)} \right]$$

Gradients can be computed by backprop

The mapping $\boldsymbol{h}$ is a deterministic neural net for fixed $\boldsymbol{\epsilon}$.

Computing the Gradients

- The gradient w.r.t the parameters: recognition and generative:

is h, not g

$$\nabla_{\theta,\lambda}\mathbb{E}_{\boldsymbol{z}\sim q(\boldsymbol{z}|\mathbf{x},\lambda)}\left[\log\frac{p(\mathbf{x},\boldsymbol{z}|\boldsymbol{\theta})}{q(\boldsymbol{z}|\mathbf{x},\boldsymbol{\lambda})}\right]=\mathbb{E}_{\boldsymbol{\epsilon}^1,\ldots,\boldsymbol{\epsilon}^L\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\nabla_{\theta,\lambda}\log\frac{p(\mathbf{x},g(\boldsymbol{\epsilon},\mathbf{x},\boldsymbol{\theta})|\boldsymbol{\theta})}{q(h(\boldsymbol{\epsilon},\mathbf{x},\lambda)|\mathbf{x},\boldsymbol{\lambda})}\right]$$

- Approximate expectation by generating k samples from $\boldsymbol{\epsilon}$:

$$\frac{1}{k}\sum_{i=1}^{k}\nabla_{\theta,\lambda}\log w\left(\mathbf{x},\mathbf{h}(\boldsymbol{\epsilon}_i,\mathbf{x},\boldsymbol{\lambda}),\boldsymbol{\theta}\right)$$

- Where we defined unnormalized importance weights:

$$w(\mathbf{x},\mathbf{h},\boldsymbol{\theta})\;=\;p(\mathbf{x},\mathbf{h}|\boldsymbol{\theta})/q(\mathbf{h}|\mathbf{x},\boldsymbol{\lambda})$$

Relation between ML and ELBO

- With i.i.d samples from dataset $\mathcal{D}$, the maximum likelihood principle is given by:

$$log\ p_\theta(\mathcal{D}) = \frac{1}{N}\sum_{\boldsymbol{x}\in\mathcal{D}} log\ p_\theta(\boldsymbol{x}) = E_{q_\mathcal{D}(\boldsymbol{x})} \log p_\theta(\boldsymbol{x})$$

- $q_\mathcal{D}(\boldsymbol{x})$ is the empirical distribution of data
- Recall that

$$Max_\theta\ log\ p_\theta(\mathcal{D}) = \min_\theta D_{kl}(q_\mathcal{D}(\boldsymbol{x})||p_\theta(\boldsymbol{x}))$$

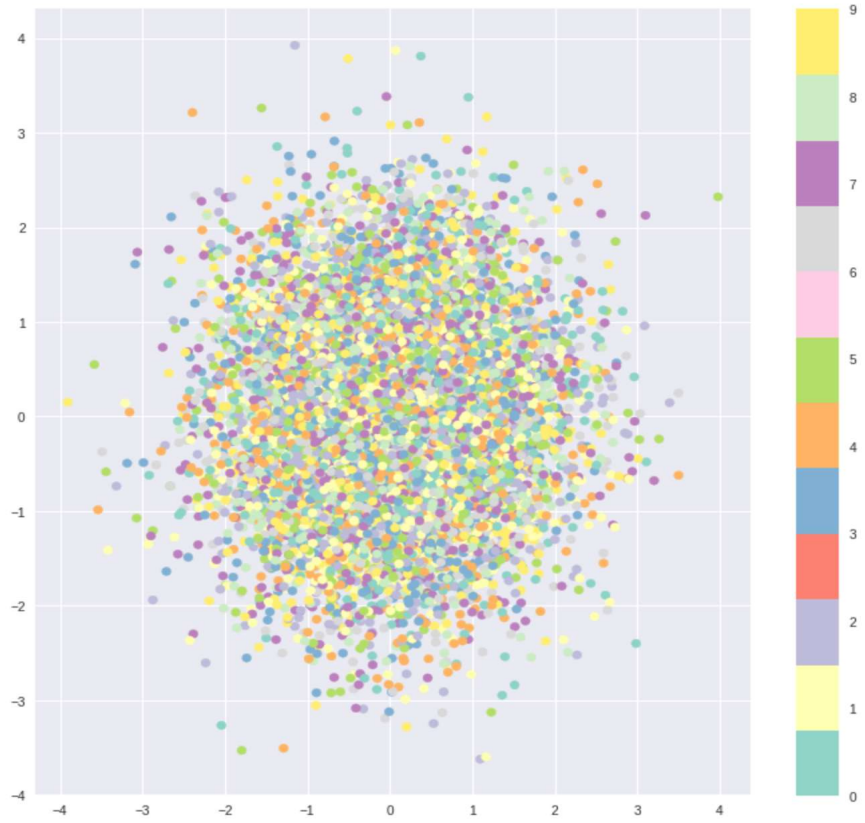
- Combining the empirical distribution of data and the inference model in VAE, $q_{\lambda,\mathcal{D}}(\boldsymbol{x},\boldsymbol{z})$=$q_\mathcal{D}(\boldsymbol{x})q_\lambda(\boldsymbol{z}|\mathrm{x})$, one can show that

$$D_{kl}(q_{\lambda,\mathcal{D}}(\boldsymbol{x},\boldsymbol{z})||p_\theta(\boldsymbol{x},\boldsymbol{z})) \geq D_{kl}(q_\mathcal{D}(\boldsymbol{x})||p_\theta(\boldsymbol{x}))$$

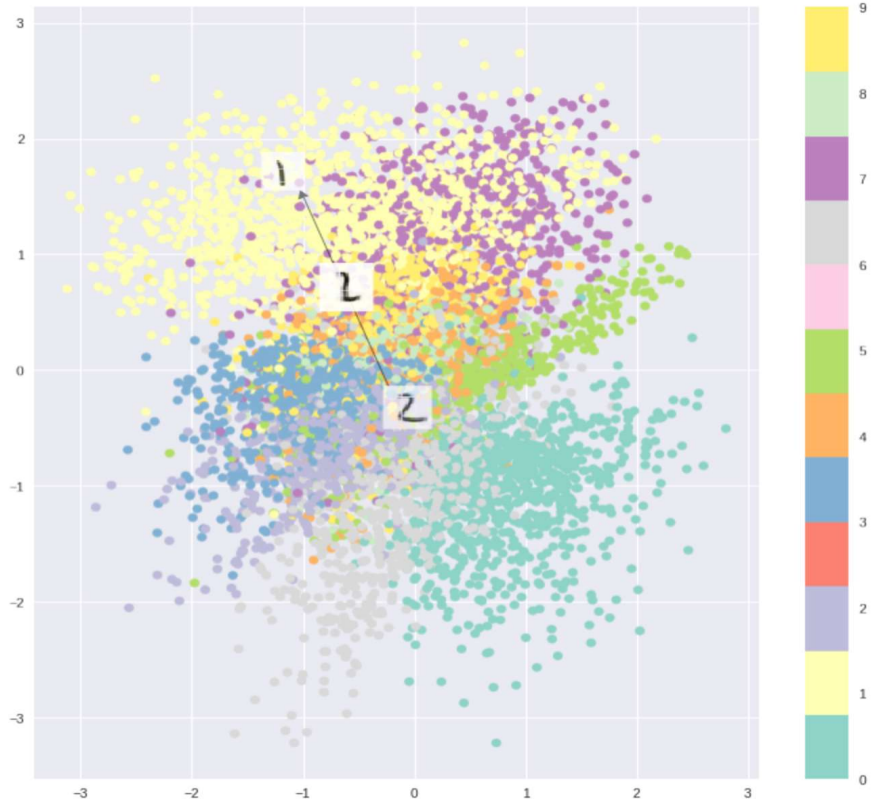
- ELBO can be thought as a maximum likelihood objective *in an augmented space,* $(\boldsymbol{x},\boldsymbol{z})$

Trade-off in the VAE loss

$$E_{q_\lambda(z|x_i)}[\log p(x_i \mid z)] - D(q_\lambda(z|x) \,||\, p(z))$$



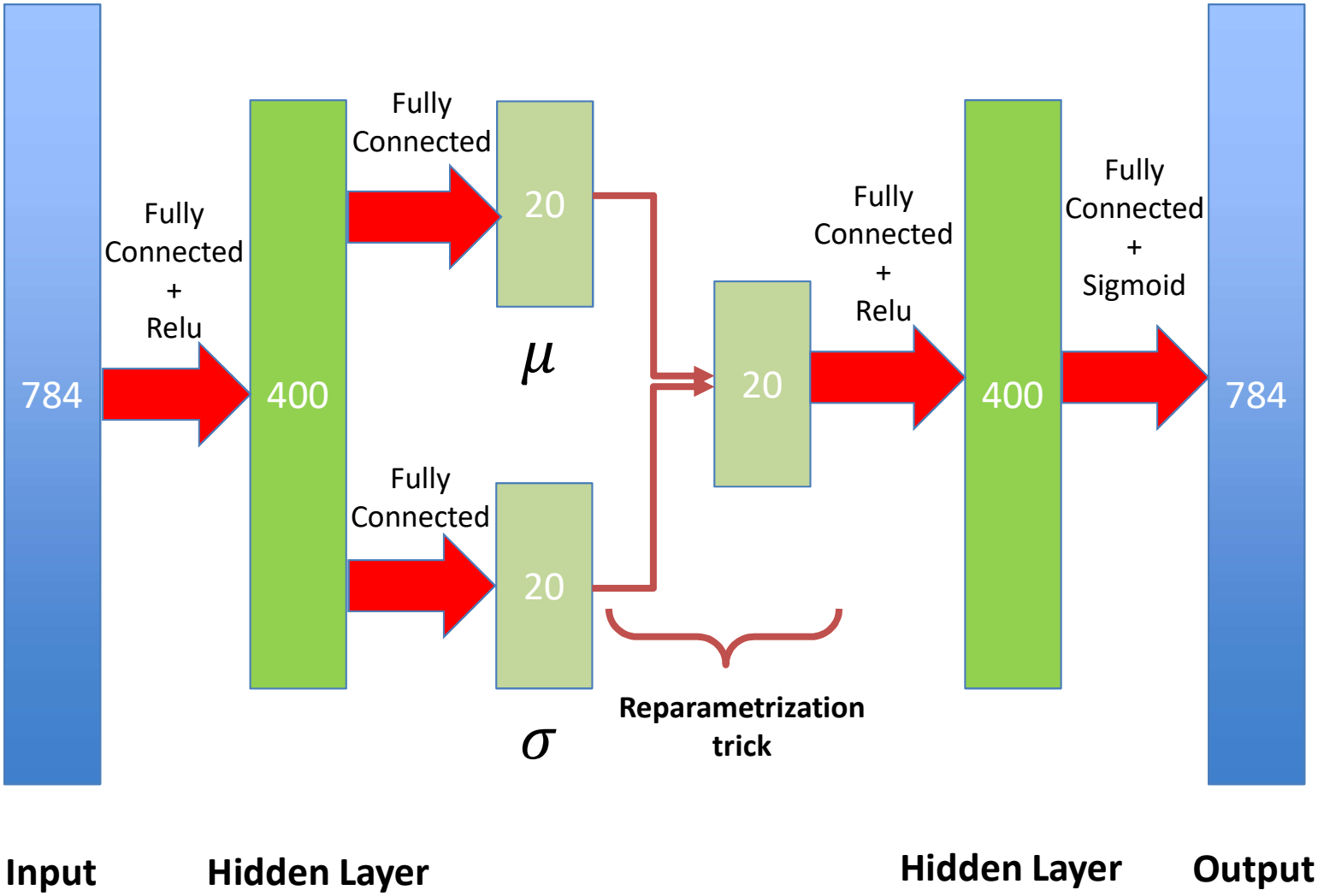
Optimizing using the second term (KL divergence) in the loss



Optimizing using both reconstruction loss (likelihood term) and KL divergence term in the loss

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

VAE Architectures for MNIST



MNIST Experiment

Reconstruction

1st epoch

5th epoch

10th epoch

- Adam optimizer, learning rate=0.001, batch size = 128, 10 epochs, no image normalization
- Reconstruction of images in the output of VAE in different epochs

MNIST Experiment

Sampling (generating)

1st epoch

5th epoch

10th epoch

- Generating of images in the output of VAE in different epochs by sampling a fixed random vector $Z \in \mathbb{R}^{128\times 20}$ $with\ Z_i \sim \mathcal{N}(0, I_{20\times 20}),\quad i = 1,2,\dots,128$

Other Variants

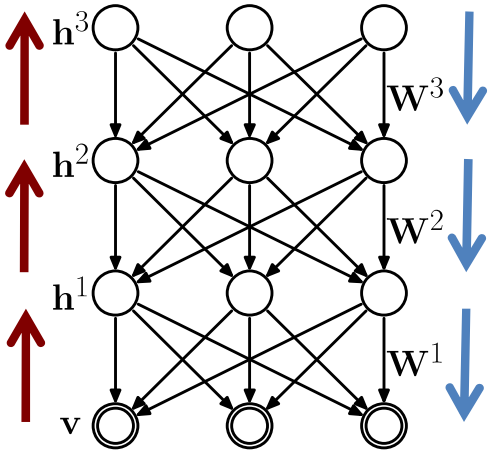
Importance Weighted Autoencoders

- Consider the following k-sample importance weighting of the log-likelihood ($z_i = h_i = h(\epsilon_i, \mathbf{x}, \lambda)$):

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})}\left[\log \frac{1}{k}\sum_{i=1}^{k} \frac{p(\mathbf{x}, \mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}\right]$$

$$= \mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})}\left[\log \frac{1}{k}\sum_{i=1}^{k} w_i\right]$$

Unnormalized importance weights



where $\mathbf{h}_1, \ldots, \mathbf{h}_k$ are sampled from the recognition network.

Importance Weighted Autoencoders

- Consider the following k-sample importance weighting of the log-likelihood ($z_i = h_i = h(\epsilon_i, \mathbf{x}, \lambda)$):

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})}\left[\log \frac{1}{k}\sum_{i=1}^{k}\frac{p(\mathbf{x},\mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}\right]$$

- This is a lower bound on the marginal log-likelihood:

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}\left[\log \frac{1}{k}\sum_{i=1}^{k} w_i\right] \leq \log \mathbb{E}\left[\frac{1}{k}\sum_{i=1}^{k} w_i\right] = \log p(\mathbf{x})$$

Calculate this

- Special Case of k=1: Same as standard VAE objective.

- Using more samples → Improves the tightness of the bound.

Tighter Lower Bound

- Using more samples can only improve the tightness of the bound.

- For all k, the lower bounds satisfy*:*

$$\log p(\mathbf{x}) \geq \mathcal{L}_{k+1}(\mathbf{x}) \geq \mathcal{L}_k(\mathbf{x})$$

- Moreover if $p(\mathbf{h}, \mathbf{x})/q(\mathbf{h}|\mathbf{x})$ is bounded, then:

$$\mathcal{L}_k(\mathbf{x}) \to \log p(\mathbf{x}), \quad \text{as} \quad k \to \infty$$

Computing the Gradients

- We can use the unbiased estimate of the gradient using reparameterization trick:

$$\nabla_{\theta,\lambda}\mathcal{L}_k(\mathbf{x}) = \nabla_{\theta,\lambda}\mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k\sim q(\mathbf{h}|\mathbf{x})}\left[\log\frac{1}{k}\sum_{i=1}^{k} w_i\right]$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_k}\left[\nabla_{\theta,\lambda}\log\frac{1}{k}\sum_{i=1}^{k} w(\mathbf{x}, h(\boldsymbol{\epsilon}_i, \mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\theta})\right]$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_k}\left[\sum_{i=1}^{k} \widetilde{w}_i \nabla_{\theta,\lambda}\log w(\mathbf{x}, h(\boldsymbol{\epsilon}_i, \mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\theta})\right]$$

- Where we define normalized importance weights:

$$\widetilde{w}_i = w_i / \sum_{i=1}^{k} w_i, \quad \text{where } w_i = \frac{p(\mathbf{x}, \mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}$$

IWAEs vs. VAEs

- Draw k-samples form the recognition network $q_\lambda(\mathbf{h}|\mathbf{x})$
  - or k-sets of auxiliary variables ϵ.

- Obtain the following Monte Carlo estimate of the gradient (IWAE):

$$\nabla_{\theta,\lambda}\mathcal{L}_k(\mathbf{x}) \approx \sum_{i=1}^{k} \widetilde{w}_i \nabla_{\theta,\lambda} \log w(\mathbf{x}, \mathbf{h}(\boldsymbol{\epsilon}_i, \mathbf{x}, \lambda), \boldsymbol{\theta})$$

- Compare this to the VAE's estimate of the gradient (VAE):

$$\nabla_{\theta,\lambda}\mathcal{L}(\mathbf{x}) \approx \frac{1}{k} \sum_{i=1}^{k} \nabla_{\theta,\lambda} \log w(\mathbf{x}, \mathbf{h}(\boldsymbol{\epsilon}_i, \mathbf{x}, \lambda), \boldsymbol{\theta})$$

Conditional VAE

Conditional VAE (CWAE)

- No control on the data generation process on VAE
  - e.g., We want to generate only a digit 2
- Conditioning all the distributions on what we want, the objective of CWAE (conditional ELBO):

$$E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\,\mathbf{z},\,\mathbf{c})- D_{kl}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x},\,\boldsymbol{c})\,||\,p(\boldsymbol{z}|\boldsymbol{c}))]$$

- $\boldsymbol{C}$ denotes the conditioning vector (e.g., a code for digit 2).

