Autoencoders and GANs

Deep Learning Lecture 13

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2017
(Slides credit to Goodfellow, Larochelle, Hinton)

Table of Contents

Review

- We talked about restricted Boltzmann machines (RBMs) and deep belief networks (DBNs) last time
  - DBNs were the first studied deep networks
  - RBMs have been served a useful tool for network pre-training
- We will look into two important neural network models: autoencoders and generative adversarial networks (GANs)

Why autoencoders? Dimension reduction

- As name suggests, the objective of dimension of reduction is to decrease the dimension of input signals to ease later processing
  - It is often a preprocessing step
  - Was commonly used to compress features
- It is a very old problem. The most representative algorithm is the principal component analysis (PCA)

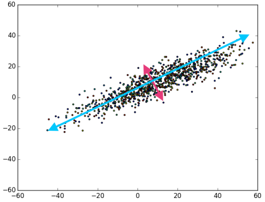
Why autoencoders? Dimension reduction

- As name suggests, the objective of dimension of reduction is to decrease the dimension of input signals to ease later processing
  - It is often a preprocessing step
  - Was commonly used to compress features
- It is a very old problem. The most representative algorithm is the principal component analysis (PCA)

Why autoencoders? Dimension reduction

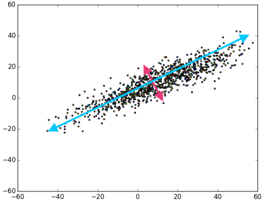
- As name suggests, the objective of dimension of reduction is to decrease the dimension of input signals to ease later processing
    - It is often a preprocessing step
    - Was commonly used to compress features
- It is a very old problem. The most representative algorithm is the principal component analysis (PCA)

Principal component analysis (PCA)



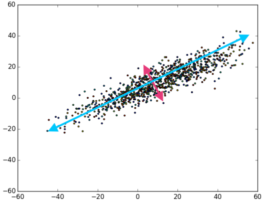
- Take N-dimensional data and find the M orthogonal directions in which the data have the most variance
  - We can represent an N-dimensional datapoint by its projections onto the M principal directions (i.e., with highest variances)
  - This loses all information about where the datapoint is located in the remaining orthogonal directions

Principal component analysis (PCA)



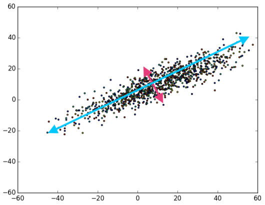
- Take N-dimensional data and find the M orthogonal directions in which the data have the most variance
  - We can represent an N-dimensional datapoint by its projections onto the M principal directions (i.e., with highest variances)
  - This loses all information about where the datapoint is located in the remaining orthogonal directions

Principal component analysis (PCA)



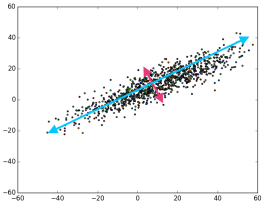
- Take N-dimensional data and find the M orthogonal directions in which the data have the most variance
  - We can represent an N-dimensional datapoint by its projections onto the M principal directions (i.e., with highest variances)
  - This loses all information about where the datapoint is located in the remaining orthogonal directions

PCA reconstruction



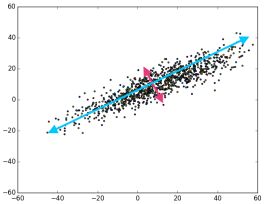
- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is “optimum”
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

PCA reconstruction



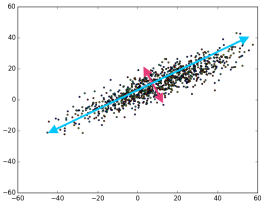
- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

PCA reconstruction



- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

PCA reconstruction



- We reconstruct by using the mean value (over all the data) on the $N - M$ directions that are not represented.
  - The reconstruction error is the sum over the variances over all these unrepresented directions
    - The variances are just eigenvalues of covariance matrix of the data
- PCA is "optimum"
  - Since we keep the largest variance components, on average the distortion is minimum among all linear dimension reduction methods

Math review: Singular value decomposition (SVD)

For any $N \times K$ matrix A (assume $K \leq N$), we can decompose it into product of three matrices

$$\begin{pmatrix} \\ \\ A \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ U \\ \\ \\ \end{pmatrix} \begin{pmatrix} \\ D \\ \\ \end{pmatrix} \begin{pmatrix} \\ V \\ \\ \end{pmatrix}^T,$$

where U is $N \times K$, D is $K \times K$, and V is $K \times K$. Moreover,

- U is orthonormal, i.e., $U^T U = I$
- D is diagonal
- V is orthonormal, i.e., $V^T V = I$

Has nice geometric interpretation. Roughly speaking, any linear transform can be decompose into rotation, scaling, and rotation again

Math review: Singular value decomposition (SVD)

For any $N \times K$ matrix A (assume $K \leq N$), we can decompose it into product of three matrices

$$\begin{pmatrix} \\ \\ A \\ \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \\ U \\ \\ \\ \end{pmatrix} \begin{pmatrix} \\ D \\ \\ \end{pmatrix} \begin{pmatrix} \\ V \\ \\ \end{pmatrix}^T,$$

where U is $N \times K$, D is $K \times K$, and V is $K \times K$. Moreover,

- U is orthonormal, i.e., $U^T U = I$
- D is diagonal
- V is orthonormal, i.e., $V^T V = I$

Has nice geometric interpretation. Roughly speaking, any linear transform can be decompose into rotation, scaling, and rotation again

Math review: Singular value decomposition (SVD)

For any $N \times K$ matrix A (assume $K \leq N$), we can decompose it into product of three matrices

$$\left(\quad A \quad \right) = \left(\quad U \quad \right) \left(\quad D \quad \right) \left(\quad V \quad \right)^T,$$

where U is $N \times K$, D is $K \times K$, and V is $K \times K$. Moreover,

- U is orthonormal, i.e., $U^T U = I$
- D is diagonal
- V is orthonormal, i.e., $V^T V = I$

Has nice geometric interpretation. Roughly speaking, any linear transform can be decompose into rotation, scaling, and rotation again

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error &= \sum_i (x-\hat{x})^T(x-\hat{x}) = tr((X-\hat{X})^T(X-\hat{X})) \\
&= tr(V(\Sigma-\hat{\Sigma})U^T U(\Sigma-\hat{\Sigma})V^T) = tr(V(\Sigma-\hat{\Sigma})(\Sigma-\hat{\Sigma})V^T) \\
&= tr(((\Sigma-\hat{\Sigma})V^T)^T(\Sigma-\hat{\Sigma})V^T) = tr((\Sigma-\hat{\Sigma})^2) \\
&= \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = \sum_i (x - \hat{x})^T (x - \hat{x}) &= tr((X - \hat{X})^T (X - \hat{X})) \\
&= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
&= tr(((\Sigma - \hat{\Sigma})V^T)^T (\Sigma - \hat{\Sigma})V^T) = tr((\Sigma - \hat{\Sigma})^2) \\
&= \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = & \sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X})) \\
= & tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
= & tr(((\Sigma - \hat{\Sigma})V^T)^T (\Sigma - \hat{\Sigma})V^T) = tr((\Sigma - \hat{\Sigma})^2) \\
= & \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = &\sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X})) \\
=&tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
=&tr(((\Sigma - \hat{\Sigma})V^T)^T (\Sigma - \hat{\Sigma})V^T) = tr((\Sigma - \hat{\Sigma})^2) \\
=&\text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error = \sum_i (x-\hat{x})^T(x-\hat{x}) &= tr((X-\hat{X})^T(X-\hat{X})) \\
&= tr(V(\Sigma-\hat{\Sigma})U^T U(\Sigma-\hat{\Sigma})V^T) = tr(V(\Sigma-\hat{\Sigma})(\Sigma-\hat{\Sigma})V^T) \\
&= tr(((\Sigma-\hat{\Sigma})V^T)^T(\Sigma-\hat{\Sigma})V^T) = tr((\Sigma-\hat{\Sigma})^2) \\
&= \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T (U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$Error = \sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X}))$$

$$= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T)$$

$$= tr(((\Sigma - \hat{\Sigma})V^T)^T (\Sigma - \hat{\Sigma})V^T) = tr((\Sigma - \hat{\Sigma})^2)$$

$=$ Sum of eigenvalues excluding the M largest ones

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$Error = \sum_i (x - \hat{x})^T (x - \hat{x}) = tr((X - \hat{X})^T (X - \hat{X}))$$

$$= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T)$$

$$= tr(((\Sigma - \hat{\Sigma})V^T)^T (\Sigma - \hat{\Sigma})V^T) = tr((\Sigma - \hat{\Sigma})^2)$$

$=$Sum of eigenvalues excluding the M largest ones

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$\begin{aligned}
Error = & \sum_i (x-\hat{x})^T(x-\hat{x}) = tr((X-\hat{X})^T(X-\hat{X})) \\
= & tr(V(\Sigma-\hat{\Sigma})U^T U(\Sigma-\hat{\Sigma})V^T) = tr(V(\Sigma-\hat{\Sigma})(\Sigma-\hat{\Sigma})V^T) \\
= & tr(((\Sigma-\hat{\Sigma})V^T)^T(\Sigma-\hat{\Sigma})V^T) = tr((\Sigma-\hat{\Sigma})^2) \\
= & \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2U^T$, thus singular values are just square root of eigenvalues
    - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error &= \sum_i (x - \hat{x})^T(x - \hat{x}) = tr((X - \hat{X})^T(X - \hat{X})) \\
&= tr(V(\Sigma - \hat{\Sigma})U^T U(\Sigma - \hat{\Sigma})V^T) = tr(V(\Sigma - \hat{\Sigma})(\Sigma - \hat{\Sigma})V^T) \\
&= tr(((\Sigma - \hat{\Sigma})V^T)^T(\Sigma - \hat{\Sigma})V^T) = tr((\Sigma - \hat{\Sigma})^2) \\
&= \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

SVD and PCA

- Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_K]$ be the matrix with columns as data vectors. We can decompose $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$ using SVD
- Assume X is zero-mean, the covariance matrix C is just $C \approx \frac{XX^T}{k}$
- Note that $C \sim U\Sigma V^T(U\Sigma V^T)^T = U\Sigma^2 U^T$, thus singular values are just square root of eigenvalues
  - Since PCA is in effect keeping the M largest eigenvalues of the covariance matrix, it is the same as keeping the M largest singular values of X
- One can easily verify that. Let $\hat{X} = U\hat{\Sigma}V^T$, where $\hat{\Sigma}$ only keeps the M largest singular values, then

$$
\begin{aligned}
Error =& \sum_i (x-\hat{x})^T(x-\hat{x}) = tr((X-\hat{X})^T(X-\hat{X})) \\
=& tr(V(\Sigma-\hat{\Sigma})U^T U(\Sigma-\hat{\Sigma})V^T) = tr(V(\Sigma-\hat{\Sigma})(\Sigma-\hat{\Sigma})V^T) \\
=& tr(((\Sigma-\hat{\Sigma})V^T)^T(\Sigma-\hat{\Sigma})V^T) = tr((\Sigma-\hat{\Sigma})^2) \\
=& \text{Sum of eigenvalues excluding the } M \text{ largest ones}
\end{aligned}
$$

Optimal linear decoder ⇒ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

$$\begin{aligned}
h(\mathbf{X}) &= \Sigma_M\mathbf{V}^T = \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X})\\
&= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X})\\
&= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X}\\
&= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}\\
&= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X}
\end{aligned}$$

Optimal linear decoder ⇒ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

$$
\begin{aligned}
h(\mathbf{X}) &= \Sigma_M\mathbf{V}^T = \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}) \\
&= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}) \\
&= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X} \\
&= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X} \\
&= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X}
\end{aligned}
$$

Optimal linear decoder ⇒ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

$$
\begin{aligned}
h(\mathbf{X}) &= \Sigma_M\mathbf{V}^T = \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}) \\
&= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}) \\
&= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X} \\
&= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X} \\
&= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X}
\end{aligned}
$$

Optimal linear decoder ⇒ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

$$
\begin{aligned}
h(\mathbf{X}) &= \Sigma_M\mathbf{V}^T = \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}) \\
&= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}) \\
&= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X} \\
&= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X} \\
&= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X}
\end{aligned}
$$

Optimal linear decoder ⇒ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

$$\begin{aligned} h(\mathbf{X}) = \Sigma_M\mathbf{V}^T &= \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}) \\ &= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}) \\ &= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X} \\ &= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X} \\ &= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X} \end{aligned}$$

Optimal linear decoder $\Rightarrow$ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

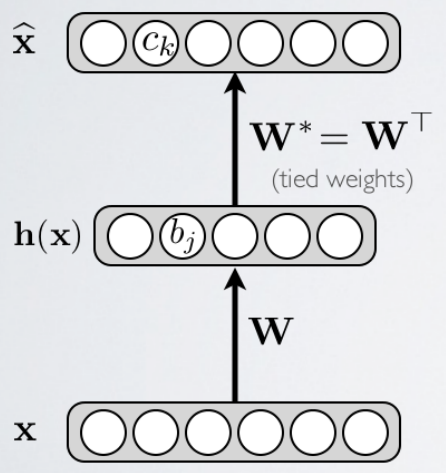
$$
\begin{aligned}
h(\mathbf{X}) &= \Sigma_M\mathbf{V}^T = \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}) \\
&= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}) \\
&= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X} \\
&= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X} \\
&= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X}
\end{aligned}
$$

Optimal linear decoder $\Rightarrow$ optimal linear encoder

- PCA is optimum when things are "linear"
- Interesting to know that as far as decoding is linear, the optimal encoding is linear (PCA) as well
  - That is, if $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X})$ for some optimal $\mathbf{W}$
  - $h(\mathbf{X}) = \mathbf{T}\mathbf{X}$ for some optimal $\mathbf{T}$
- If decoding is restricted to be linear, then ultimately the optimal $\hat{\mathbf{X}} = \mathbf{W}h(\mathbf{X}) = \mathbf{U}\Sigma_M\mathbf{V}^T$
- Let's assume $\mathbf{W} = \mathbf{U}$, then

$$
\begin{aligned}
h(\mathbf{X}) &= \Sigma_M\mathbf{V}^T = \Sigma_M\mathbf{V}^T(\mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X}) \\
&= \Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{V}^T)^{-1}(\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X}) \\
&= \underbrace{\Sigma_M\mathbf{V}^T(\mathbf{V}\Sigma^T\Sigma\mathbf{V}^T)^{-1}\mathbf{V}\Sigma^T\mathbf{U}^T}_{\text{some linear transform}}\mathbf{X} \\
&= \Sigma_M\mathbf{V}^T\mathbf{V}(\Sigma^T\Sigma)^{-1}\mathbf{V}^T\mathbf{V}\Sigma^T\mathbf{U}^T\mathbf{X} \\
&= \Sigma_M(\Sigma^T\Sigma)^{-1}\Sigma^T\mathbf{U}^T\mathbf{X} = \Sigma_M\Sigma^{-1}\mathbf{U}^T\mathbf{X}
\end{aligned}
$$

Autoencoders



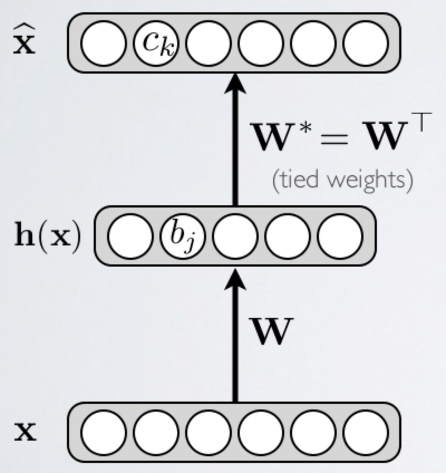
- Autoencoder is a way to perform dimension reduction with neural networks

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

$$\begin{cases} \hat{\mathbf{x}} = \underbrace{\text{sigm}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))}_{\text{binary inputs}} \\ \hat{\mathbf{x}} = \underbrace{\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x})}_{\text{continuous inputs}} \end{cases}$$

- loss = $\|\mathbf{x} - \hat{\mathbf{x}}\|$
- N.B., for continuous inputs, the decoder is linear and so the optimum autoencoder is just equivalent to PCA

Autoencoders



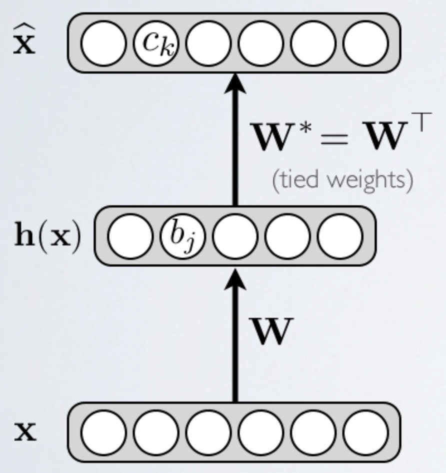
- Autoencoder is a way to perform dimension reduction with neural networks

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

$$\begin{cases} \hat{\mathbf{x}} = \underbrace{\text{sigm}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))}_{\text{binary inputs}} \\ \hat{\mathbf{x}} = \underbrace{\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x})}_{\text{continuous inputs}} \end{cases}$$

- loss = $\|\mathbf{x} - \hat{\mathbf{x}}\|$
- N.B., for continuous inputs, the decoder is linear and so the optimum autoencoder is just equivalent to PCA