CWAE in Practice

- Conditioned image generation

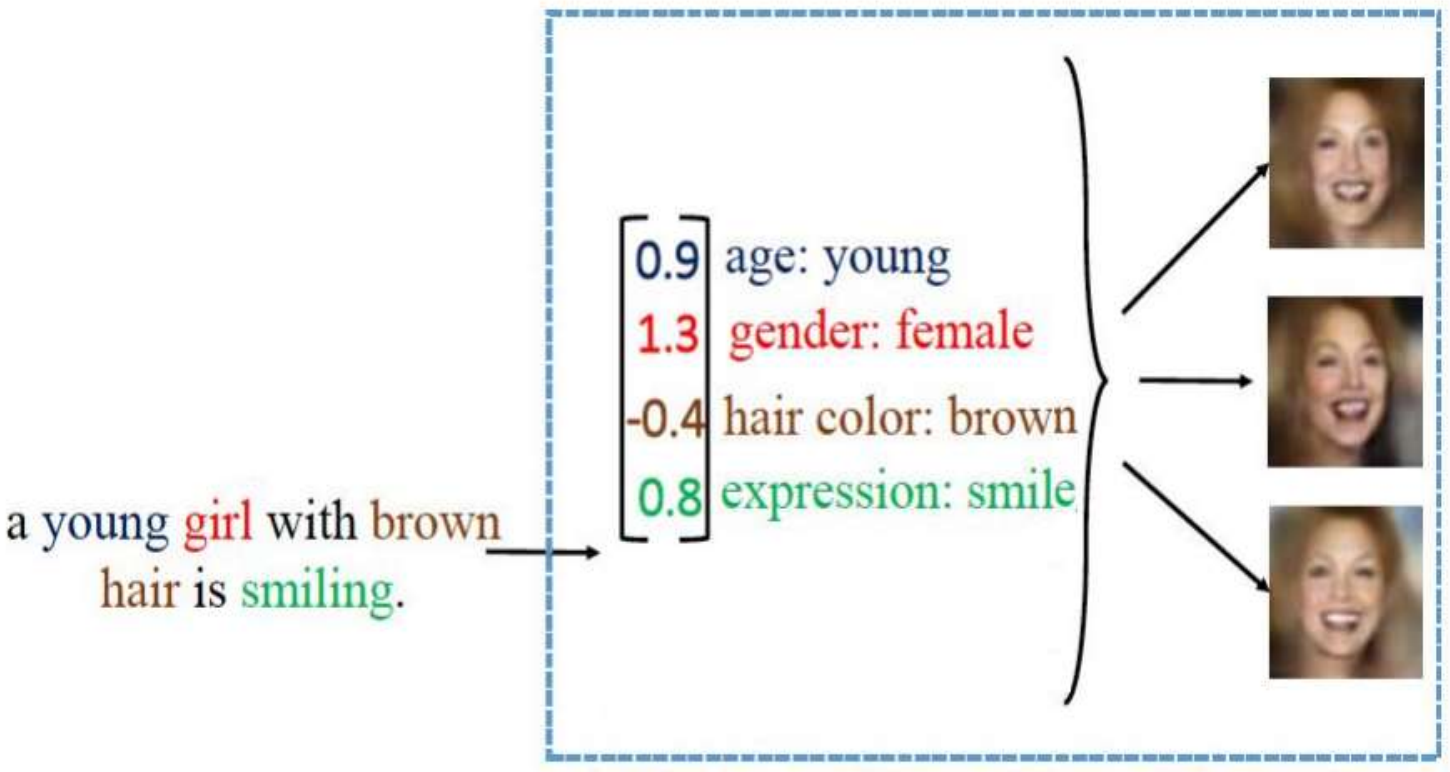


Image source: Yeh et al., lecture notes, 2017

β-VAE

Disentangled Latent Features—β-VAE

- A latent variable z is called disentangled factor if it is
  - Only sensitive to one single generative factor
  - Relatively invariant to other factors
- Advantage:
  - Interpretability
  - Easy generalization to a different task
- Example: Disentangled factors in Human face
  - Skin color
  - Hair length
  - Having glass or not

Disentangled Latent Features—β-VAE

- β-VAE was proposed by Higgins et al., 2017 to discover the disentangled latent representation:

$$\underset{\theta,\lambda}{\text{Max}}\, E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})]$$
$$s.t.\;\; D_{kl}(q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})) < \delta$$

- If β=1, this is regular VAE
- If $\beta > 1$, a stronger constraint on the latent factors; hence, limiting the representation capacity of z
- The larger β, the better discovering of disentangled latent features

Using Lagrangian form

$$\underset{\theta,\lambda}{\text{Max}}\, E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - \beta D_{kl}(q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z}))]$$

β-VAE in Practice

- β-VAE: $\beta = 250$
- VAE: $\beta = 1$

- Disentangled latent factors: Azimuth, Emotion, Hair
- β-VAE learns to **disentangle** factors
- VAE learns an **entangled** representation
  - ➢ Entangling azimuth with emotion
  - ➢ presence of glasses and gender

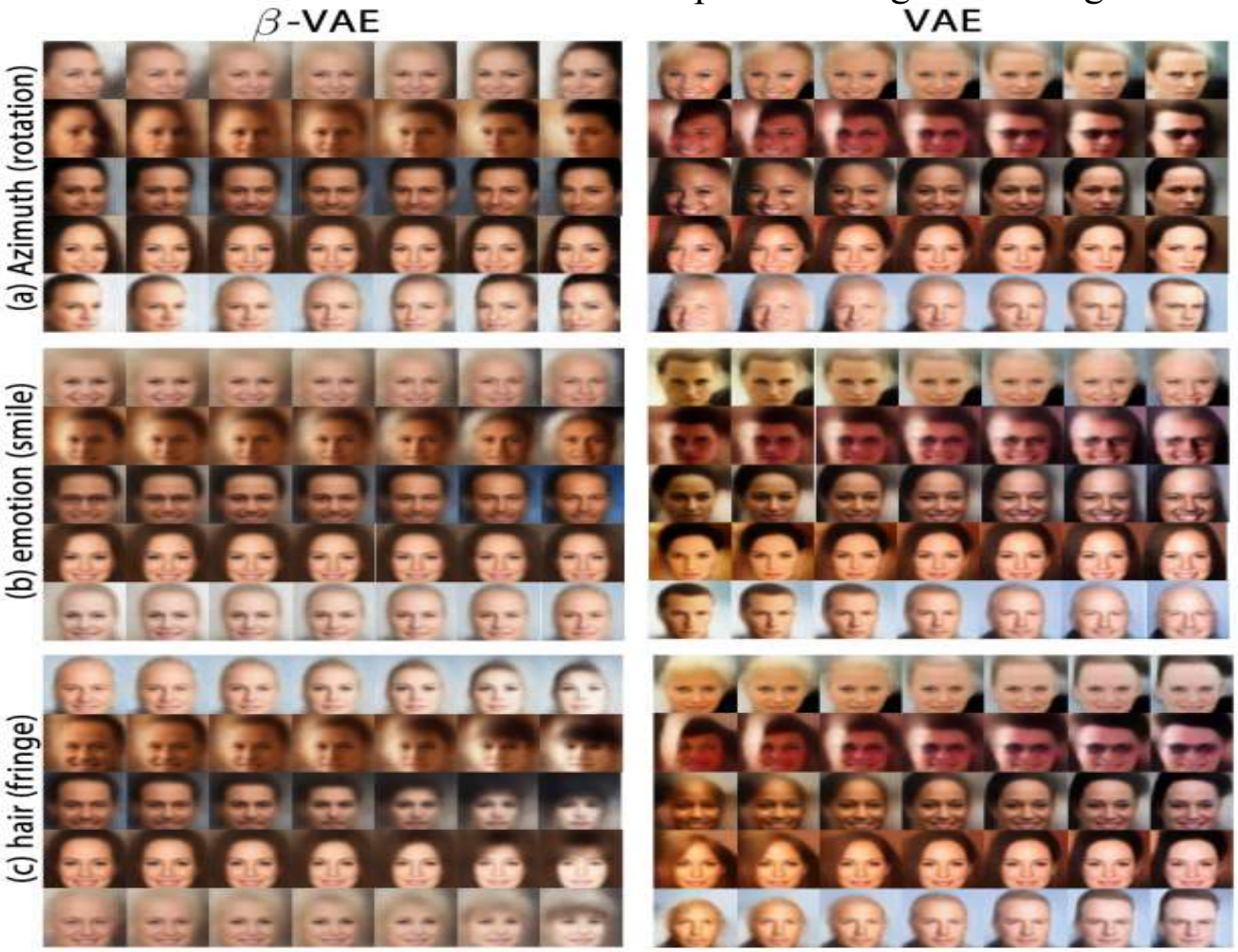


Image source: Higgins et al., 2017

β-VAE in Practice

- β-VAE: $\beta = 5$
- VAE: $\beta = 1$

- Disentangled latent factors: Azimuth, Chair Width, Leg Style
- β-VAE learns to **disentangle** factors
- VAE learns an **entangled** representation
  - ➢ Entangling chair width with azimuth and leg style

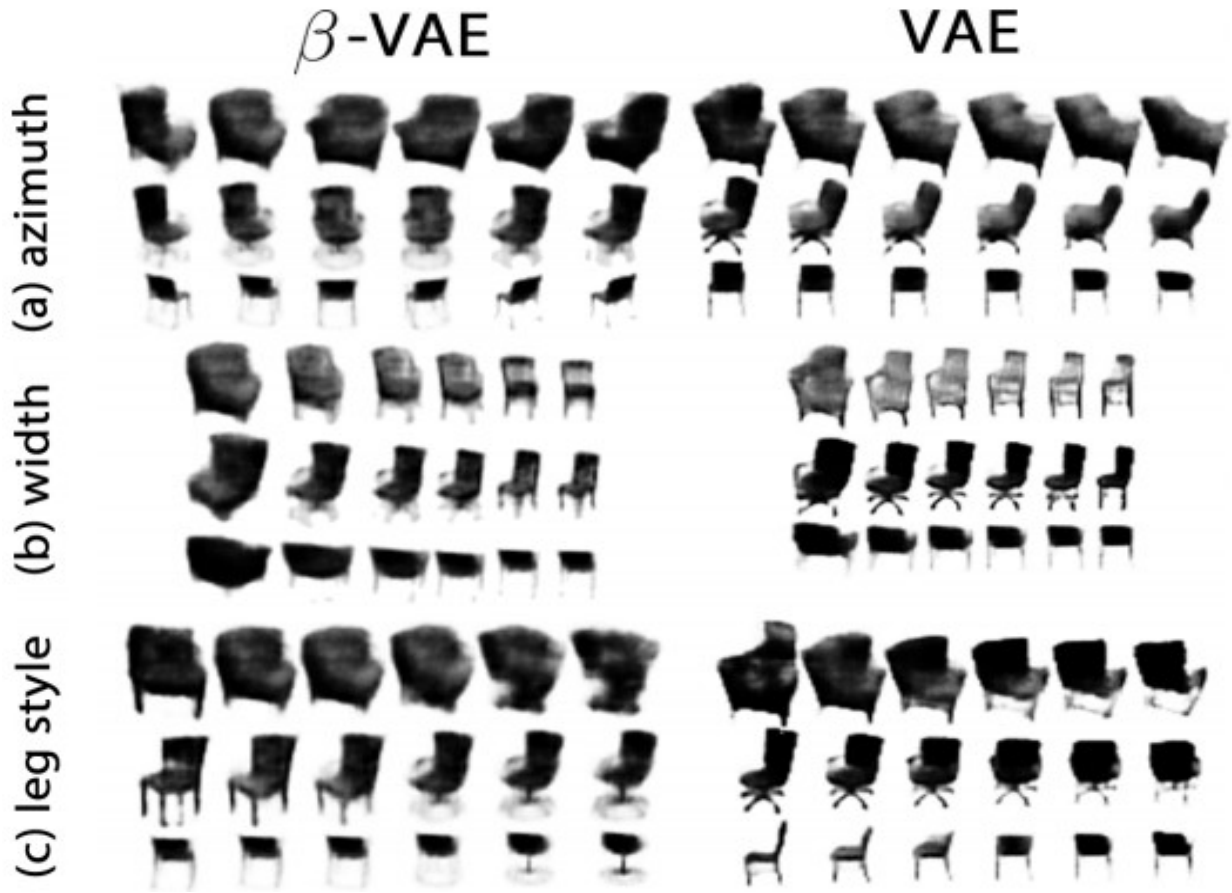
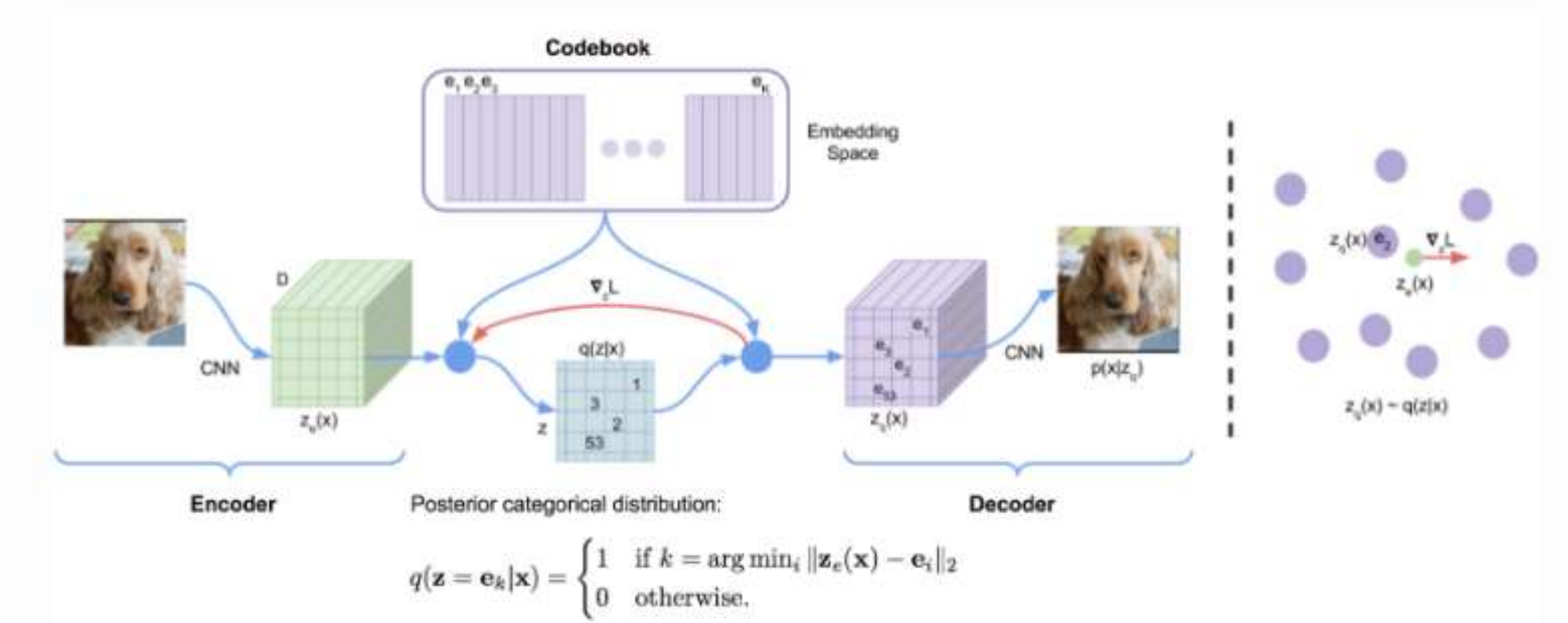


Image source: Higgins et al., 2017

Vector Quantized-VAE

Vector Quantized-Variational AutoEncoder (VQ-VAE)

- Learning a discrete latent variable by the encoder



$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

- Mapping K-dimensional vectors into a finite set of "code" vectors.
- Similar to K-means algorithm

Image source: van den Oord, et al. 2017

Generative Adversarial Networks
GANs

Vahid Tarokh

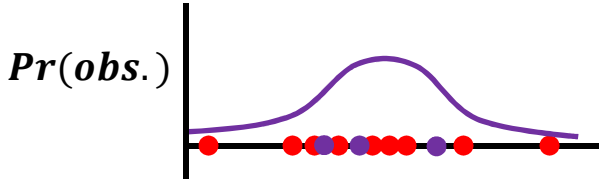
ECE 685D, Fall 2025

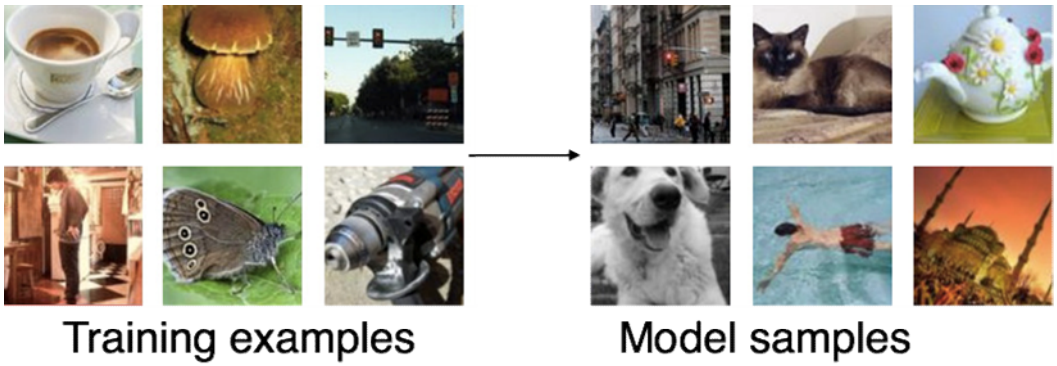
Probabilistic Generative Models

Density Estimation

$$Pr(observation) \sim Pr(synthetic\ obs.)$$

Synthesizing Examples From Probabilistic Generative Model





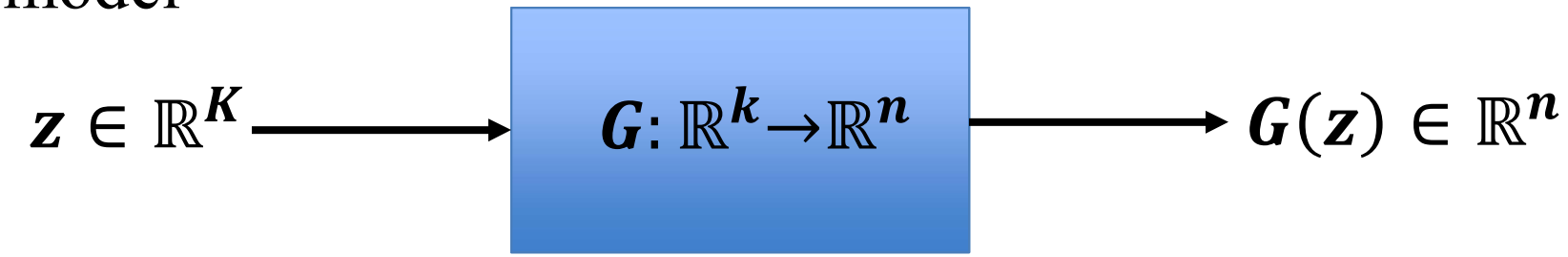
Goodfellow NIPS tutorial and accompanying paper (arXiv:1701.00160v4 [cs.LG]) provided some figures