Autoencoders



- Autoencoder is a way to perform dimension reduction with neural networks

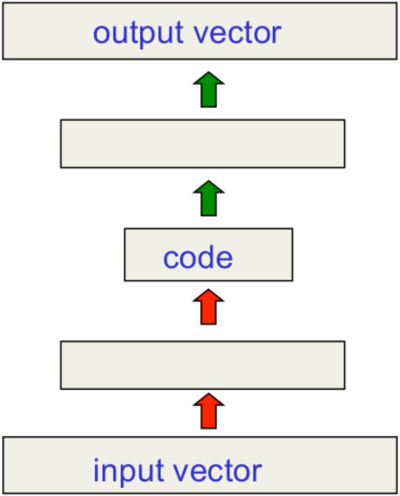
$$\mathbf{h}(\mathbf{x}) = \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$$

$$\begin{cases} \hat{\mathbf{x}} = \underbrace{\text{sigm}(\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x}))}_{\text{binary inputs}} \\ \hat{\mathbf{x}} = \underbrace{\mathbf{c} + \mathbf{W}^*\mathbf{h}(\mathbf{x})}_{\text{continuous inputs}} \end{cases}$$

- loss = $\|\mathbf{x} - \hat{\mathbf{x}}\|$
- N.B., for continuous inputs, the decoder is linear and so the optimum autoencoder is just equivalent to PCA

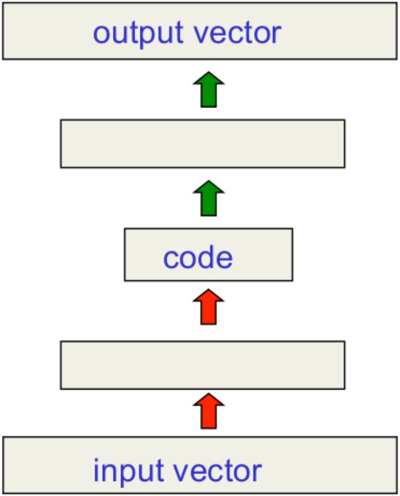
Deep autoencoders

Hinton & Salakhutdinov, Science 2006



- When using multiple layers, PCA is no longer optimal for continuous input
- The introduced nonlinearity can efficiently represent data that lies on a non-linear manifold
- It was an old idea (dated back to 80's) but it was considered to be very hard to train

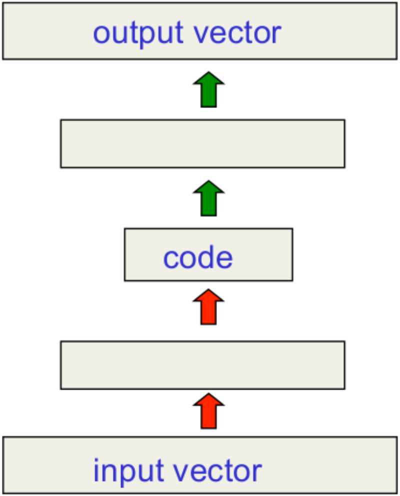
Deep autoencoders
Hinton & Salakhutdinov, Science 2006



- When using multiple layers, PCA is no longer optimal for continuous input
- The introduced nonlinearity can efficiently represent data that lies on a non-linear manifold
- It was an old idea (dated back to 80's) but it was considered to be very hard to train

Deep autoencoders

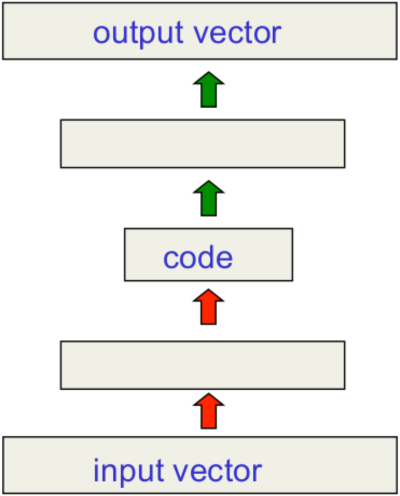
Hinton & Salakhutdinov, Science 2006



- When using multiple layers, PCA is no longer optimal for continuous input
- The introduced nonlinearity can efficiently represent data that lies on a non-linear manifold
- It was an old idea (dated back to 80's) but it was considered to be very hard to train

Deep autoencoders

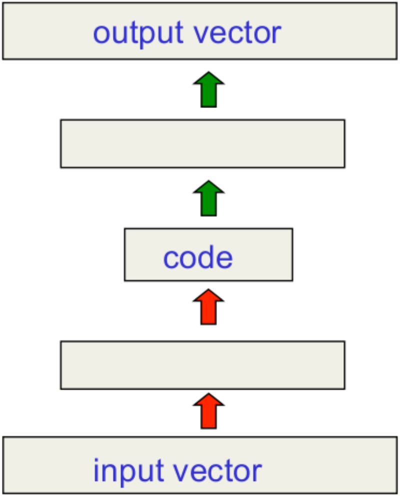
Hinton & Salakhutdinov, Science 2006



- First really successful deep autoencoder was trained in 2006 by Hinton's group
- It uses layer-by-layer RBM pre-training as described in the last lecture
- Just use regular backprob for fine-tuning

Deep autoencoders

Hinton & Salakhutdinov, Science 2006



- First really successful deep autoencoder was trained in 2006 by Hinton's group
- It uses layer-by-layer RBM pre-training as described in the last lecture
- Just use regular backprob for fine-tuning

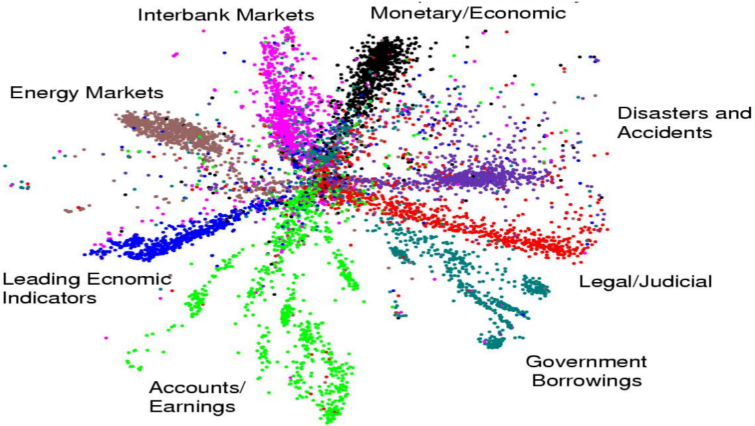
Deep autoencoder vs PCA

From Hinton and Salakhutdinov, Science, 2006

Deep autoencoder for 400,000 business documents

Hinton 2006

First compress all documents to 2 numbers using deep auto. Then use different colors for different document categories



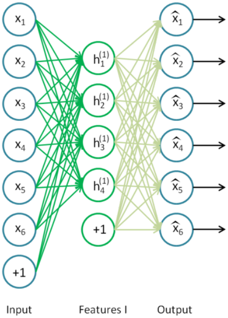
Deep autoencoder for 400,000 image retrieval

Hinton 2006

Leftmost column is the search image.

Other columns are the images that have the most similar feature activities in the last hidden layer.

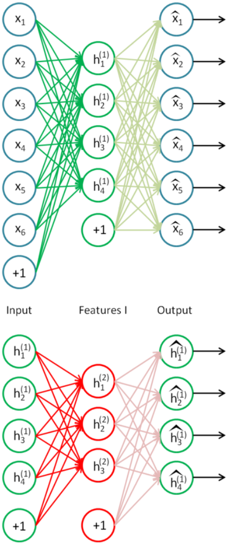
Stacked autoencoders
Alternative pretrianing approach



- Besides pre-training using RBMs, we may also “expand” a deep autoencoders as a stack of shallow autoecoders
- Shallow autoencoders are easier to train than RBM

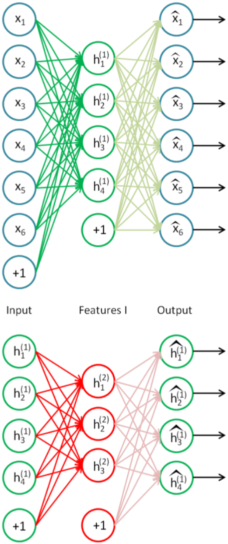
Stacked autoencoders

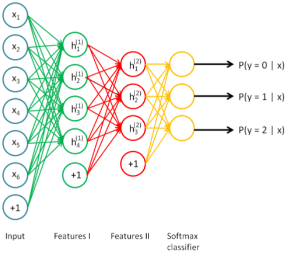
Alternative pretrianing approach



- Besides pre-training using RBMs, we may also “expand” a deep autoencoders as a stack of shallow autoecoders
- Shallow autoencoders are easier to train than RBM

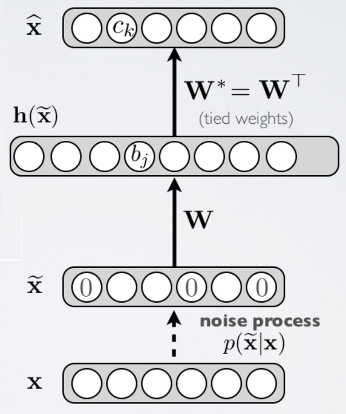
Stacked autoencoders
Alternative pretrianing approach





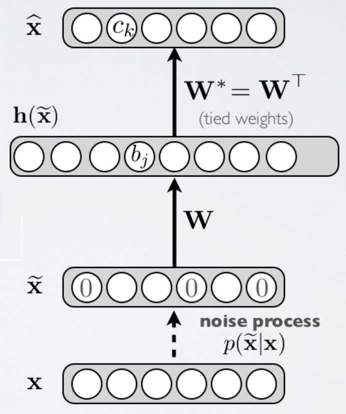
- Besides pre-training using RBMs, we may also “expand” a deep autoencoders as a stack of shallow autoecoders
- Shallow autoencoders are easier to train than RBM

Denoising autoencoders
Vincent *et al.* 2008



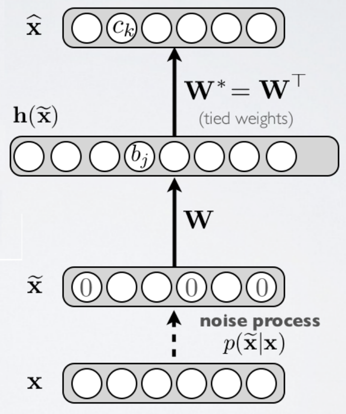
- Idea: representation should be robust to introduction of noise
  - Randomly assign bits to zero for binary case
    - Similar to dropout but for inputs only
  - Gaussian additive noise for continuous case
- Loss function compares $\hat{\mathbf{x}}$ with noiseless input $\mathbf{x}$

Denoising autoencoders
Vincent *et al.* 2008



- Idea: representation should be robust to introduction of noise
  - Randomly assign bits to zero for binary case
    - Similar to dropout but for inputs only
  - Gaussian additive noise for continuous case
- Loss function compares $\hat{\mathbf{x}}$ with noiseless input $\mathbf{x}$

Denoising autoencoders
Vincent *et al.* 2008



- Idea: representation should be robust to introduction of noise
  - Randomly assign bits to zero for binary case
    - Similar to dropout but for inputs only
  - Gaussian additive noise for continuous case
- Loss function compares $\hat{\mathbf{x}}$ with noiseless input $\mathbf{x}$

Contractive autoencoders
Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders
Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
    - + deterministic gradient $\Rightarrow$ can use second order optimizers
    - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
    - - Need to compute Jacobian of hidden layer
    - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders

Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \rightarrow l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders
Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders
Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \rightarrow l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient ⇒ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Contractive autoencoders

Rifai *et al.* 2011

- Idea: encourage robustness of the model by forcing the hidden units to be insensitive to slight change of inputs
- Achieve this by penalizing the squared gradient of each hidden activity w.r.t. the inputs

$$l(\mathbf{x}) \to l(\mathbf{x}) + \lambda \|\nabla_{\mathbf{x}} h(\mathbf{x})\|_F^2$$

- Pros and cons
  - + deterministic gradient $\Rightarrow$ can use second order optimizers
  - + could be more stable than denoising autoencoder, which needs to use a sampled gradient
  - - Need to compute Jacobian of hidden layer
  - - More complex than denoising autoencoder, which just needs to add one two lines of code

Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, DBNs and RBMs
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning
- "Generative autoencoders" ⇒ variational autoencoders
  - Instead of spitting out an approximate for the input
  - The decoder spits out parameters of a distribution

Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, DBNs and RBMs
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning
- "Generative autoencoders" $\Rightarrow$ variational autoencoders
  - Instead of spitting out an approximate for the input
  - The decoder spits out parameters of a distribution

Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, DBNs and RBMs
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning
- "Generative autoencoders" $\Rightarrow$ variational autoencoders
  - Instead of spitting out an approximate for the input
  - The decoder spits out parameters of a distribution

Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, DBNs and RBMs
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning
- "Generative autoencoders" ⇒ variational autoencoders
  - Instead of spitting out an approximate for the input
  - The decoder spits out parameters of a distribution

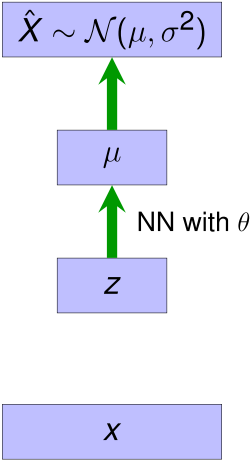
Discriminative models vs generative models

- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, DBNs and RBMs
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning
- "Generative autoencoders" $\Rightarrow$ variational autoencoders
  - Instead of spitting out an approximate for the input
  - The decoder spits out parameters of a distribution

Discriminative models vs generative models

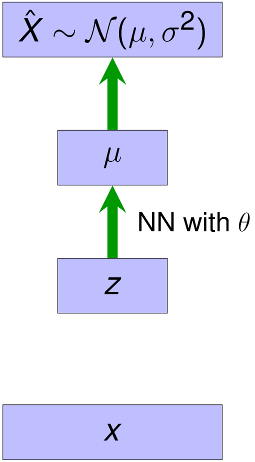
- Discriminative models try to discriminate if one input is different from another. But it is not possible to generate samples from the models. Many classifiers are based on discriminative models, for example, support vector machines
- Generative models on the other hand can generate simulated data, for example, DBNs and RBMs
- Many older machine learning problems are classification problems. Discriminative models provide a more direct solution and thus were more attractive
- Generative models have gained quite some attentions in recent years
  - Generate labeled simulation data for semi-supervised learning
  - Simulate data for planning and reinforcement learning
- "Generative autoencoders" $\Rightarrow$ variational autoencoders
  - Instead of spitting out an approximate for the input
  - The decoder spits out parameters of a distribution

Variational autoencoder
Kingma and Willing 2014



- $p(z|x) = \frac{p(z)p(x|z)}{p(x)} = \frac{p(z)p(x|z)}{\int p(z)p(x|z)dz}$
- For simplicity, pick $p(z) = \mathcal{N}(z; 0, 1)$ and $p(x|z) = \mathcal{N}(\mu, \sigma^2)$, the posterior $p(z|x)$ is still intractable since computing $p(x)$ needs to integrate over all possible z
- We might use MAP or Monte Carlo sampling (MCMC) to estimate $p(z|x)$ but
  - MAP: - too biased
  - MCMC: - too expensive
  - $\Rightarrow$ Variational inference

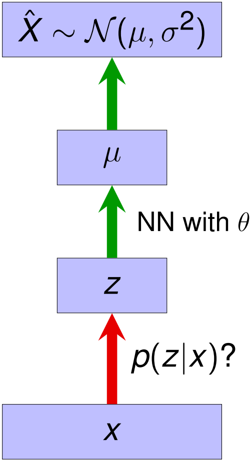
Variational autoencoder
Kingma and Willing 2014



- $p(z|x) = \frac{p(z)p(x|z)}{p(x)} = \frac{p(z)p(x|z)}{\int p(z)p(x|z)dz}$
- For simplicity, pick $p(z) = \mathcal{N}(z; 0, 1)$ and $p(x|z) = \mathcal{N}(\mu, \sigma^2)$, the posterior $p(z|x)$ is still intractable since computing $p(x)$ needs to integrate over all possible z
- We might use MAP or Monte Carlo sampling (MCMC) to estimate $p(z|x)$ but
  - MAP: - too biased
  - MCMC: - too expensive
  - $\Rightarrow$ Variational inference

Variational autoencoder

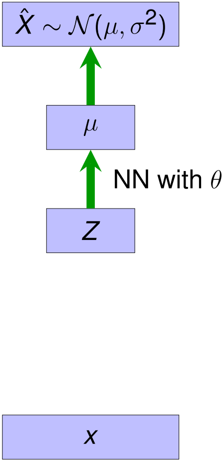
Kingma and Willing 2014



- $p(z|x) = \frac{p(z)p(x|z)}{p(x)} = \frac{p(z)p(x|z)}{\int p(z)p(x|z)dz}$
- For simplicity, pick $p(z) = \mathcal{N}(z; 0, 1)$ and $p(x|z) = \mathcal{N}(\mu, \sigma^2)$, the posterior $p(z|x)$ is still intractable since computing $p(x)$ needs to integrate over all possible z
- We might use MAP or Monte Carlo sampling (MCMC) to estimate $p(z|x)$ but
  - MAP: - too biased
  - MCMC: - too expensive
  - $\Rightarrow$ Variational inference