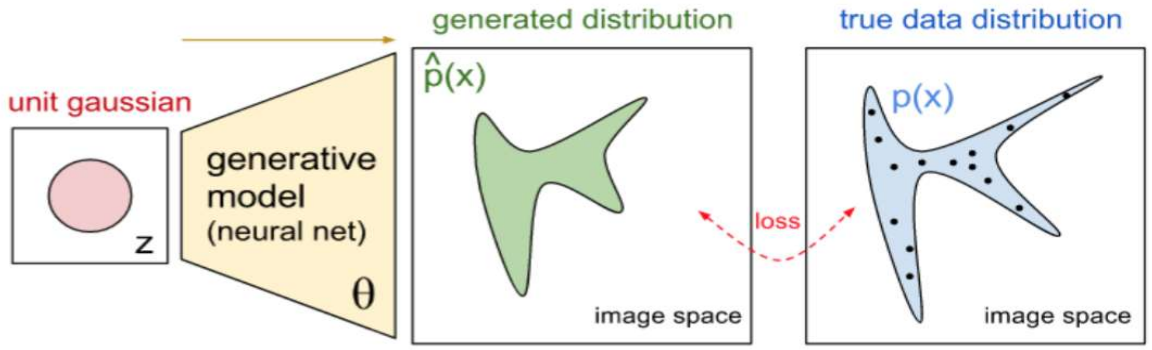
Density function $Pr_{\text{model}}(\boldsymbol{x}|\boldsymbol{\theta})$

- Explicit and analytical
  - Gaussian
  - Can sample directly from model
- Explicit and approximate
  - Boltzmann machine
  - VAE
  - Normalizing Flow
  - Autoregressive models
- Implicit
  - GAN
  - Can't estimate probability but can draw from distribution with given probability

Implicit Generative Models

- Sampling a latent code (vector) z from a fixed and simple distribution such Gaussian
- Passing the drawn code to a differentiable Generator model





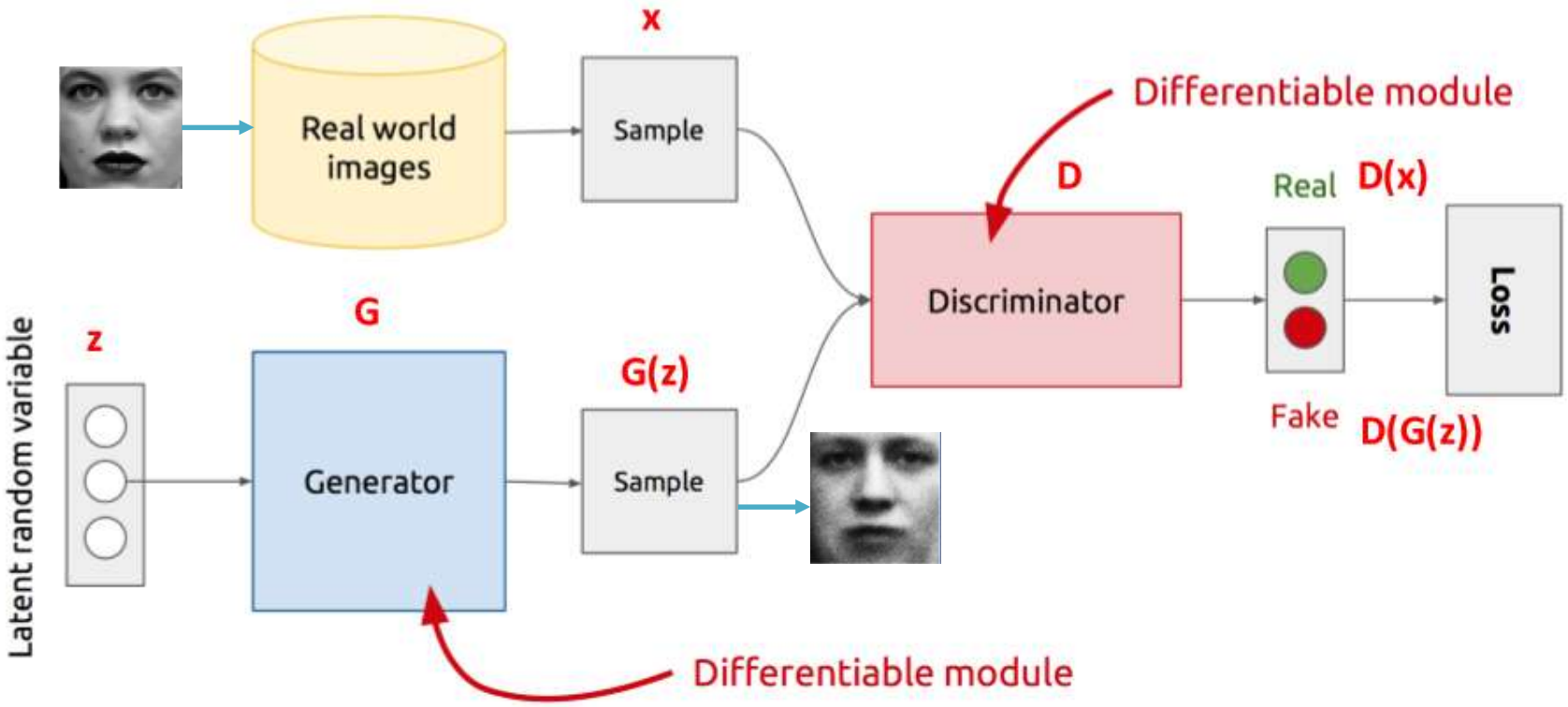
https://blog.openai.com/generative-models/

Why GANs?

- VAE are known to generate blurry images
  - Not exact inference (Variational inference)
  - Maximizing a lower bound

- GANs revolutionized generative modeling by producing crisp, high-resolution images
  - Sampling (or generation) is straightforward
  - Training doesn't involve Maximum Likelihood estimation
  - Robust to overfitting since Generator never sees the training data
  - Empirically, GANs are good at capturing the modes of the distribution
  - Generate samples through a competition between two-players (a pair of generator and discriminator)
  - It is not known how well they're modeling the distribution
  - No clear way to tell if they are dropping important modes from the distribution

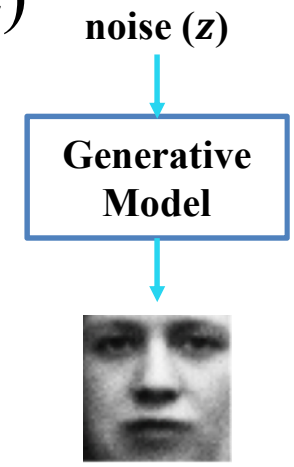
General GAN's Architecture



- **Generator:** generate fake samples, tries to fool the Discriminator
- **Discriminator**: tries to distinguish between real and fake samples
- Training Generator and Discriminator against each other
- z is some **random noise** (Gaussian/Uniform)

Generative Model

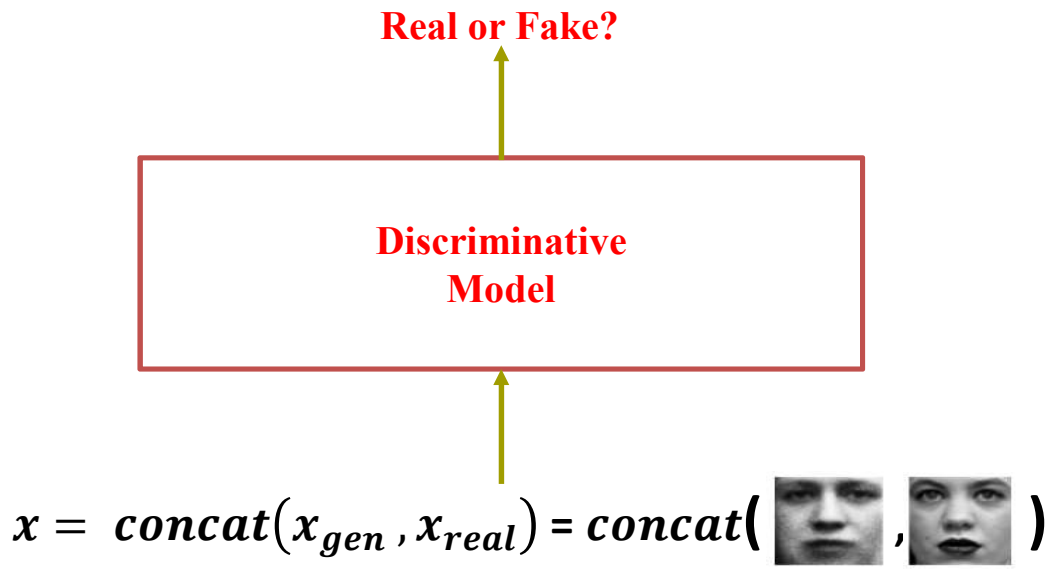
- How to make it generate different samples?
  - ➢Input to model is noise (latent/hidden code)



- Generative model as a neural network
  - ➢Computes $\boldsymbol{x_{gen} = G(z|\theta)}$
  - ➢Differentiable
  - ➢Does not have to be invertible
  - ➢$\boldsymbol{z}$ typically has lower dimension than $\boldsymbol{x_{gen}}$

Discriminative Model

- Think of it as a critic
  - A good critic can tell real from fake
- Discriminative model as a neural net
  - differentiable
  - computes $0 \leq \boldsymbol{D}(\boldsymbol{x}) \leq 1$, with value 1 if real, 0 if fake

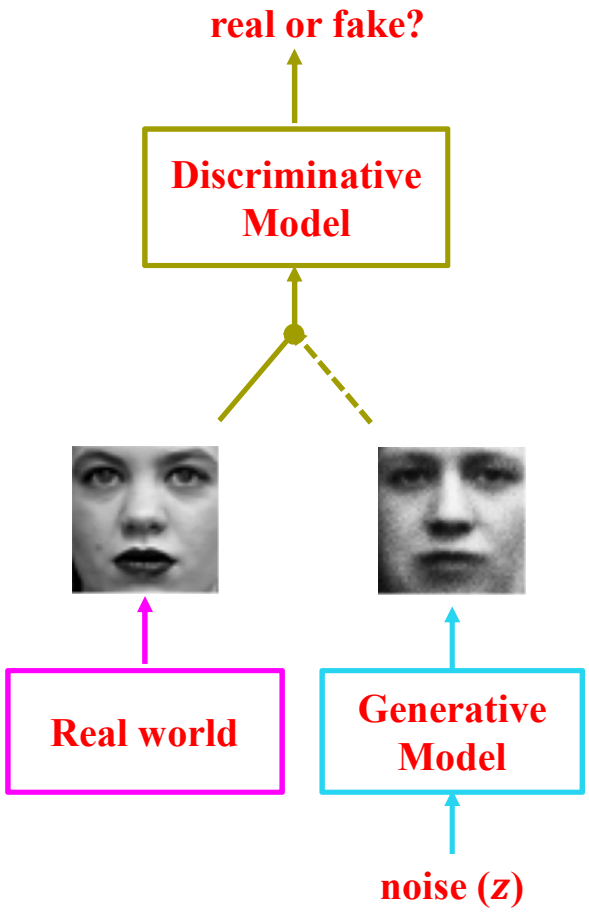


Training Procedure: Basic Idea

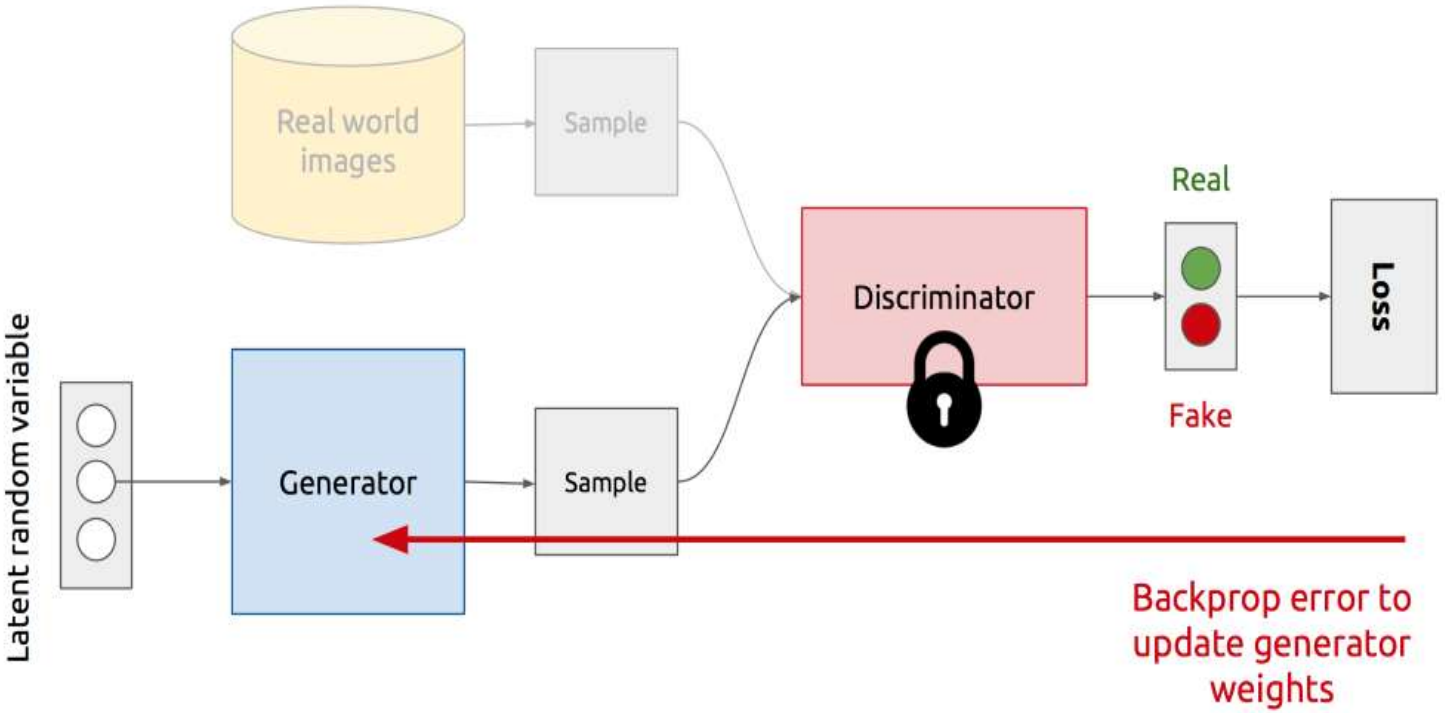
- G tries to fool D
- D tries not to be fooled by G

- Models are trained **simultaneously**
  - ➢ As G gets better, D has a more challenging task
  - ➢ As D gets better, G has a more challenging task

- Ultimately, we don't care about the D
  - ➢ Its role is to force G to work harder

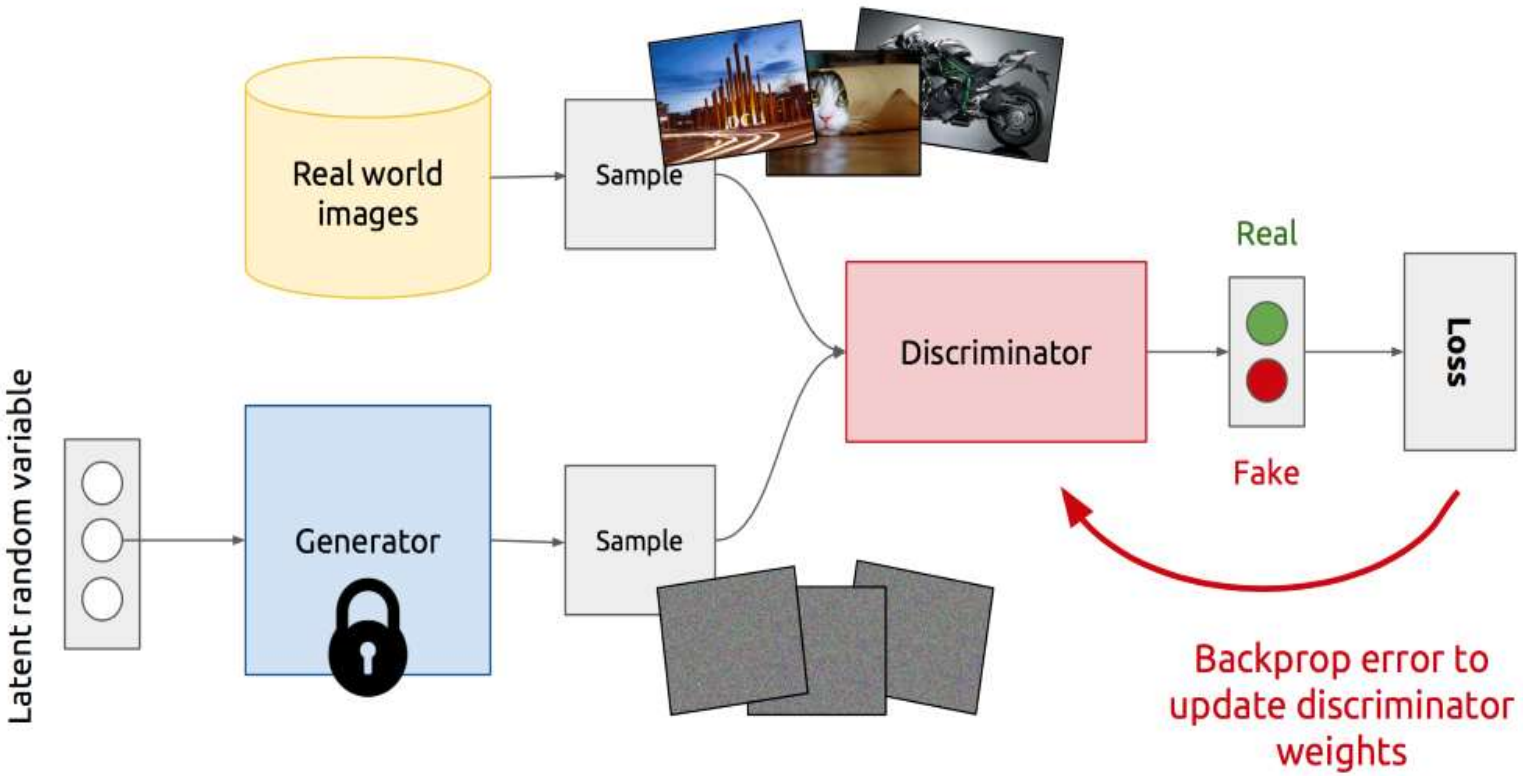


Training Generator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Training Discriminator