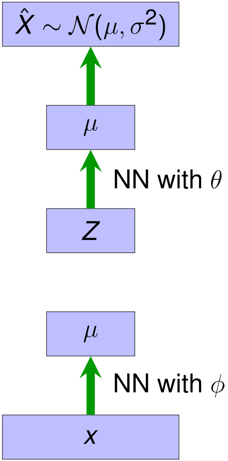
Variational autoencoder

Kingma and Willing 2014



- Instead of trying to find the exact posterior $p(z|x)$, approximate it as a Gaussian distribution with parameters obtained through an NN
- Unfortunately, the loss $-\log p(x)$ is still intractable, but we can approximate $\log p(x)$ with a lower bound
- Instead of minimizing the loss, or maximizing $\log p(x)$ directly, we will maximize its lower bound instead

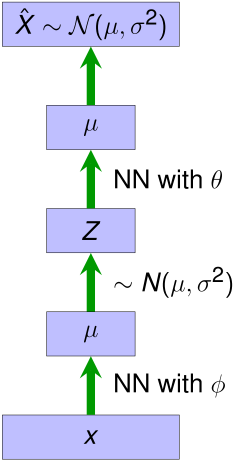
Variational autoencoder
Kingma and Willing 2014



- Instead of trying to find the exact posterior $p(z|x)$, approximate it as a Gaussian distribution with parameters obtained through an NN
- Unfortunately, the loss $-\log p(x)$ is still intractable, but we can approximate $\log p(x)$ with a lower bound
- Instead of minimizing the loss, or maximizing $\log p(x)$ directly, we will maximize its lower bound instead

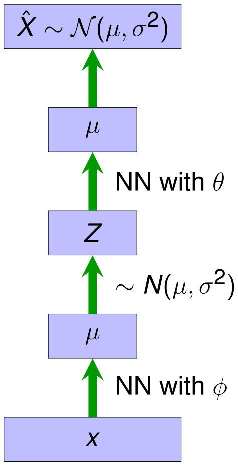
Variational autoencoder

Kingma and Willing 2014



- Instead of trying to find the exact posterior $p(z|x)$, approximate it as a Gaussian distribution with parameters obtained through an NN
- Unfortunately, the loss $-\log p(x)$ is still intractable, but we can approximate $\log p(x)$ with a lower bound
- Instead of minimizing the loss, or maximizing $\log p(x)$ directly, we will maximize its lower bound instead

Variational autoencoder
Kingma and Willing 2014



- Instead of trying to find the exact posterior $p(z|x)$, approximate it as a Gaussian distribution with parameters obtained through an NN
- Unfortunately, the loss $-\log p(x)$ is still intractable, but we can approximate $\log p(x)$ with a lower bound
- Instead of minimizing the loss, or maximizing $\log p(x)$ directly, we will maximize its lower bound instead

Variational lower bound (EBLO)

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)} = \log \frac{p(x|z)p(z)}{p(z|x)} \frac{q(z|x)}{q(z|x)}$$
$$= \log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}$$

Since the above is true for all z,

$$\log p(x) = E_{z\sim q(z|x)}\left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}\right]$$
$$= \underbrace{E_{z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)\|p(z))}_{\text{EBLO}(x,\theta,\phi)\ \text{“Evidence Lower BOund”}} + \underbrace{KL(q(z|x)\,\|p(z|x))}_{\geq 0}$$

Training: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{EBLO}(x^{(i)}, \theta, \phi)$

Variational lower bound (EBLO)

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)} = \log \frac{p(x|z)p(z)}{p(z|x)} \frac{q(z|x)}{q(z|x)}$$
$$= \log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}$$

Since the above is true for all z,

$$\log p(x) = E_{z\sim q(z|x)}\left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}\right]$$
$$= \underbrace{E_{z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)\|p(z))}_{\text{EBLO}(x,\theta,\phi) \text{ “Evidence Lower BOund”}} + \underbrace{KL(q(z|x)\,\|p(z|x))}_{\geq 0}$$

Training: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{EBLO}(x^{(i)}, \theta, \phi)$

Variational lower bound (EBLO)

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)} = \log \frac{p(x|z)p(z)}{p(z|x)} \frac{q(z|x)}{q(z|x)}$$
$$= \log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}$$

Since the above is true for all z,

$$\log p(x) = E_{z\sim q(z|x)}\left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}\right]$$
$$= \underbrace{E_{z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)\|p(z))}_{\text{EBLO}(x,\theta,\phi)\text{ “Evidence Lower BOund”}} + \underbrace{KL(q(z|x)\,\|p(z|x))}_{\geq 0}$$

Training: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{EBLO}(x^{(i)}, \theta, \phi)$

Variational lower bound (EBLO)

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)} = \log \frac{p(x|z)p(z)}{p(z|x)} \frac{q(z|x)}{q(z|x)}$$
$$= \log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}$$

Since the above is true for all z,

$$\log p(x) = E_{z\sim q(z|x)}\left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}\right]$$
$$= \underbrace{E_{z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)\|p(z))}_{\text{EBLO}(x,\theta,\phi)\text{ “Evidence Lower BOund”}} + \underbrace{KL(q(z|x)\,\|p(z|x))}_{\geq 0}$$

Training: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{EBLO}(x^{(i)}, \theta, \phi)$

Variational lower bound (EBLO)

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)} = \log \frac{p(x|z)p(z)}{p(z|x)} \frac{q(z|x)}{q(z|x)}$$
$$= \log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}$$

Since the above is true for all z,

$$\log p(x) = E_{Z\sim q(z|x)} \left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)} \right]$$
$$= \underbrace{E_{Z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)\|p(z))}_{\text{EBLO}(x,\theta,\phi)\text{ “Evidence Lower BOund”}} + \underbrace{KL(q(z|x)\,\|p(z|x))}_{\geq 0}$$

Training: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{EBLO}(x^{(i)}, \theta, \phi)$

Variational lower bound (EBLO)

$$\log p(x) = \log \frac{p(x|z)p(z)}{p(z|x)} = \log \frac{p(x|z)p(z)}{p(z|x)} \frac{q(z|x)}{q(z|x)}$$
$$= \log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}$$

Since the above is true for all z,

$$\log p(x) = E_{Z\sim q(z|x)}\left[\log p(x|z) - \log \frac{q(z|x)}{p(z)} + \log \frac{q(z|x)}{p(z|x)}\right]$$
$$= \underbrace{E_{Z\sim q(z|x)}[\log p(x|z)] - KL(q(z|x)\|p(z))}_{\text{EBLO}(x,\theta,\phi)\text{ “Evidence Lower BOund”}} + \underbrace{KL(q(z|x)\,\|p(z|x))}_{\geq 0}$$

Training: $\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_i \text{EBLO}(x^{(i)}, \theta, \phi)$

Variational autoencoder
Kingma and Willing 2014

Maximizing EBLO means that:

- Want small $KL(q(z|x)||p(z))$ (the difference between the approx distribution from $p(z)$)
  - This turns out to have closed-form solution since we are dealing with Gaussian distributions
- Want large $E_{z\sim q(z|x)}[\log p(x|z)]$ (expected log prob of the evidence with approx distribution)
  - need to backprop through a random node z
  - can be solved by the ”reparametrization trick”

Variational autoencoder
Kingma and Willing 2014

Maximizing EBLO means that:

- Want small $KL(q(z|x)||p(z))$ (the difference between the approx distribution from $p(z)$)
  - This turns out to have closed-form solution since we are dealing with Gaussian distributions
- Want large $E_{Z \sim q(z|x)}[\log p(x|z)]$ (expected log prob of the evidence with approx distribution)
  - need to backprop through a random node z
  - can be solved by the "reparametrization trick"

Variational autoencoder
Kingma and Willing 2014

Maximizing EBLO means that:

- Want small $KL(q(z|x)||p(z))$ (the difference between the approx distribution from $p(z)$)
    - This turns out to have closed-form solution since we are dealing with Gaussian distributions
- Want large $E_{Z\sim q(z|x)}[\log p(x|z)]$ (expected log prob of the evidence with approx distribution)
    - need to backprop through a random node z
    - can be solved by the "reparametrization trick"

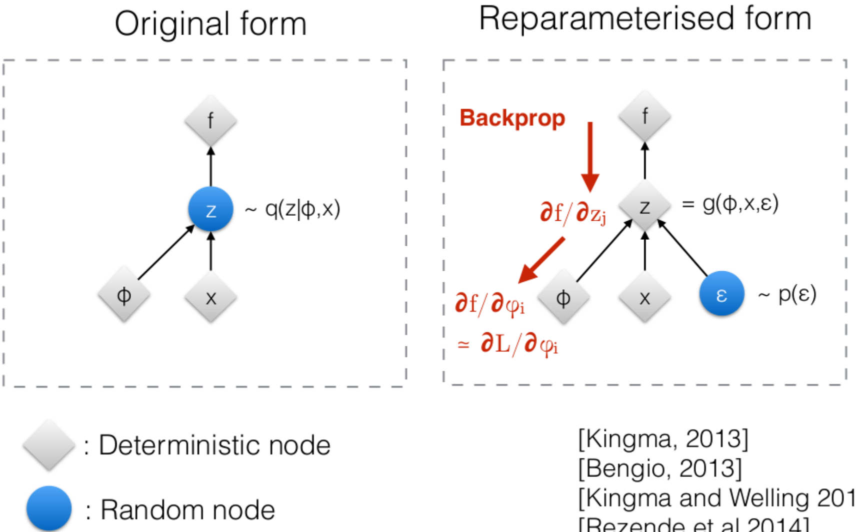
Variational autoencoder

Kingma and Willing 2014

Maximizing EBLO means that:

- Want small $KL(q(z|x)||p(z))$ (the difference between the approx distribution from $p(z)$)
  - This turns out to have closed-form solution since we are dealing with Gaussian distributions
- Want large $E_{Z\sim q(z|x)}[\log p(x|z)]$ (expected log prob of the evidence with approx distribution)
  - need to backprop through a random node z
  - can be solved by the ”reparametrization trick”

Reparametrization trick

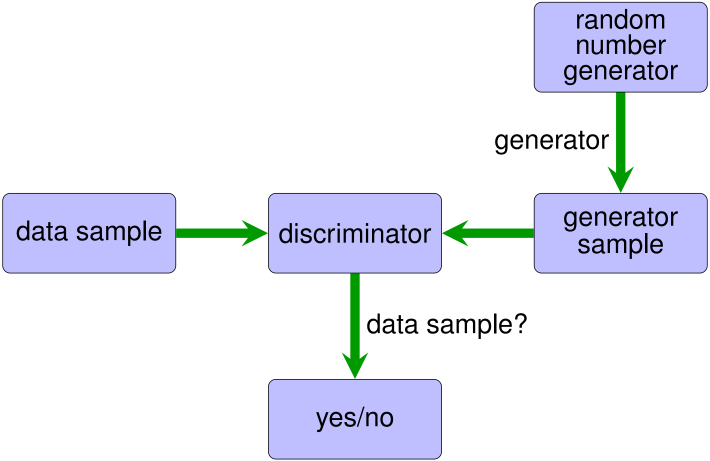


[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

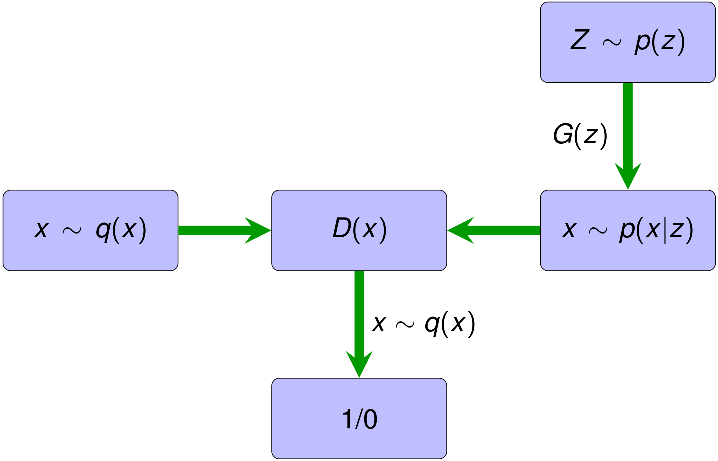
Trained on faces with convnet encoder/decoder
Alec Radford 2015

Generative adversarial networks (GANs)

Goodfellow *et al.* 2014



Generative adversarial networks (GANs)
Goodfellow *et al.* 2014



Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_z \log(1 - D(G(z)))$$
$$= -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$$

- Generator wants to fool the discriminator

$$J^{(G)} = -J^{(D)}$$

Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_z \log(1 - D(G(z)))$$
$$= -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$$

- Generator wants to fool the discriminator

$$J^{(G)} = -J^{(D)}$$

Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_z \log(1 - D(G(z)))$$
$$= -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$$

- Generator wants to fool the discriminator

$$J^{(G)} = -J^{(D)}$$

Minimax game of a GAN

- Probability of model data: $p_{model}(x) = \int_z p(z)p(x|z)dz$
- Probability of true data: $p_{data}(x) = q(x)$
- Discriminator wants to catch fake data

$$J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_z \log(1 - D(G(z)))$$
$$= -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$$

- Generator wants to fool the discriminator

$$J^{(G)} = -J^{(D)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\left.\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\left.\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\left.\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\left.\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+E_{x\sim p_{model}}\left.\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Optimal discriminator $D^*(x)$

By calculus of variations, for any $\Delta(x)$,

$$\left.\frac{\partial J^{(D)}(D^*(X)+\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -\left.\frac{\partial E_{x\sim p_{data}}\log(D^*(x)+\lambda\Delta(x))}{\partial\lambda}-\frac{\partial E_{x\sim p_{model}}\log(1-D^*(x)-\lambda\Delta(x))}{\partial\lambda}\right|_{\lambda=0}=0$$

$$\Rightarrow -E_{x\sim p_{data}}\left[\frac{1}{D^*(x)+\lambda\Delta(x)}\right]+\left.E_{x\sim p_{model}}\left[\frac{1}{1-D^*(x)-\lambda\Delta(x)}\right]\right|_{\lambda=0}=0$$

$$\Rightarrow \int_x\left[\frac{p_{data}(x)}{D^*(x)}-\frac{p_{model}(x)}{1-D^*(x)}\right]dx=0$$

$$\Rightarrow D^*(x)=\frac{p_{data}(x)}{p_{data}(x)+p_{model}(x)}$$

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - The minimax cost $J^{(G)} = -J^{(D)}$ is theoretically appealing but is not the default choice in practice
  - The main problem is that gradient may be small because the gradients contributed by the two terms could cancel each other in some cases
  - The default choice is to maximize only the log-probability of the discriminator being mistake (**non-saturating cost**), i.e.,

$$J^{(G)} = -E_{x\sim p_{model}} \log D(x).$$

This guarantees that the gradient is always non-zero (unless discriminator always get fooled)

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - The minimax cost $J^{(G)} = -J^{(D)}$ is theoretically appealing but is not the default choice in practice
  - The main problem is that gradient may be small because the gradients contributed by the two terms could cancel each other in some cases
  - The default choice is to maximize only the log-probability of the discriminator being mistake (**non-saturating cost**), i.e.,

$$J^{(G)} = -E_{x\sim p_{model}} \log D(x).$$

  This guarantees that the gradient is always non-zero (unless discriminator always get fooled)

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -\frac{1}{2}E_{x\sim p_{data}} \log D(x) - \frac{1}{2}E_{x\sim p_{model}} \log(1 - D(x))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - The minimax cost $J^{(G)} = -J^{(D)}$ is theoretically appealing but is not the default choice in practice
  - The main problem is that gradient may be small because the gradients contributed by the two terms could cancel each other in some cases
  - The default choice is to maximize only the log-probability of the discriminator being mistake (**non-saturating cost**), i.e.,

$$J^{(G)} = -E_{x\sim p_{model}} \log D(x).$$

    This guarantees that the gradient is always non-zero (unless discriminator always get fooled)

Non-saturating cost function

- The discriminator cost function $J^{(D)} = -\frac{1}{2} E_{x \sim p_{data}} \log D(x) - \frac{1}{2} E_{x \sim p_{model}} \log(1 - D(x))$ is a very reasonable choice and usually will not be modified
- On the other hand, we have more freedom on choosing the generator cost
  - The minimax cost $J^{(G)} = -J^{(D)}$ is theoretically appealing but is not the default choice in practice
  - The main problem is that gradient may be small because the gradients contributed by the two terms could cancel each other in some cases
  - The default choice is to maximize only the log-probability of the discriminator being mistake (**non-saturating cost**), i.e.,

$$J^{(G)} = -E_{x \sim p_{model}} \log D(x).$$

    This guarantees that the gradient is always non-zero (unless discriminator always get fooled)

One-sided label smoothing
Salimans *et al.* 2016

- Default discriminator cost can also be written as

$$\begin{aligned}&\text{cross\_entropy}(1,\text{discriminator}(\text{data}))\\&+\text{cross\_entropy}(0,\text{discriminator}(\text{samples}))\end{aligned}$$

- Experiment shows that one-sided label smoothed cost enhance system stability

$$\begin{aligned}&\text{cross\_entropy}(0.9,\text{discriminator}(\text{data}))\\&+\text{cross\_entropy}(0,\text{discriminator}(\text{samples}))\end{aligned}$$