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Generative models as distance/divergence minimization

- **KL-divergence (asymmetric):**

$$KL(P_{model} \,||\, P_{real}) = \int P_{model}(x) log \frac{P_{model}(x)}{P_{real}(x)} dx$$

$$KL(P_{model} \,||\, P_{real}) = 0 \;\Rightarrow\; P_{model} = P_{real}$$

- **Jensen Shannon divergence (symmetric):**

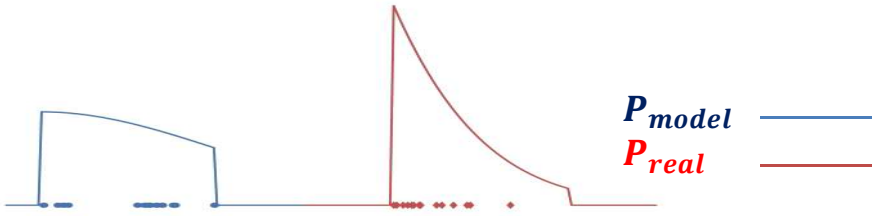
$$JSD(P_{model} \,||\, P_{real}) = \frac{1}{2} KL\left(P_{real} \,||\, \frac{P_{real}+P_{model}}{2}\right) + \frac{1}{2} KL\left(P_{model} \,||\, \frac{P_{real}+P_{model}}{2}\right)$$

$$JSD(P_{model} \,||\, P_{real}) = 0 \;\Rightarrow\; P_{model} = P_{real}$$

- **No signal is learned from KL or JSD divergence if non overlapping support between the data and the model.**

$$KL(P_{model} \,||\, P_{real}) = \infty \qquad JSD(P_{model} \,||\, P_{real}) = log2$$

NOTE: The generator is minimizing the *Jensen Shannon divergence* between the real and generated (model) distributions.



https://storage.googleapis.com/deepmind-media/UCLxDeepMind_2020/L9%20-%20UCLxDeepMind%20DL2020.pdf

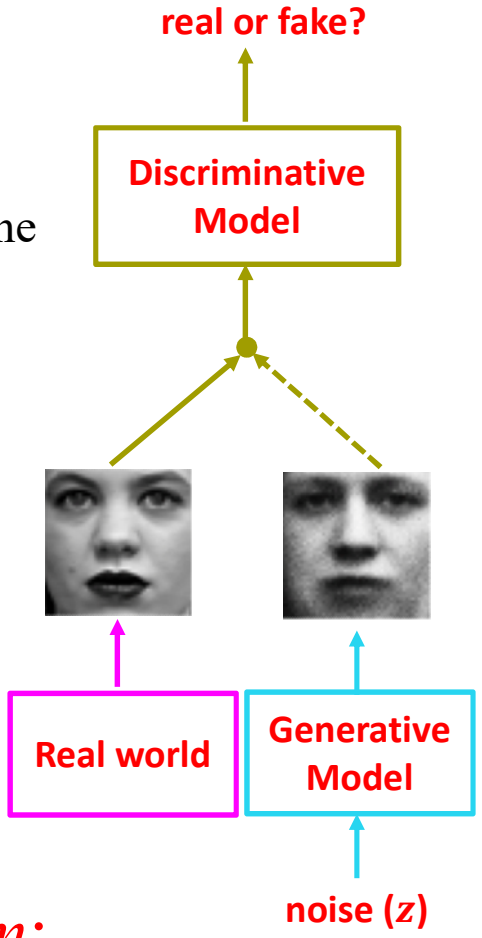
Loss Functions in Vanilla GAN

- We discuss only the parametric case
- Loss function for D
  - Maximizing the likelihood such that the model says 'real' to the samples from the world and 'fake' to the generated samples

$$V(D,G) = \mathbb{E}_{x\sim \text{real}} \log D_{\theta_d}(x) + \mathbb{E}_z \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)$$

log-probability that D correctly predicts real data x are real

log-probability that D correctly predicts generated data $G(z)$ are generated



  - The Discriminator is trying to **maximize** its reward.
  - The Generator is trying to **minimize** Discriminator's reward (or maximize its loss).

- Known as a ***minimax*** *optimization problem*:

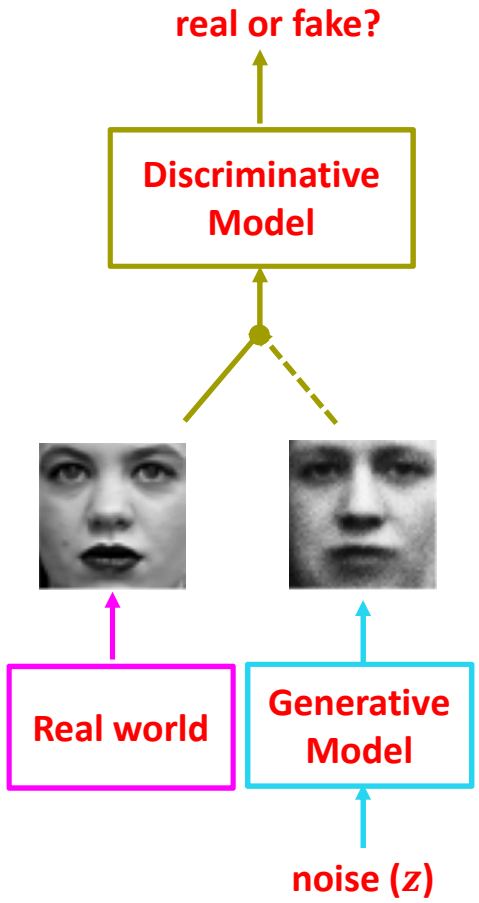
$$\min_{\theta_g} \max_{\theta_d} V(D,G) = \mathbb{E}_{x\sim real} \log D_{\theta_d}(x) + \mathbb{E}_z \log\left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)$$

$\boldsymbol{\theta_d}$: parameters of Discriminator $\boldsymbol{\theta_g}$: parameters of Generator

Training Procedure

- Train both models simultaneously via stochastic gradient descent using mini-batches consisting of
  - Some generated samples
  - Some real-world samples

- D can be trained without altering G, and vice versa
  - May want multiple training epochs of just D so it can stay ahead
  - May want multiple training epochs of just G because it has a harder task



Optimization by Alternative Gradient Descent 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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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:

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

end for

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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.

Discriminator updates

$k = 1$ may result more stability, others use $k > 1$, no best rule.

Generator updates

Vanishing Gradient

- Saturation problem in $\mathbb{E}_z \log\left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)$
- If the generated sample is bad, the discriminator's prediction **is close to 0**, and the generator's cost is flat

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

$$V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

$$\nabla_{\theta_G} V(D,G) = \nabla_{\theta_G} \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

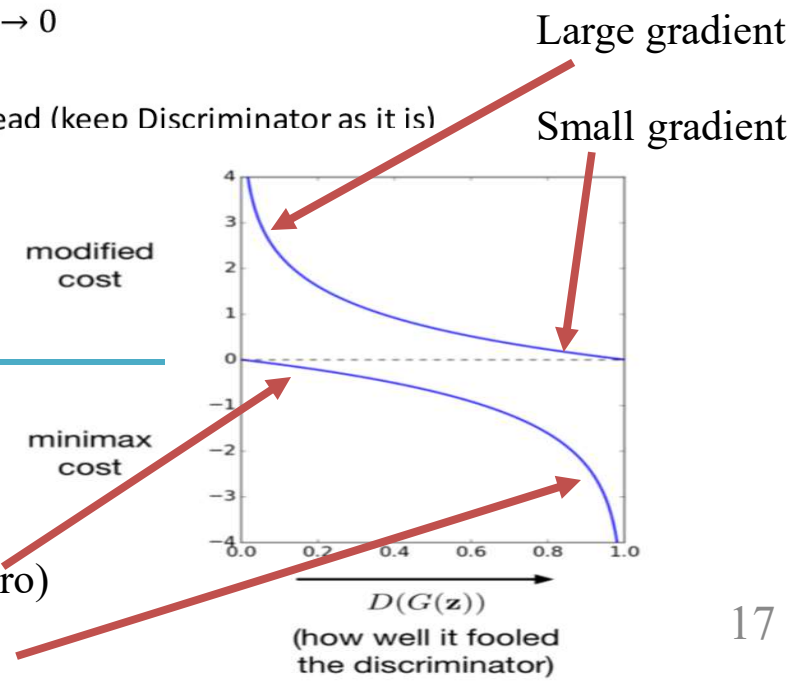
- $\nabla_a \log\left(1-\sigma(a)\right) = \frac{-\nabla_a \sigma(a)}{1-\sigma(a)} = \frac{-\sigma(a)\,(1-\sigma(a))}{1-\sigma(a)} = -\sigma(a) = -D\left(G(z)\right)$
- Gradient goes to 0 if D is confident, i.e. $D\left(G(z)\right) \to 0$
- Minimize $-\mathbb{E}_{z\sim q(z)}\left[\log D\left(G(z)\right)\right]$ for **Generator** instead (keep Discriminator as it is)

➢ Modified generator cost:

$$\mathbb{E}_z - \log\left(D_{\theta_d}(G_{\theta_g}(z))\right)$$

➢ Original minimax cost:

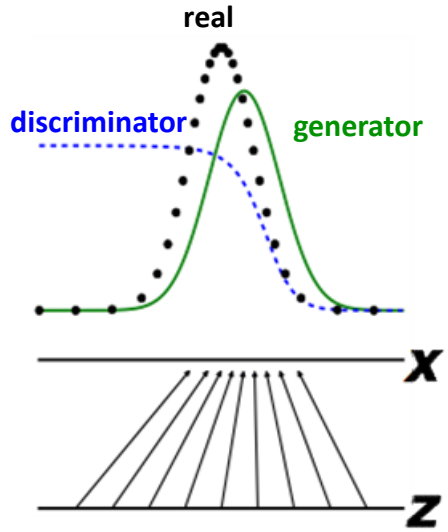
$$\mathbb{E}_z \log\left(1 - D_{\theta_d}(G_{\theta_g}(z))\right)$$



Global Optimality of $Pr(x|real) = Pr(x|synthesized)$ (Nash Equilibrium)

Recall the minimax loss:

$$V(D,G) = \mathbb{E}_{x\sim real}\, log\, D_{\theta_d}(x) + \mathbb{E}_z\, log\left(1 - D(G_{\theta_g}(z))\right)$$



Proposition. For G fixed, the optimal discriminator D_G^* is given by

$$D_G^*(x) = \frac{Pr(x|\text{real})}{Pr(x|\text{real}) + Pr(x|\text{synthesized})}$$

Theorem. The global minimum of the training criterion, $max_D\, V(D,G)$ is achieved if and only if $Pr(x|real) = Pr(x|synthesized)$.

NOTE:

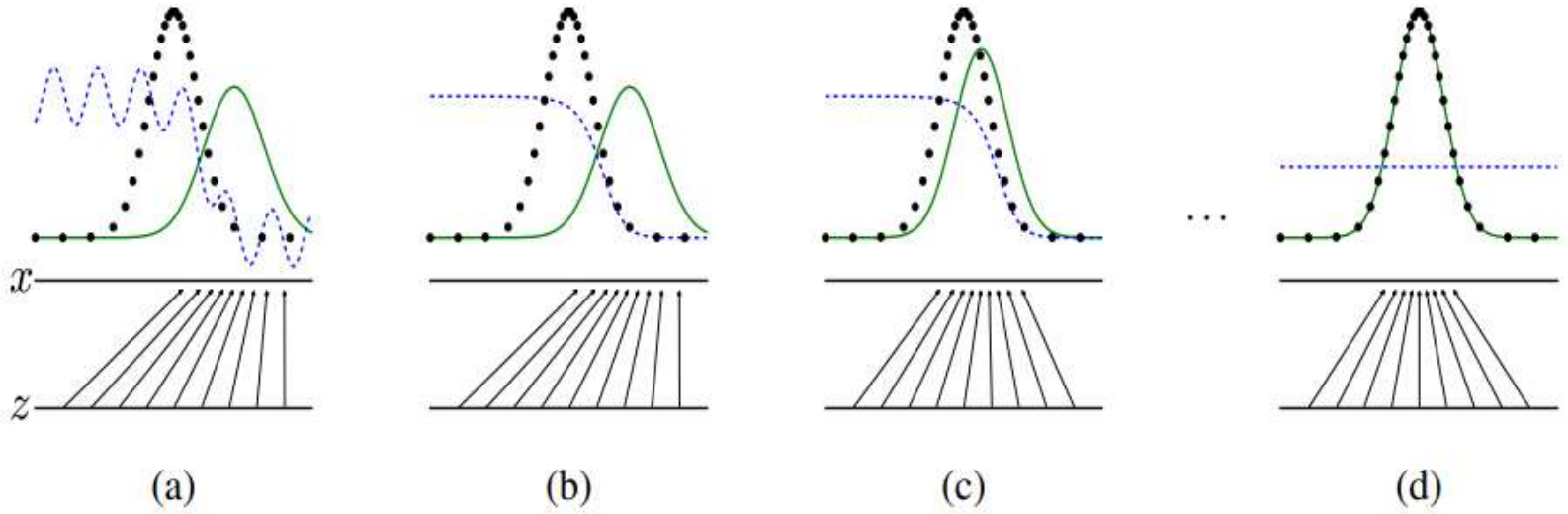
$max_D = max_{\theta_d}$: maximizing over the parameters of discriminator

- **Jointly training two networks is difficult.**
- **It can be unstable.**
- **Choosing objectives with better loss landscapes helps training.**

Generative Adversarial networks, Goodfellow et al., 2014

Towards Nash Equilibrium

Solid green: generator
Dashed blue: discriminator
Dotted black: real samples



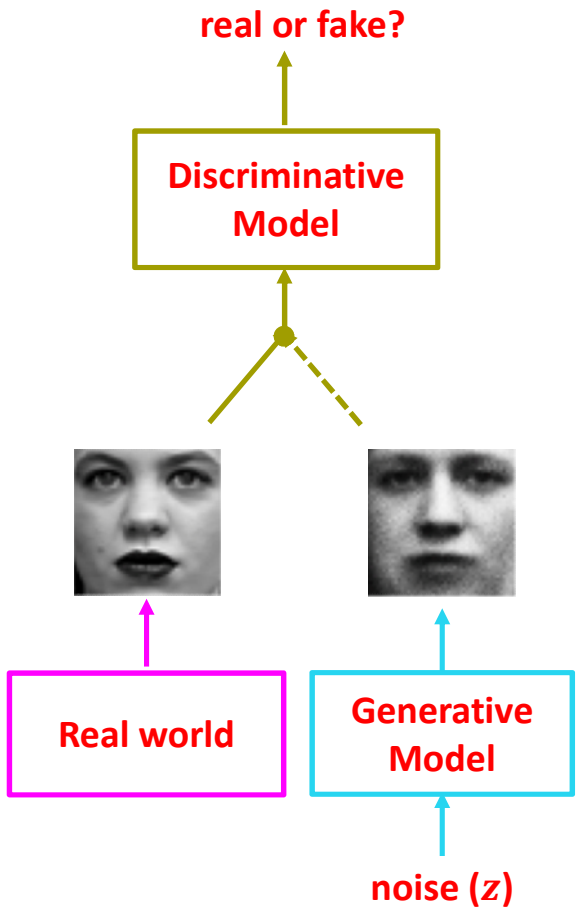
a) Near convergence. $Pr(x|synthesized)$ is similar to $Pr(x|real)$ and D is a partially accurate classifier.
b) D is trained to discriminate samples from data
c) After update to G, gradient of D has guided $G(z)$ to flow to regions that are more likely to be classified as data
d) Reaching an equilibrium point at which both distribution cannot improve because as (with enough capacity of D and G):

$$\boldsymbol{Pr(x|synthesized) = Pr(x|real)}$$

Goodfellow et al., 2014

Three Intuitive Reasons that GANs Work

- G has a "forced" learning task
  - ➢ It knows when it does good (i.e., fools D) but it is not given a supervised signal
  - ➢ Backprop through D provides G with a supervised signal; the better D is, the better this signal will be
- Can't describe optimum via a single loss
  - ➢ Will there be an equilibrium?
- D is seldom fooled
  - ➢ But G still learns because it gets a gradient telling it how to change in order to do better the next round



Optimal and Non-Optimal Discriminator

- **If the discriminator (D) is optimal:**
  - The generator is minimizing the *Jensen Shannon divergence* between the real and generated (model) distributions.

$$\boldsymbol{min_{\theta_g}\, max_{\theta_d} V(D,G) = \mathbb{E}_{x\sim \text{real}}\, log\, D_{\theta_d}(x) + \mathbb{E}_z \log\left(1 - D_{\theta_d}\left(G_{\theta_g}(z)\right)\right)}$$

- **However, D is not optimal in practice**
  - We have access to the limited computational resources.
  - Optimization problems are non-convex.
  - W do not see the real data distribution (only samples)
  - May not learn anything if there is no overlapping support between the data and the model.
    - The theory assumes that $\boldsymbol{P_{model}(x)\ and\ P_{real}(x)}$ are non-zero everywhere.
    - No true if we have data lying on a manifold.
  - The theory assumes that the optimal discriminator is unique. In practice other discriminators can do nearly as good a job: i.e. the discriminator can overfit the data.