  - Essentially prevent extrapolating effect from extreme samples
  - Generally does not reduce classification accuracy, only confidence

One-sided label smoothing

Salimans *et al.* 2016

- Default discriminator cost can also be written as

$$\text{cross\_entropy}(1,\text{discriminator}(\text{data})) + \text{cross\_entropy}(0, \text{discriminator}(\text{samples}))$$

- Experiment shows that one-sided label smoothed cost enhance system stability

$$\text{cross\_entropy}(0.9,\text{discriminator}(\text{data})) + \text{cross\_entropy}(0, \text{discriminator}(\text{samples}))$$

  - Essentially prevent extrapolating effect from extreme samples
  - Generally does not reduce classification accuracy, only confidence

One-sided label smoothing
Salimans *et al.* 2016

- Default discriminator cost can also be written as

$$\text{cross\_entropy}(1,\text{discriminator}(\text{data}))$$
$$+\text{cross\_entropy}(0, \text{discriminator}(\text{samples}))$$

- Experiment shows that one-sided label smoothed cost enhance system stability

$$\text{cross\_entropy}(0.9,\text{discriminator}(\text{data}))$$
$$+\text{cross\_entropy}(0, \text{discriminator}(\text{samples}))$$

  - Essentially prevent extrapolating effect from extreme samples
  - Generally does not reduce classification accuracy, only confidence

One-sided label smoothing
Salimans *et al.* 2016

- It is important not to smooth the negative labels though, i.e., say

$$\text{cross\_entropy}(1-\alpha, \text{discriminator}(\text{data})) + \text{cross\_entropy}(\beta, \text{discriminator}(\text{samples}))$$

with $\beta > 0$

- Just follow the same derivation as before, we can get the optimum $D(x)$ as

$$D^*(x) = \frac{(1-\alpha)p_{data}(x) + \beta p_{model}(x)}{p_{data}(x) + p_{model}(x)}$$

- Since the numerator has significantly more effect on the peak locations of $D(x)$, consequently affect where the generator will create data. $\beta > 0$ can reinforce undesirable positive feedback

One-sided label smoothing
Salimans *et al.* 2016

- It is important not to smooth the negative labels though, i.e., say

$$\text{cross\_entropy}(1-\alpha,\text{discriminator(data)})$$
$$+\text{cross\_entropy}(\beta,\text{discriminator(samples)})$$

  with $\beta > 0$

- Just follow the same derivation as before, we can get the optimum $D(x)$ as

$$D^*(x) = \frac{(1-\alpha)p_{data}(x) + \beta p_{model}(x)}{p_{data}(x) + p_{model}(x)}$$

- Since the numerator has significantly more effect on the peak locations of $D(x)$, consequently affect where the generator will create data. $\beta > 0$ can reinforce undesirable positive feedback

One-sided label smoothing

Salimans *et al.* 2016

- It is important not to smooth the negative labels though, i.e., say

$$\begin{aligned}&\text{cross\_entropy}(1-\alpha,\text{discriminator(data)})\\+&\text{cross\_entropy}(\beta,\text{discriminator(samples)})\end{aligned}$$

  with $\beta > 0$

- Just follow the same derivation as before, we can get the optimum $D(x)$ as

$$D^*(x) = \frac{(1-\alpha)p_{data}(x) + \beta p_{model}(x)}{p_{data}(x) + p_{model}(x)}$$

- Since the numerator has significantly more effect on the peak locations of $D(x)$, consequently affect where the generator will create data. $\beta > 0$ can reinforce undesirable positive feedback

Issue on batch normalization
Goodfellow 2016

Batch normalization is preferred and highly recommended. But it can cause strong intra-batch correlation

Fixing batch norm

- Reference batch norm: one possible approach is keep one reference batch and always normalized based on that batch. That is, always subtract mean from that of the reference batch and adjust variance to that of the reference batch
  - Can easily overfit to the particular reference batch
- Virtual batch norm: a partial solution by combining the reference batch norm and conventional batch norm. Fix a reference batch, but every time inputs are normalize to the net mean and variance of the virtual batch containing both inputs and all elements of the reference batch

Fixing batch norm

- Reference batch norm: one possible approach is keep one reference batch and always normalized based on that batch. That is, always subtract mean from that of the reference batch and adjust variance to that of the reference batch
  - Can easily overfit to the particular reference batch
- Virtual batch norm: a partial solution by combining the reference batch norm and conventional batch norm. Fix a reference batch, but every time inputs are normalize to the net mean and variance of the virtual batch containing both inputs and all elements of the reference batch

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being "too smart"
  - The original theoretical justification is that D is supposed to be perfect

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being “too smart”
  - The original theoretical justification is that D is supposed to be perfect

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being "too smart"
  - The original theoretical justification is that D is supposed to be perfect

Balancing G and D

- Usually it is more preferable to have a bigger and deeper D
- Some researchers also run more D steps than G steps. The results are mixed though
- Some take home messages
  - Use non-saturating cost
  - Use label smoothing
- Do not try to limit D from being “too smart”
  - The original theoretical justification is that D is supposed to be perfect

Mode collapse
Metz *et al.* 2016

Below demonstrates why D should be smart.

- Basically the minmax and the minmax problem is not the same and can lead to drastically different solutions

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

- D in the inner loop: converge to the correct distribution
- G in the inner loop: place all mass on most likely point

Mode collapse

Metz *et al.* 2016

Below demonstrates why D should be smart.

- Basically the minmax and the minmax problem is not the same and can lead to drastically different solutions

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

- D in the inner loop: converge to the correct distribution
- G in the inner loop: place all mass on most likely point

Minibatch features

Salimans *et al.* 2016

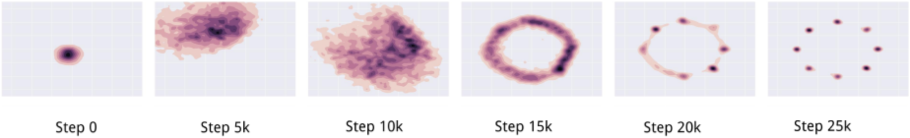
- Mode collapse can lead to low diversity of generated data
- One attempt to mitigate this problem is to introduce the so-called minibatch features
  - Basically classify each example by comparing the features to other members in the minibatch
  - Reject a sample if the feature to close to existing ones

Minibatch features
Salimans *et al.* 2016

- Mode collapse can lead to low diversity of generated data
- One attempt to mitigate this problem is to introduce the so-called minibatch features
  - Basically classify each example by comparing the features to other members in the minibatch
  - Reject a sample if the feature to close to existing ones

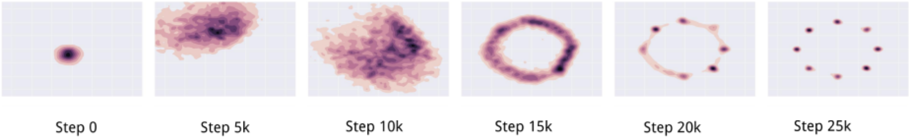
Unrolled Gans
Metz *et al.* 2016

- A more direct approach was proposed by Google brain
- Trying to ensure that the generated sample is a solution of the minmax rather than the maxmin problem
- Have the generator to unroll k future steps and predict what discriminator will think of the current sample
  - Since generator is the one who unrolls, generator is in the outer loop and discriminator is in the inner loop
  - We ensure that we have solution approximating a minmax rather than maxmin problem



Unrolled Gans
Metz *et al.* 2016

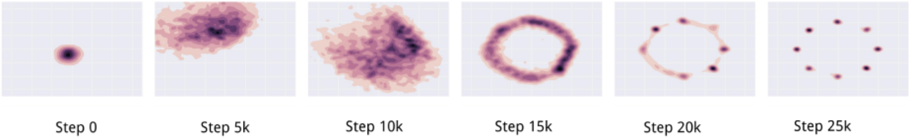
- A more direct approach was proposed by Google brain
- Trying to ensure that the generated sample is a solution of the minmax rather than the maxmin problem
- Have the generator to unroll *k* future steps and predict what discriminator will think of the current sample
  - Since generator is the one who unrolls, generator is in the outer loop and discriminator is in the inner loop
  - We ensure that we have solution approximating a minmax rather than maxmin problem