Problems with (Vanilla) GANs

- Estimating probability distribution is Implicit
  - Not straightforward to compute
  - Vanilla GANs may only be good for Sampling/Generation

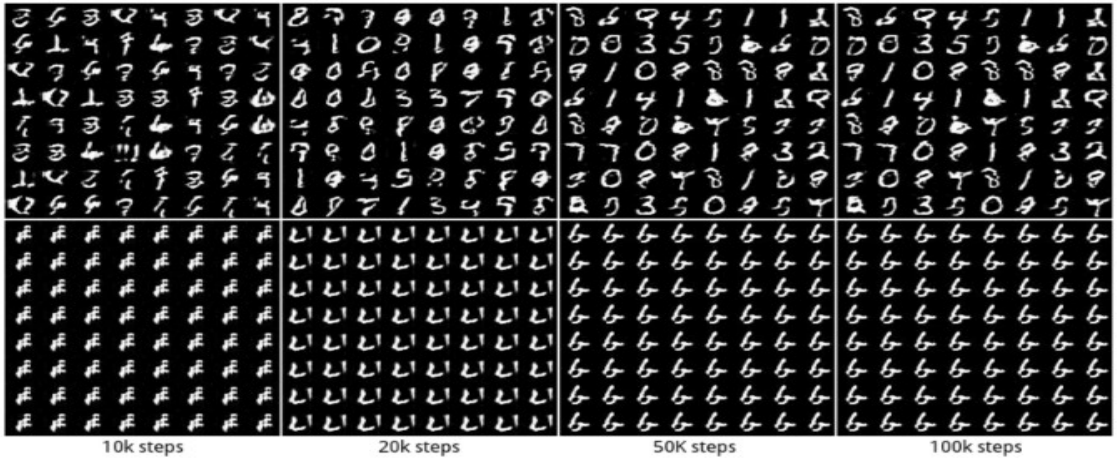
- Training is Hard (Instable)
  - Non-Convergence
  - Mode-Collapse
  - Vanishing Gradient

Problems with (Vanilla) GANs

- Non-Convergence Issue
  - ❑ Training GANs involves minimax optimization
    - ➢ SGD was not designed to find the Nash equilibrium of a game
    - ➢ May not converge to the Nash equilibrium at all
- Mode-Collapse
  - ❑ Only a few modes of data are generated

No mode Collapse

Mode Collapse



- ❑ Class leakage from a partially trained BigGAN, showing a cross between a tennis ball and perhaps a dog

https://arxiv.org/pdf/1809.11096.pdf

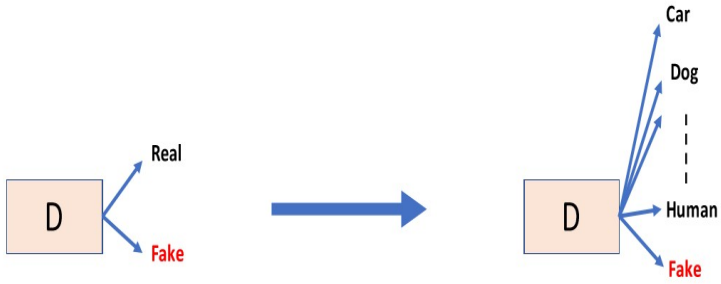
Mode-Collapse (more example)

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

Some Solutions to Partially Prevent Mode-Collapse

- Mini-Batch GANs
  - ➢ Extract features that capture diversity in the mini-batch
  - ➢ Feed those features to the discriminator along with the image

- Supervision with labels
  - ➢ Similar to conditional GANs



Some Solutions to Partially Prevent Mode-Collapse

- Changing the loss function
  - ➢ Some choices have better behavior (more stable) during training
  - ➢ Some choices will modify the latent space
    - WGAN (Wasserstein-GAN)
    - f-GAN
    - MMD-GAN
    - Fisher-GAN
- Constraining the discriminator model during the training process

https://arxiv.org/pdf/1611.02163.pdf
https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2015/slides/lec13.GAN.pdf

WGAN (Wasserstein GAN)

- If data comes from a low-dimensional manifold of a high dimensional space
  - ➢ Negligible intersection between the model's manifold
    - KL divergence is undefined or infinite

- The loss function and gradients may not be continuous and well behaved

- The Earth Mover's Distance is well defined:
  - ➢ Minimum transportation cost for making one pile of dirt (pdf/pmf) look like the other

WGAN (Wasserstein GAN)

- **The Earth-Mover (EM) distance or Wasserstein-1**

$$inf_{\gamma \in \Pi(\mathrm{P_{real}}, \mathrm{P}_{model})} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|_1$$

- $\Pi(P_{real}, P_{model})$ denotes the set of all joint distributions γ(x, y) whose marginals are respectively P_{real} and P_{model}.

- Intuitively, γ(x, y) indicates how much "mass" must be transported from x to y in order to transform the distributionsP_{real}into the distribution P_{model}. https://yoo2080.wordpress.com/2015/04/09/introduction-to-wasserstein-metric-earth-movers-distance/

- **Using Kantorovich-Rubinstein duality**

$$sup_{||f||_L \leq 1} \mathbb{E}_{x\sim P_{real}} f(x) - \mathbb{E}_{x\sim P_{model}} f(x)$$

- The EM distance then is the "cost" of the optimal transport plan.

  - Sup is taken over w.r.t. 1-Lipschitz continuous
  - f is a discriminator belonging to the family of 1-Lipschitz

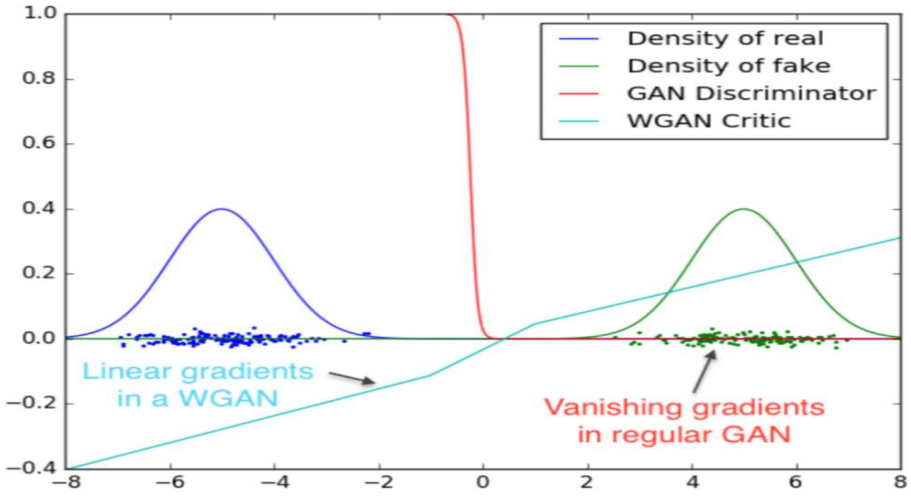
Arjovsky et al., "WGAN", 2017.

WGAN (Wasserstein GAN)

- **Training loss:**

$$\min_{G} \sup_{\|D\|_L \leq 1} \mathbb{E}_{x \sim P_{real}} D(x) - \mathbb{E}_z D(G(z))$$

- The Discriminator is trained for multiple steps before the Generator is updated
- To ensure $D(x)$ has the Lipschitz continuity (i.e., $||D||_L \leq 1$) ***gradient-clipping*** is used in the discriminator.
- It is said that WGAN resolves many training issues such as mode collapse.



https://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2015/slides/lec13.GAN.pdf

Enforcing the Lipschitz Constraint

1. **WGAN + Gradient-clipping**
2. **WGAN + Gradient Norm Penalty Regularizer**

$$\mathbb{E}_{\bar{x}\sim P_{real}} D(\bar{x}) - \mathbb{E}_{x\sim P_{model}} D(x) + \lambda \mathbb{E}_{P_{\hat{X}}} \left(\left\| \nabla_{\hat{X}} \mathrm{D}(\hat{X}) \right\|_2 - 1 \right)^2$$

- Gradient Norm Penalty Regularizer forces the Lipschitz Constraint.
- $P_{\hat{X}}$ samples uniformly along straight lines between pairs of points sampled from the data distribution P_{real} and the generator distribution P_{model}.
- This is motivated by the fact that the optimal critic (Discriminator) contains straight lines with gradient norm 1 connecting coupled points from $P_{\hat{X}}$ and P_{model}.
- Enforcing the unit gradient norm constraint everywhere is intractable.
- Enforcing only along these straight lines is sufficient and experimentally has good performance.

https://arxiv.org/pdf/1704.00028.pdf

Enforcing the Lipschitz Constraint

3. WGAN + Spectral Normalization

- The weights of the generator are normalized using spectral normalization.
- Zhang et al. (2018) (SAGAN)) found that employing Spectral Normalization in Generator improves stability, allowing for fewer Discriminator steps per iteration.
- Controlling the Lipschitz constant of the Discriminator function D (with L layers) by constraining the spectral norm of each layer $g: h_{in} \to h_{out}$
- Define the Lipschitz norm of *layer g* by $\|g\|_{Lip} = \sup_h \sigma(\nabla_{\mathrm{h}}(g))$, where $\sigma(A)$ denotes the spectral norm of the matrix A.
- Using the facts that $\|g_1 \circ g_2\|_{Lip} \leq \|g_1\|_{Lip} \, \|g_2\|_{Lip}$, and for many activation functions (e.g., Relu), the Lipschitz constant is 1, we have:

$$\|G\|_{Lip} \leq \prod_{l=0}^{L} \sigma(W^l),$$

- W^l denotes the weights of the l^{th} layer
- By normalizing the weights of each layer, $\overline{w_{SN}^l} = \dfrac{W}{\sigma(W^l)}$, we guarantee that $\|G\|_{Lip} \leq 1$.

Enforcing the Lipschitz Constraint

3. WGAN + Spectral Normalization

- Very computationally cheap even in comparison to the calculation of the forward and backward propagations on neural networks

Algorithm 1 SGD with spectral normalization

- Initialize $\tilde{\boldsymbol{u}}_l \in \mathcal{R}^{d_l}$ for $l = 1, \ldots, L$ with a random vector (sampled from isotropic distribution).
- For each update and each layer l:
  1. Apply power iteration method to a unnormalized weight W^l:

$$\tilde{\boldsymbol{v}}_l \leftarrow (W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l / \|(W^l)^{\mathrm{T}} \tilde{\boldsymbol{u}}_l\|_2$$
$$\tilde{\boldsymbol{u}}_l \leftarrow W^l \tilde{\boldsymbol{v}}_l / \|W^l \tilde{\boldsymbol{v}}_l\|_2$$

  2. Calculate $\bar{W}_{\mathrm{SN}}$ with the spectral norm:

$$\bar{W}^l_{\mathrm{SN}}(W^l) = W^l / \sigma(W^l), \text{ where } \sigma(W^l) = \tilde{\boldsymbol{u}}_l^{\mathrm{T}} W^l \tilde{\boldsymbol{v}}_l$$

  3. Update W^l with SGD on mini-batch dataset $\mathcal{D}_M$ with a learning rate α:

$$W^l \leftarrow W^l - \alpha \nabla_{W^l} \ell(\bar{W}^l_{\mathrm{SN}}(W^l), \mathcal{D}_M)$$

https://arxiv.org/pdf/1802.05957.pdf

Evaluation of Generated Samples

- **Inception Score (IS)**
  - Intuitively, the conditional label distribution of samples containing meaningful objects should have **low entropy**, and the variability of the samples should be high:

$$IS = e^{KL(P(y|x)||P(y))}$$

    - Applying trained Inception-V3 model with ImageNet to every generated image to get $P(y|x)$
    - **The larger the IS, the better generated image**
    - **Advantage**: Well correlated with scores from human annotators
    - **Disadvantage**: Insensitivity to the prior distribution over labels and not being a proper distance

https://openreview.net/pdf/6e08ef84b70c2be77e96f8e15ebfb24506482a19.pdf

Evaluation of Generated Samples

- **Fréchet Inception Distance (FID)**
  - Comparing the statistics of generated samples to real samples
  - Samples from P_{real} and P_{model} are first embedded into a feature space (the 2048-dimensional activations of the Inception-v3 pool3 layer).
  - By assuming the multivariate Gaussian distribution on the embedded data, the mean and covariance are estimated:

$$X_{real}^{em} \sim \mathcal{N}(\mu_1, \Sigma_1) \qquad X_{model}^{em} \sim \mathcal{N}(\mu_2, \Sigma_2)$$

  - The Fréchet distance between the two Gaussians is given by:

$$FID = \|\mu_1 - \mu_2\|_2^2 + Tr\left(\Sigma_1 + \Sigma_2 - 2(\Sigma_1\Sigma_2)^{\frac{1}{2}}\right)$$

  - **The lower FID, the more similar real and generated samples**

The GAN ZOO

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

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

Different GANs

- Since GANs were introduced in 2014, there have been hundreds of papers introducing various architectures and training methods.

- Most modern architectures are based on the Deep Convolutional GAN (DCGAN), where the generator and discriminator are both conv nets.

GAN Samples

Celebrities:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

GAN Samples

Bedrooms:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

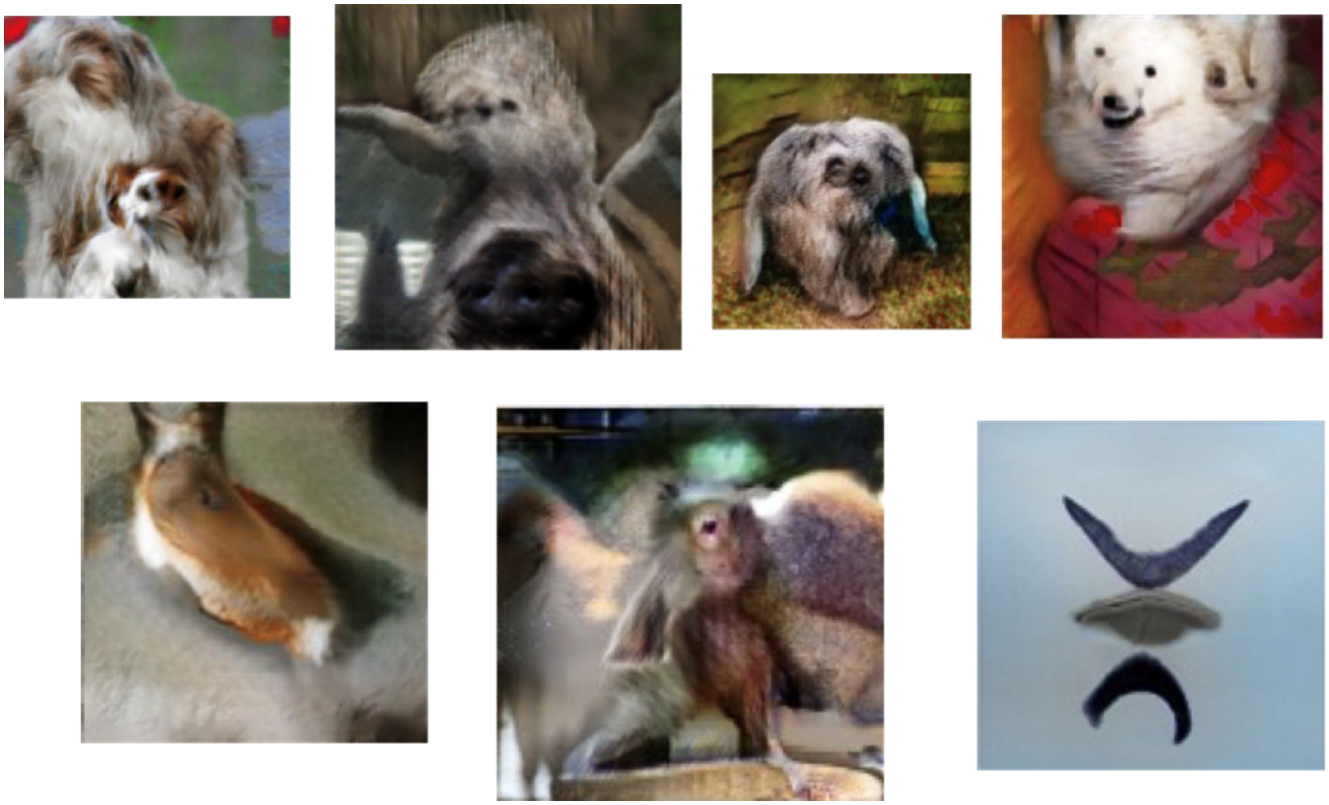
GAN Samples

Objects:

Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

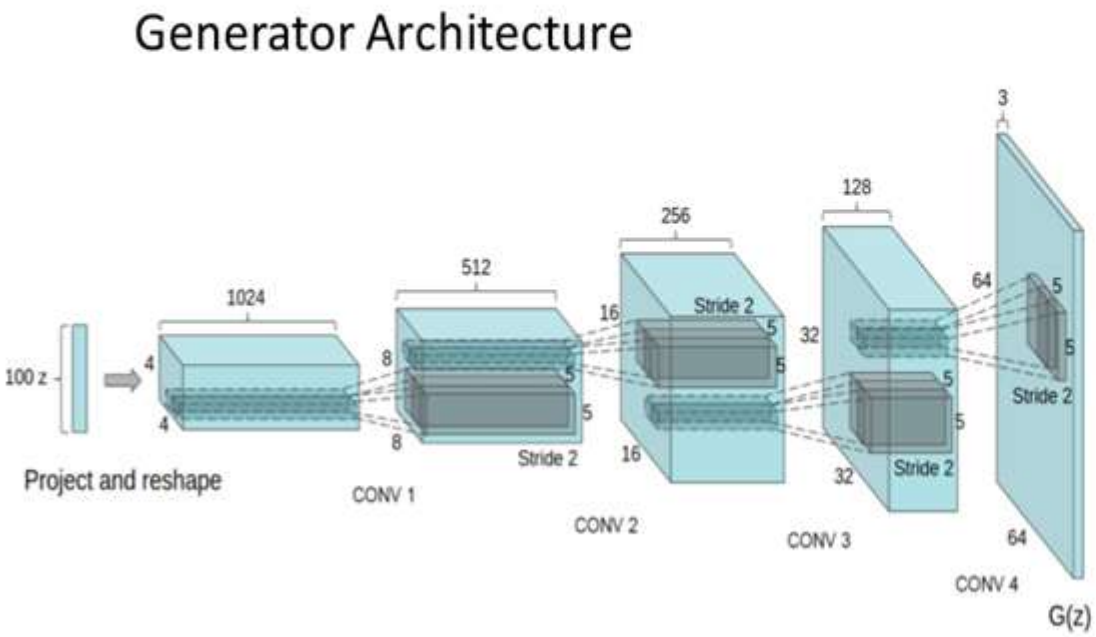
GAN Samples

Cherry Picked:



Goodfellow (2017)

Deep convolutional GANs (DCGAN) (Radford et al., 2015)



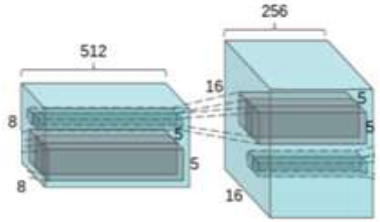
Key ideas:

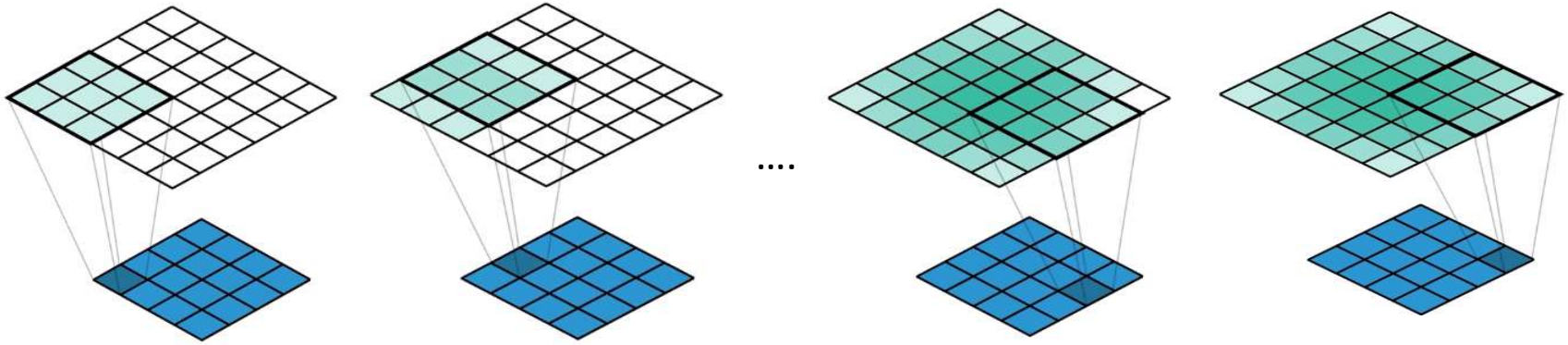
- Replace FC hidden layers with Convolutions
  - **Generator:** Fractional-Strided convolutions
- Use Batch Normalization after each layer
- **Inside Generator**
  - Use ReLU for hidden layers
  - Use Tanh for the output layer

- Batch normalization is important here, apparently.
- **Fractional-Strided convolutions** is one type of *Convolutional Transpose* operation (see next slide).

Deconvolutional GANs (DCGAN)

- Convolutional transpose operation
- It is like upsampling





Conv2D Transpose with $3x3$ kernel (not seen explicitly) applied to a $4x4$ input to give a $6x6$ output.

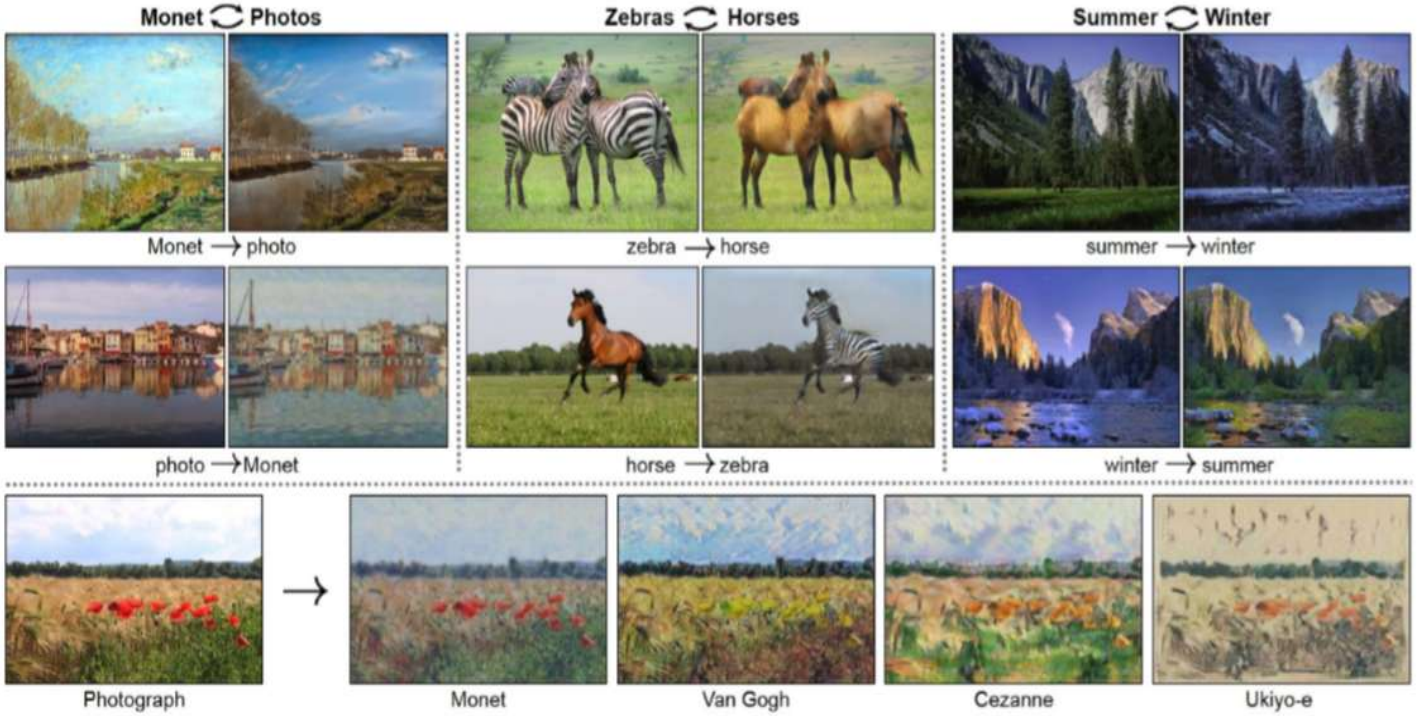
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1																						
2			Input							Kernel						Output						
3																						
4																1	2	3	3	2	1	
5			1	1	1	1				1	1	1				2	4	6	6	4	2	
6			1	1	1	1				1	1	1				3	6	9	9	6	3	
7			1	1	1	1				1	1	1				3	6	9	9	6	3	
8			1	1	1	1										2	4	6	6	4	2	
9																1	2	3	3	2	1	
10																						

Conv2D Transpose with $3x3$ kernel (seen explicitly) applied to a $4x4$ input to give a $6x6$ output.

https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

CycleGAN

- Style transfer problem: change the style of an image while preserving the content



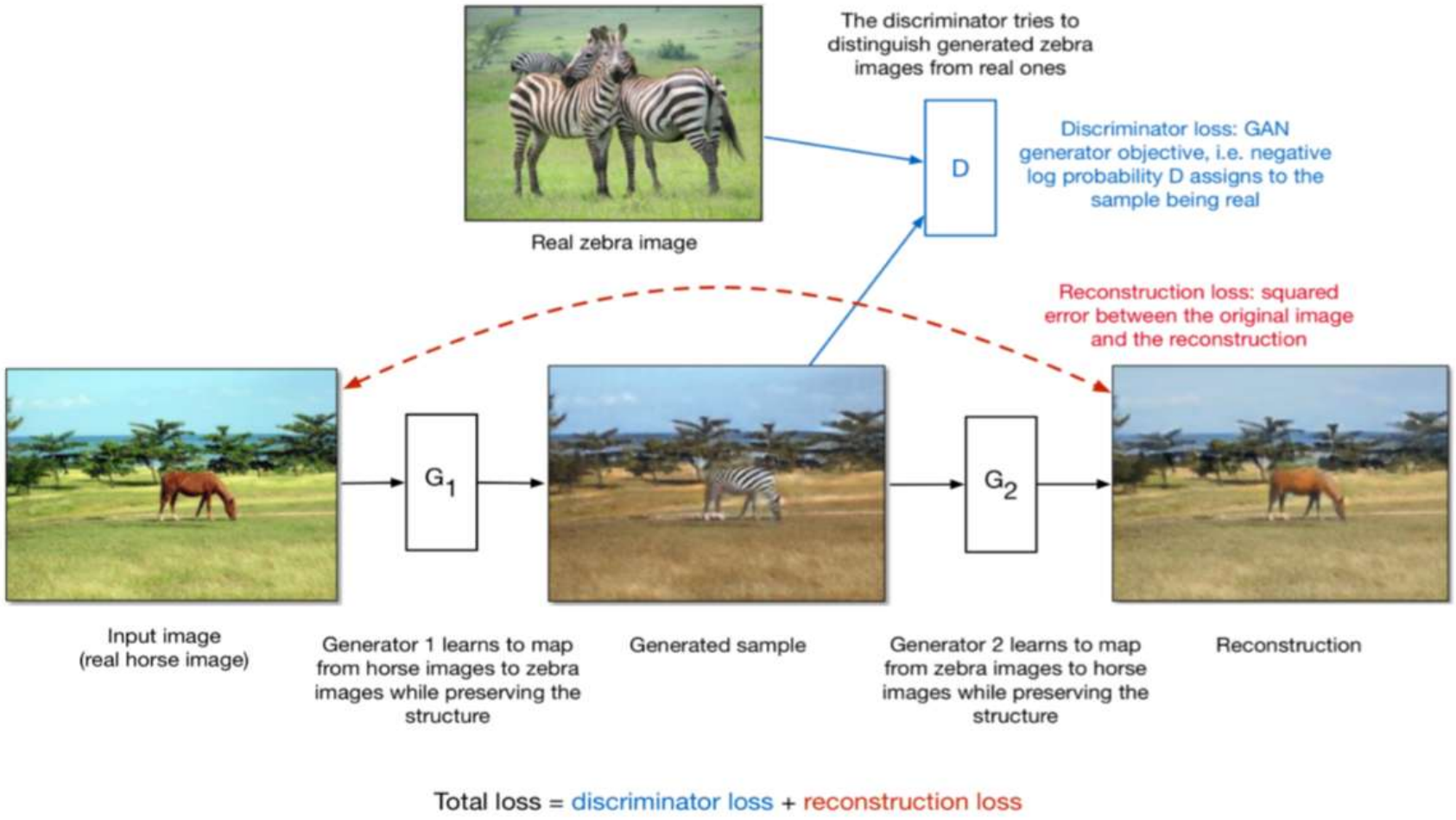
Data: Two unrelated collections of images, one for each style

https://junyanz.github.io/CycleGAN/

CycleGAN

- This is not a supervised learning problem as we do not have paired data (same content in both styles).
  - ➢ This is hard to find.
- The CycleGAN architecture learns to do it from unpaired data.
- Train two different generator nets to go from style 1 to style 2, and vice versa.
  - ➢ Make sure the generated samples of style 2 are indistinguishable from real images by a discriminator net.
  - ➢ Make sure the generators are cycle-consistent: mapping from style 1 to style 2 and back again should give you almost the original image.

CycleGAN



For more details see_ https://junyanz.github.io/CycleGAN/

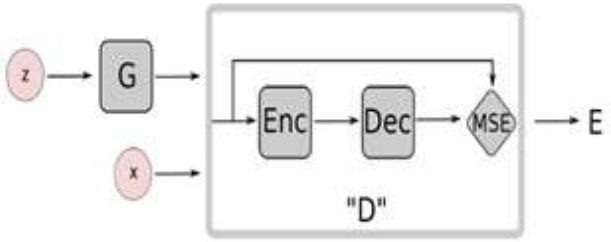
Energy-Based GAN

- Modified game plans
  - **Generator** will try to generate samples with low values
  - **Discriminator** will try to assign high scores to fake values

$$D(x) = ||Dec(Enc(x)) - x||_{MSE}$$

- Use AutoEncoder inside the Discriminator

- Use Mean-Squared Reconstruction error as $D(x)$
  - High Reconstruction Error for Fake samples
  - Low Reconstruction Error for Real samples



Zhao, Junbo, Michael Mathieu, and Yann LeCun. "Energy-based generative adversarial network." arXiv preprint arXiv:1609.03126 (2016)

- Views the discriminator as an energy function that attributes
  - Low energies to the regions near the data manifold
  - Higher energies to other regions
- Stable behavior than regular GANs during training
- Discriminator cost function composes of two goals:
  - A good autoencoder: The reconstruction cost $\boldsymbol{D(x)}$ for real images to be low
  - A good critic: Penalizing the discriminator if the reconstruction error for generated images drops below some value $\boldsymbol{m>0}$.
  - Function $\boldsymbol{D}$: $\boldsymbol{u \rightarrow D(u)}$ is a real valued function not a binary one (the original GAN)

$$\boldsymbol{L_D(x,z) = \|Dec(Enc(x) - x)\|_2^2 + max\left(0, m - D\left(G(z)\right)\right)}$$

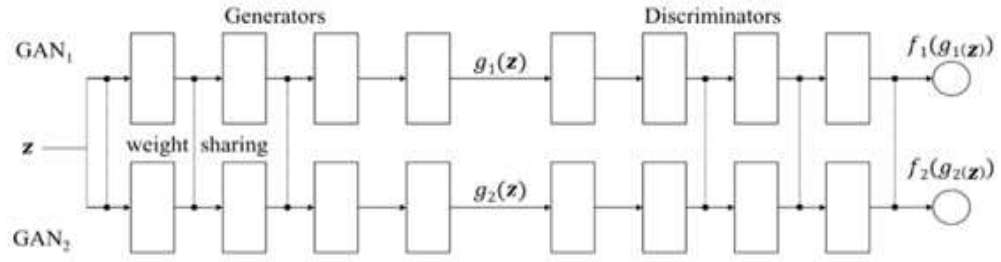
$$\boldsymbol{L_G(z) = D\left(G(z)\right)}$$

https://medium.com/@jonathan_hui/gan-energy-based-gan-ebgan-boundary-equilibrium-gan-began-4662cceb7824

Coupled GAN

- Learning a joint distribution of multi-domain images
- Using GANs to learn the joint distribution with samples drawn from the marginal distributions
- Applications in domain adaptation and image translation

• Architecture



Weight-sharing constraints the network to learn a *joint distribution* without corresponding supervision.

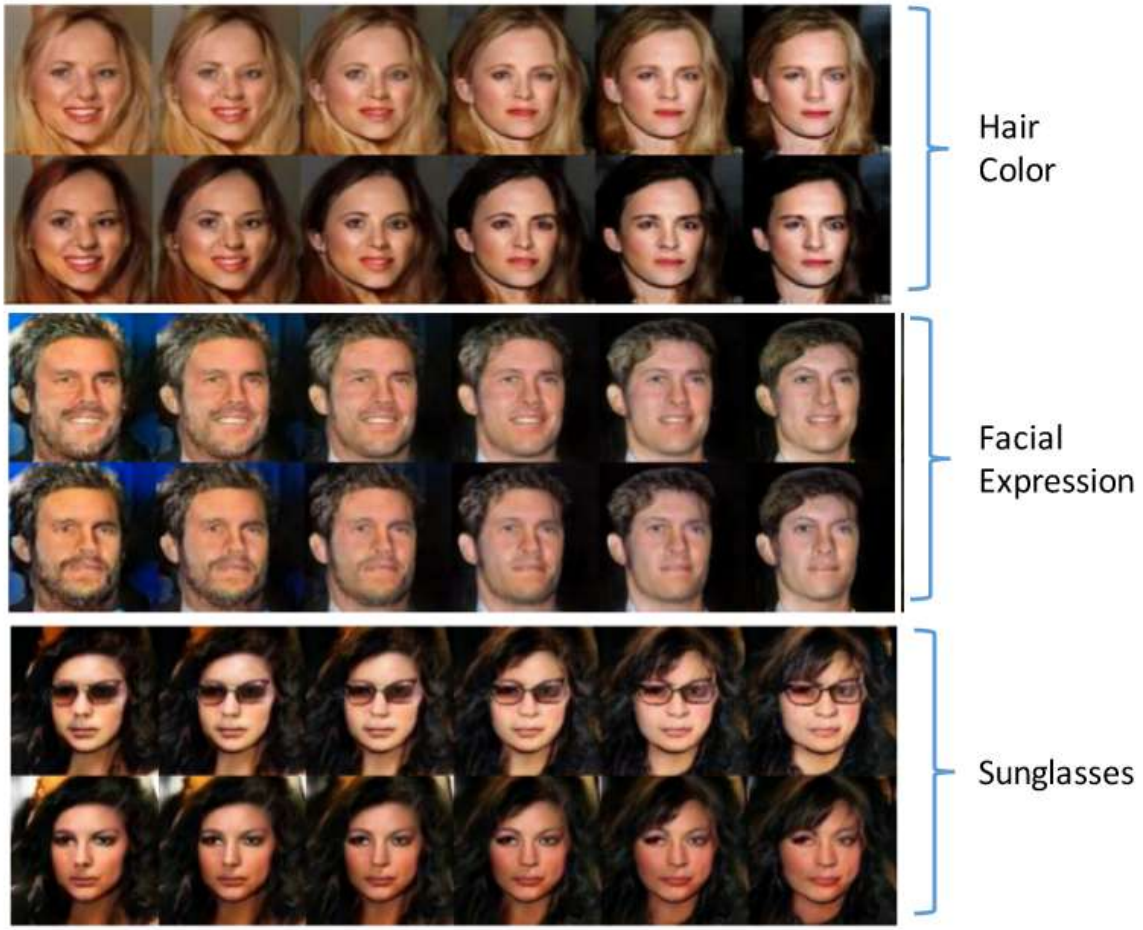
Liu, Ming-Yu, and Oncel Tuzel. "Coupled generative adversarial networks". NIPS (2016).

- Two teams and each team has two players
- The generative models form a team and work together for synthesizing a pair of images in two different domains for confusing the discriminative models
- The discriminative models try to differentiate images drawn from the training data distribution in the respective domains from those drawn from the respective generative models.
- Learning requires training samples drawn from the marginal distributions, P_{x1} and P_{x2} . It does not rely on samples drawn from the joint distribution
- Learn a joint distribution of images in the two domains

Coupled GAN

Coupled GANs

- Some examples of generating facial images across different feature domains.
- Corresponding images in a column are generate from the same latent code z

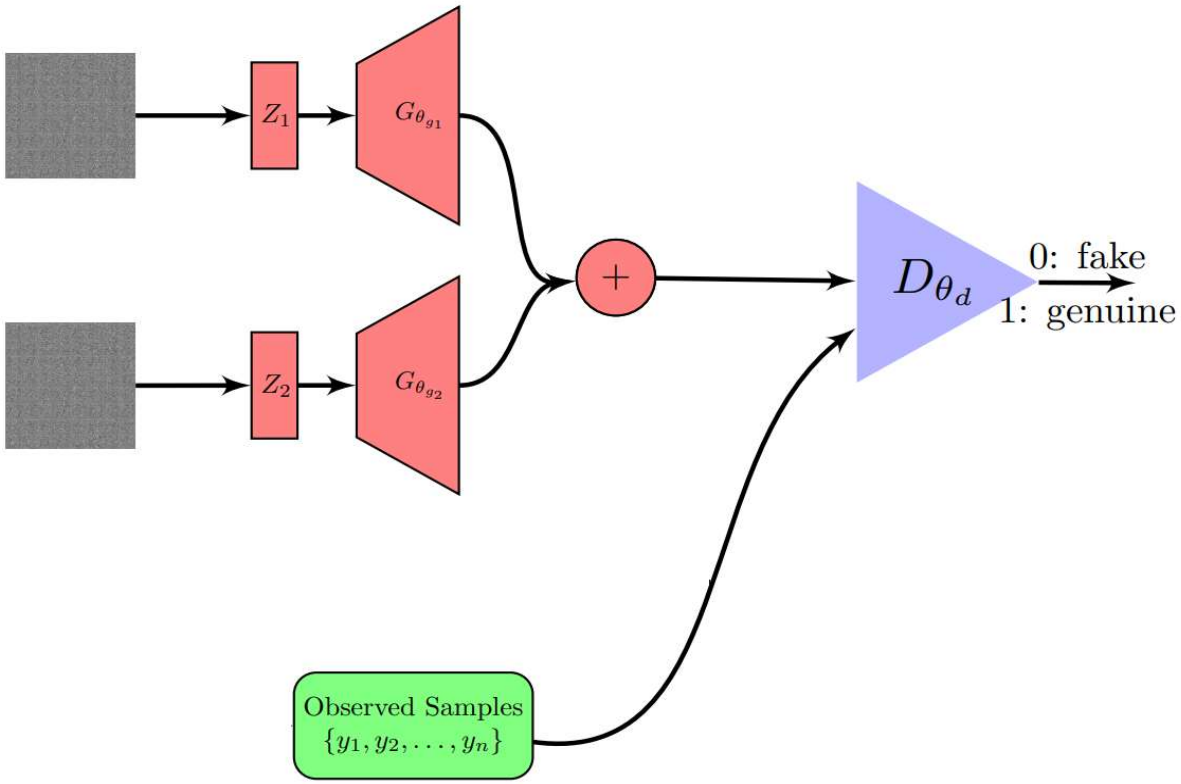


Liu, Ming-Yu, and Oncel Tuzel. "**Coupled generative adversarial networks**". NIPS (2016).

Demixing GAN

- Learning separate sources from mixtures using GAN

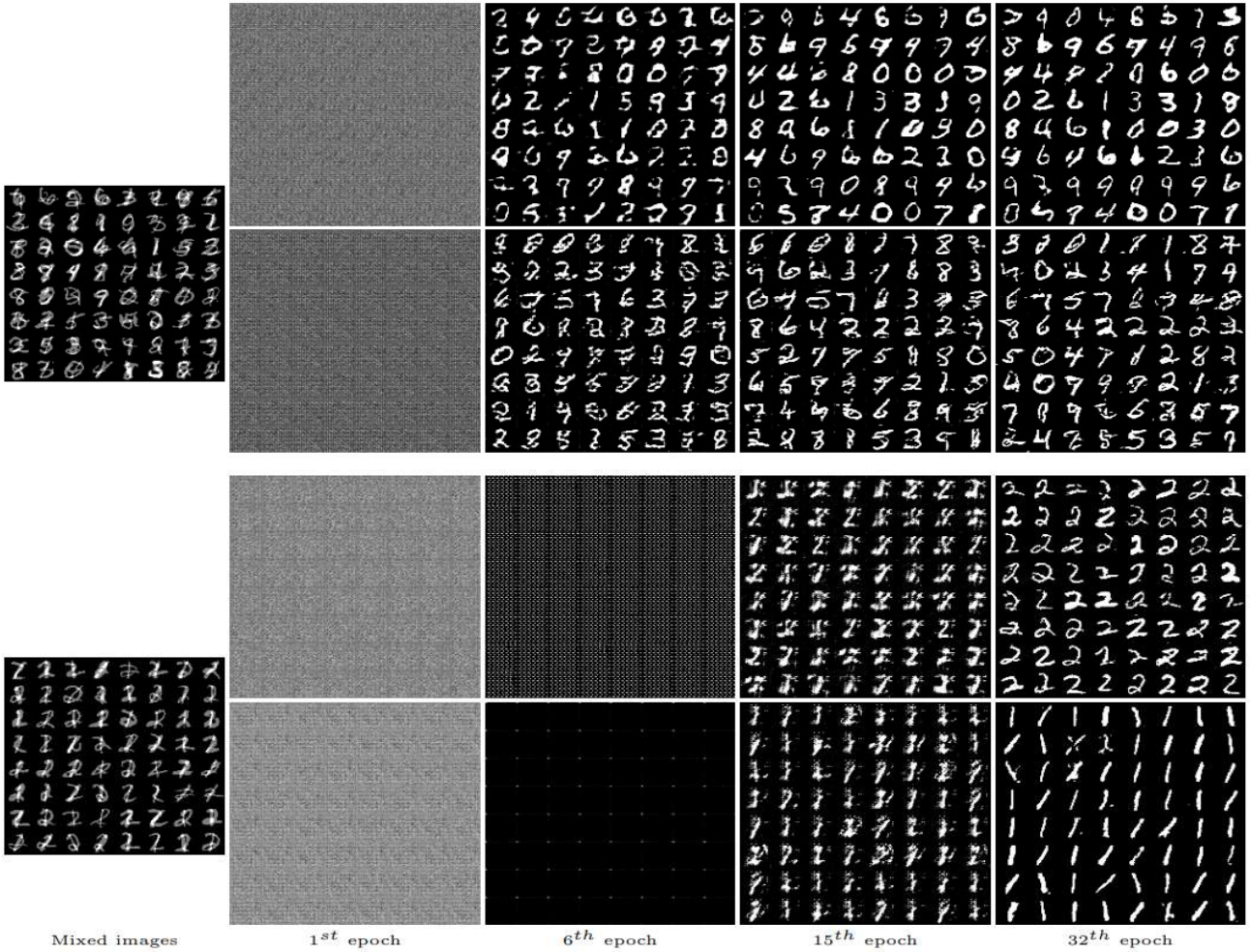
$$Y_i = X_i + N_i = 1,2, \dots, n$$



M. Soltani et al., "Learning Generative Models of Structured Signals from Their Superposition Using GANs", 2018.

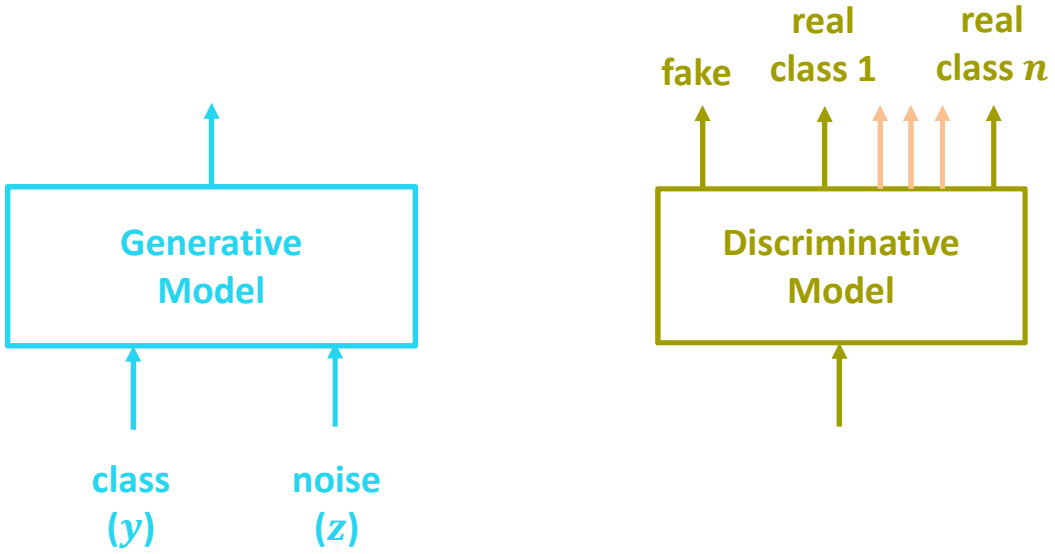
Demixing GAN

- Superposing two MNIST digits
- Separating them using demixing GAN



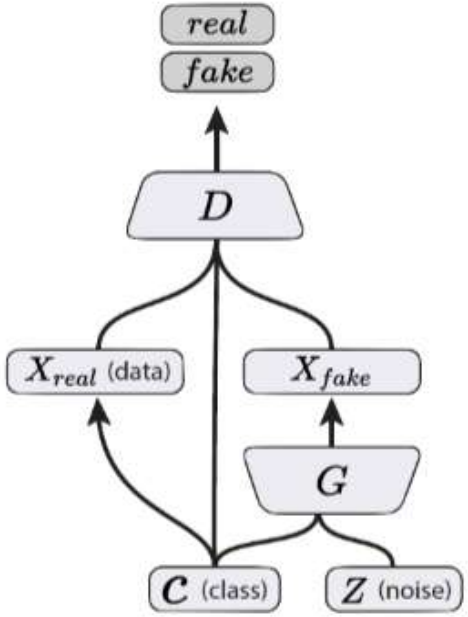
Using Labels Can Improve Generated Samples

Denton et al. (2015)
Salimans et al. (2016)



Conditional GANs

- Providing side information for generator
- Simple modification to the original GAN framework that conditions the model on additional information for better multi-modal learning
- It gives you more control on what you can generate
- Lends to many practical applications of GANs when we have explicit supervision available



Conditional GANs

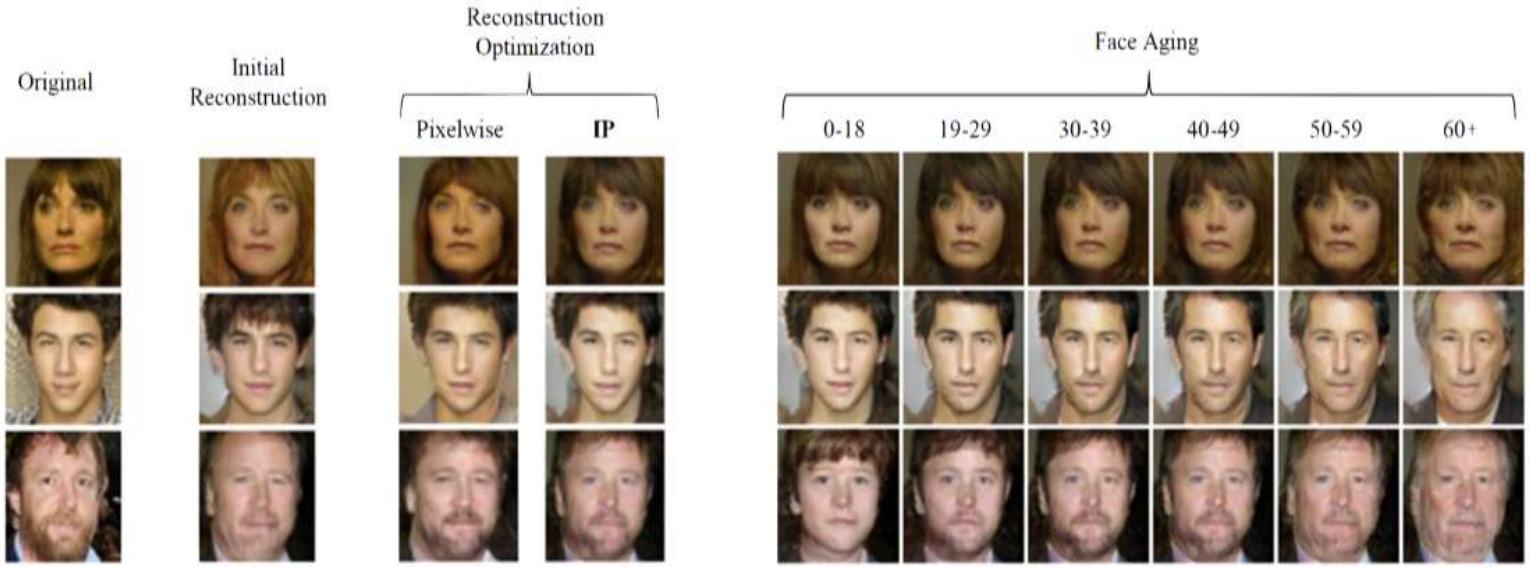
MNIST digits generated conditioned on their class label.

$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 1, 0, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 1, 0, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 1, 0, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 1, 0, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 1, 0, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 0, 1, 0, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] \rightarrow$
$[0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \rightarrow$

Mirza, Mehdi, and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014).

Applications of Conditional GANs

- Face Aging



Antipov, G., Baccouche, M., & Dugelay, J. L. (2017). "**Face Aging With Conditional Generative Adversarial Networks**". arXiv preprint arXiv:1702.01983.

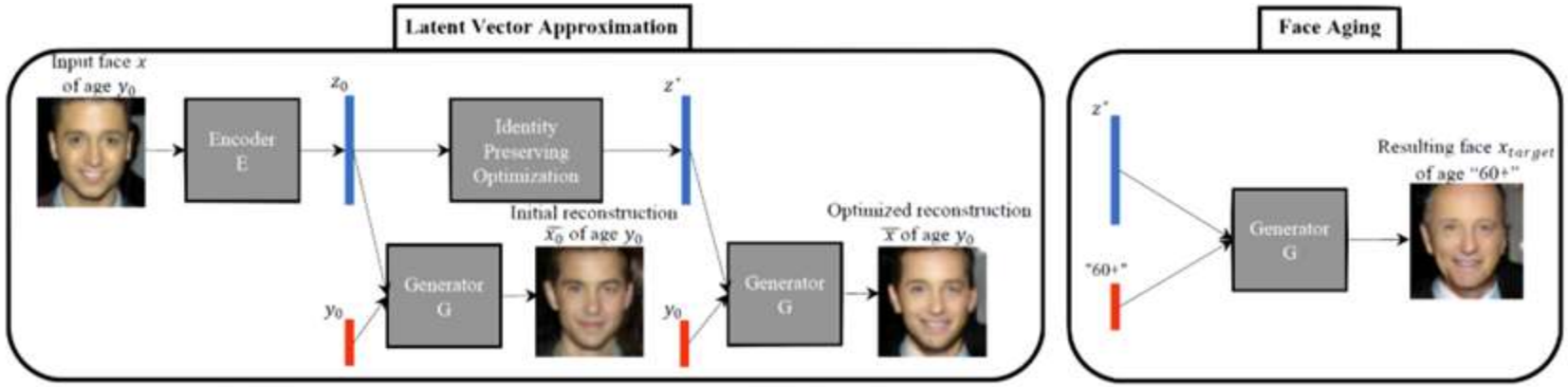
IP: Identity Preserving (please see the above paper for more details)

Applications of Conditional GANs

- Face Aging
  - Using an auxiliary network to get a better approximation of the latent code (z^*) for an input image.
    - Given an input face image x of age y_0, find an optimal latent vector z^* which allows to generate a reconstructed face $\bar{x} = G(z^*, y_0)$ as close as possible to the initial one
    - Identity Preserving Optimization (FR is neural network called Face Recognition (recognizes a person's identity in a given image))

$$z^*_{IP} = \underset{z}{\operatorname{argmin}} ||FR(x) - FR(\bar{x})||_{L_2}$$

  - Latent code is then conditioned on a discrete (one-hot) embedding of age categories



Antipov, G., Baccouche, M., & Dugelay, J. L. (2017). "**Face Aging With Conditional Generative Adversarial Networks**". arXiv preprint arXiv:1702.01983.

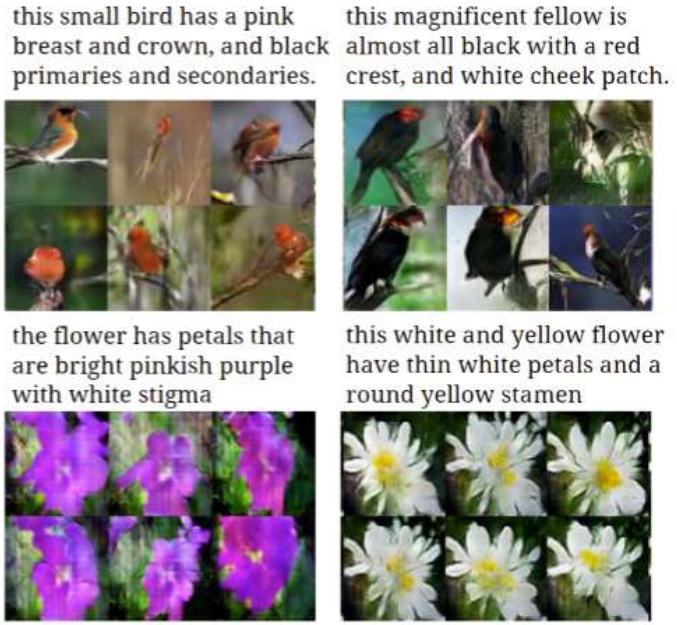
Applications of Conditional GANs

- Text-to-Image Synthesis

Motivation

Given a text description, generate images closely associated.

Uses a conditional GAN with the generator and discriminator being condition on “dense” text embedding.

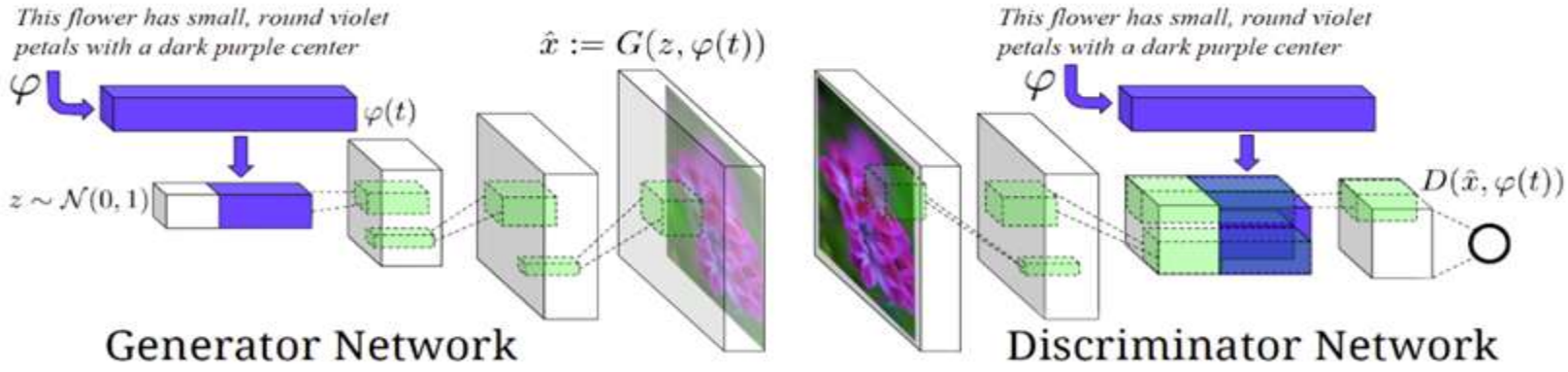


Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. **“Generative adversarial text to image synthesis”**. ICML (2016).

Applications of Conditional GANs

- Text-to-Image Synthesis

Text-conditional convolutional GAN architecture



Positive Example:
Real Image, Right Text

Negative Examples:
Real Image, Wrong Text
Fake Image, Right Text

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. "Generative adversarial text to image synthesis". ICML (2016).

- Text encoding $\phi(t)$ is used by both generator and discriminator.
- It is projected to a lower-dimensions and depth concatenated with image feature maps for further stages of convolutional processing.

InfoGAN

- InfoGAN tries to provide more control over the generated samples (similar to the conditional GAN) by creating a ***disentangled representation***.
- Splitting the Generator input into two parts:
  - The traditional noise vector
  - A new "latent code" vector
- InfoGAN creates a meaningful code by maximizing the Mutual Information (MI) between the code and the generator output:

$$min_{\theta_g}\, max_{\theta_d} V\left(D_{\theta_d}, G_{\theta_g}\right) - \lambda I(c, G_{\theta_g}(z, c))$$

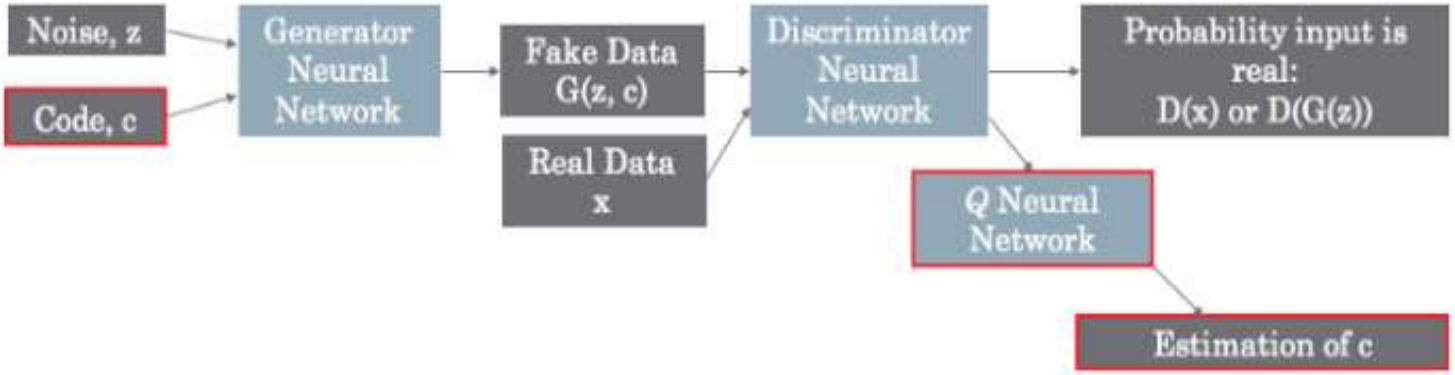
- $V\left(D_{\theta_d}, G_{\theta_g}\right)$ can be any min-max loss in (Jensen-Shannon, Wasserstein,…) .
  - Lambda is is typically just set to one.
- A variational (lower bound) approximation is used for the MI to make the computation tractable (requires access to the posterior $P(c|x)$):

$$\begin{aligned} L_I(G,Q) &= E_{c\sim P(c),x\sim G(z,c)}[\log Q(c|x)] + H(c) \\ &= E_{x\sim G(z,c)}[\mathbb{E}_{c'\sim P(c|x)}[\log Q(c'|x)]] + H(c) \\ &\leq I(c; G(z,c)) \end{aligned}$$

  - $Q(c|x)$ is an auxiliary distribution to approximate $P(c|x)$:

https://arxiv.org/pdf/1606.03657.pdf

InfoGAN Architecture

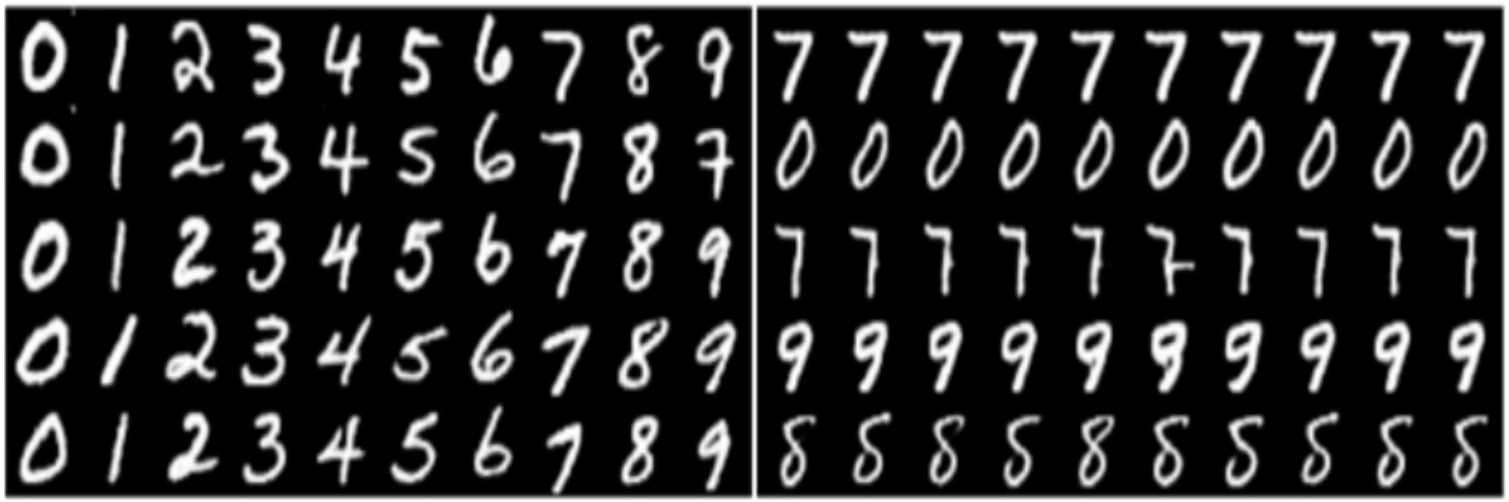


- Parametrizing the auxiliary distribution Q as a neural network
- Q and D share all convolutional layers and there is one final fully connected layer to output parameters for the conditional distribution $Q(c|x)$
- Adding a negligible computation cost to GAN

https://towardsdatascience.com/infogan-generative-adversarial-networks-part-iii-380c0c6712cd

InfoGAN Generated Samples

- The images below show a process where a particular noise vector is held constant (each row), but the latent code is changed (each column).
- In InfoGAN, the discrete code consistently change the digit. The regular GAN has essentially no meaningful or consistent change.



InfoGAN Regular GAN

https://towardsdatascience.com/infogan-generative-adversarial-networks-part-iii-380c0c6712cd

BigGAN

- GANs typically generate small images.
- The training process is instable (needs careful hyperparameters fine-tuning).
- **The BigGAN combines a suite of recent best practices in training class-conditional images:**
  - More model parameters
    - 2 to 4 times as many parameters compared to prior art
  - Larger Batch Sizes
    - 8 times the batch size compared to prior art
  - Architectural changes
    - Improving scalability with general architectural changes, and modifying a regularization scheme

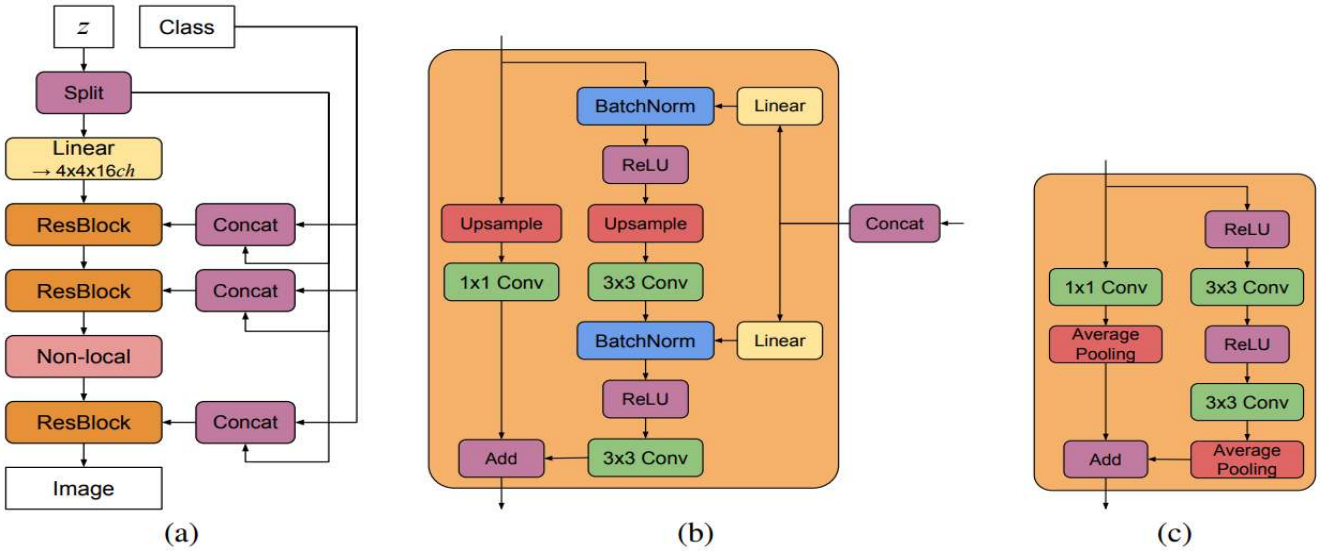
https://arxiv.org/pdf/1809.11096.pdf

BigGAN Samples

Samples generated by BigGAN model at 256×256 resolution.

Samples generated by BigGAN model at 512×512 resolution.

BigGAN—Architecture (128×128)



(a) A typical architectural layout for BigGAN's **G**; details are in the following tables.
(b) A Residual Block (*ResBlock up*) in BigGAN's **G**.
(c) A Residual Block (*ResBlock down*) in BigGAN's **D**.

$z \in \mathbb{R}^{120} \sim \mathcal{N}(0, I)$ Embed$(y) \in \mathbb{R}^{128}$
Linear $(20 + 128) \rightarrow 4 \times 4 \times 16ch$
ResBlock up $16ch \rightarrow 16ch$
ResBlock up $16ch \rightarrow 8ch$
ResBlock up $8ch \rightarrow 4ch$
ResBlock up $4ch \rightarrow 2ch$
Non-Local Block (64×64)
ResBlock up $2ch \rightarrow ch$
BN, ReLU, 3×3 Conv $ch \rightarrow 3$
Tanh

(a) Generator

RGB image $x \in \mathbb{R}^{128 \times 128 \times 3}$
ResBlock down $ch \rightarrow 2ch$
Non-Local Block (64×64)
ResBlock down $2ch \rightarrow 4ch$
ResBlock down $4ch \rightarrow 8ch$
ResBlock down $8ch \rightarrow 16ch$
ResBlock down $16ch \rightarrow 16ch$
ResBlock $16ch \rightarrow 16ch$
ReLU, Global sum pooling
Embed$(y) \cdot \boldsymbol{h}$ + (linear $\rightarrow 1$)

(b) Discriminator

- BigGAN architecture for 128 × 128 images.
- ch represents the channel width multiplier (e.g., ch=64).
- "split" splits the vector z along its channel dimension into chunks of equal size (20-D).
- Each chunk is concatenated to the shared class embedding and passed to a corresponding residual block as a conditioning vector.

https://arxiv.org/pdf/1809.11096.pdf

BigGAN—Innovations

1. Using Hing-Loss (similar to SVM loss):

$$V(D,G) = -\mathbb{E}_{\boldsymbol{x}\sim \text{real}} \max(0, 1 - D_{\theta_d}(\boldsymbol{x})) - \mathbb{E}_{\boldsymbol{z}} \max(0, 1 - D_{\theta_d}(G_{\theta_g}(z)))$$

2. Class conditional information

- Class-conditional batch normalization (introduced by Dumoulin, et al. 2016)
  - Normalizing activations based on the statistics from images of a given class
  - Transforming a layer's activations into a normalized activation specific to the class of images
- Providing *class-conditional batch normalization* to the generator network
  - Instead of using one class embedding per label, a shared class-conditional code is used in order to reduce the number of weights.

3. Spectral normalization for the weights of Generator function G!

https://machinelearningmastery.com/a-gentle-introduction-to-the-biggan/

BigGAN—Innovations

4. **Update the Discriminator more than the Generator**
   - Updating the Discriminator twice before updating the Generator in each training iteration
5. **Orthogonal Weight Initialization**
   - Initialized weights by random orthogonal matrix.
6. **Larger Batch Size**
   - e.g., batch sizes of 256, 512, 1024, and 2,048 images
7. **Larger number of parameters**
   - Doubling the number of channels or feature maps (filters) in each layer
8. **Skip-z Connections**
   - Directly connecting the input latent vector z to specific layers deep in Generator

BigGAN—Innovations

9. Truncation Trick

- during training samples the generator's latent code z from standard distribution
- During inference samples the generator's latent code z using a truncated standard distribution
  - Resampling if the values of z is above a threshold until it falls below the threshold
- The smaller the threshold, the better quality
- The larger the threshold, the more variety in sampled images

10. Orthogonal Weight Regularization

- Some of the deeper models creates saturated artifacts when using the truncation trick.
- Instead, using *Orthogonal Regularization* (Brock et al., 2017)

$$R_\beta(W) = \left\|W^T W - \boldsymbol{I}\right\|_F^2$$

- Using more complicated *Orthogonal Regularization:*

$$R_\beta(W) = \left\|W^T W \odot (\mathbf{1} - \boldsymbol{I})\right\|_F^2$$

  - W denotes the weight matrices.
  - $\mathbf{1}$ denotes a matrix with all ones.
  - $\odot$ denotes an element-wise multiplication.

The GAN ZOO

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

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