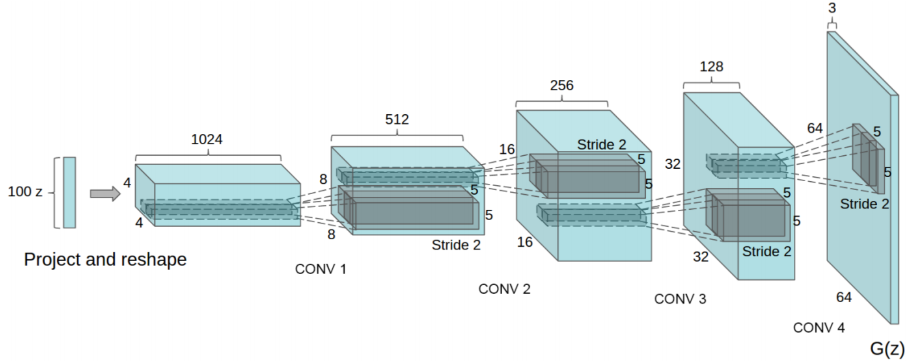
Unrolled Gans

Metz *et al.* 2016

- A more direct approach was proposed by Google brain
- Trying to ensure that the generated sample is a solution of the minmax rather than the maxmin problem
- Have the generator to unroll k future steps and predict what discriminator will think of the current sample
  - Since generator is the one who unrolls, generator is in the outer loop and discriminator is in the inner loop
  - We ensure that we have solution approximating a minmax rather than maxmin problem



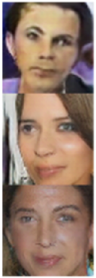
Deep convolutional GAN (DCGAN)
Radford *et al.* 2016



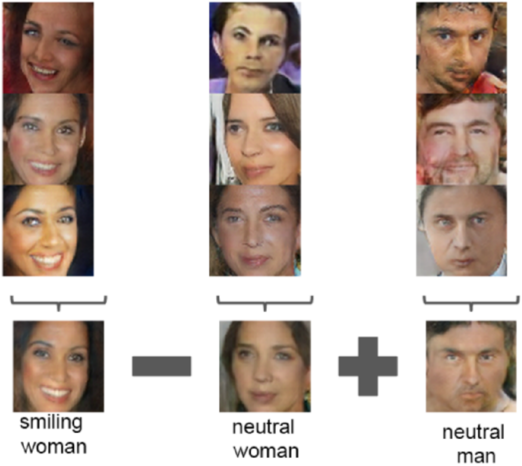
Generated bedroom after 5 epochs (LSUN dataset)

Radford *et al.* 2016

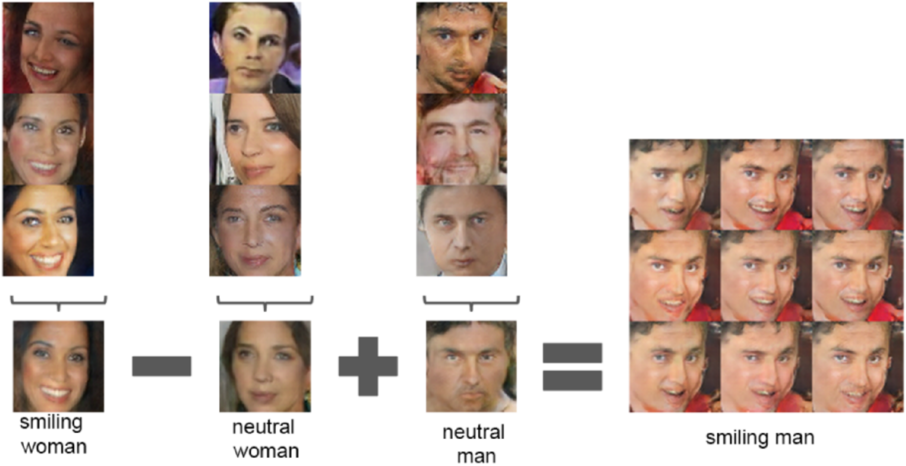
Vector arithmetics
Radford *et al.* 2016


Vector arithmetics
Radford *et al.* 2016



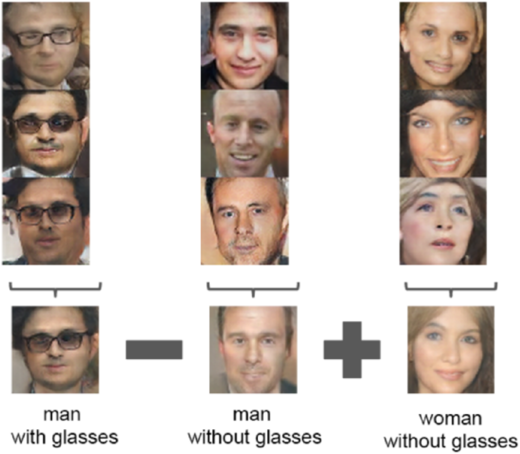
Vector arithmetics
Radford *et al.* 2016



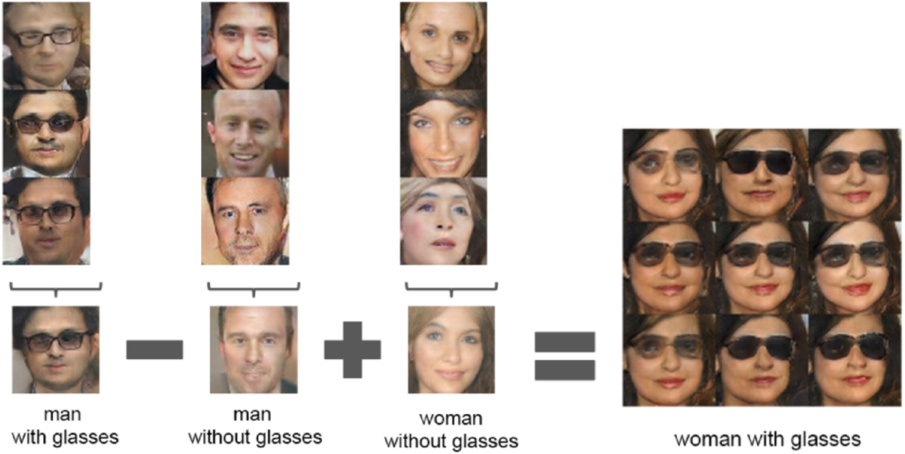
Vector arithmetics
Radford *et al.* 2016



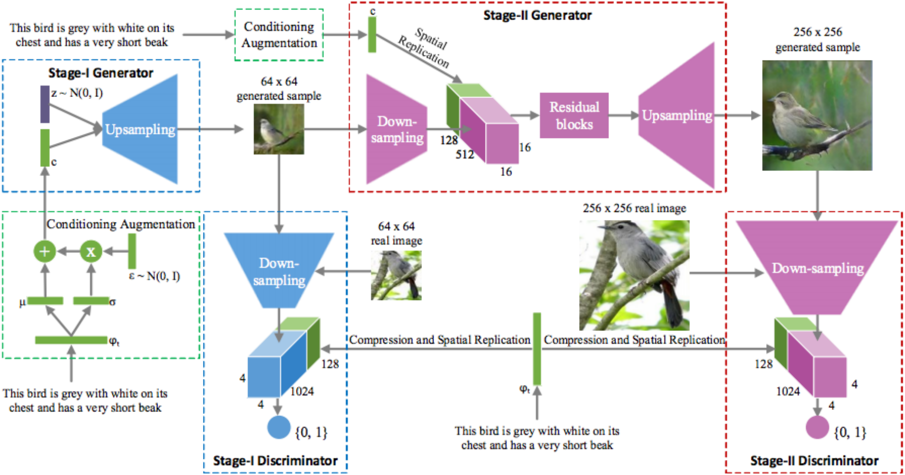
Vector arithmetics
Radford *et al.* 2016



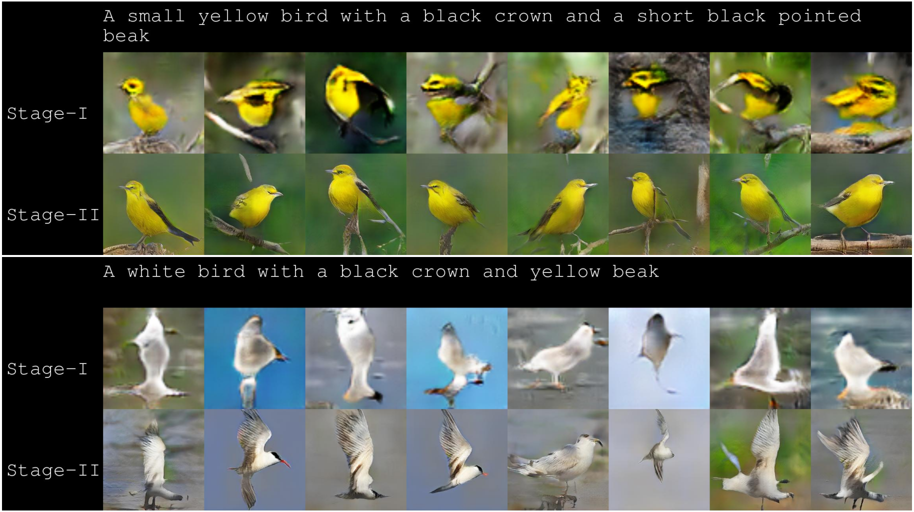
Vector arithmetics
Radford *et al.* 2016



StackGAN
Zhang et al. 2016



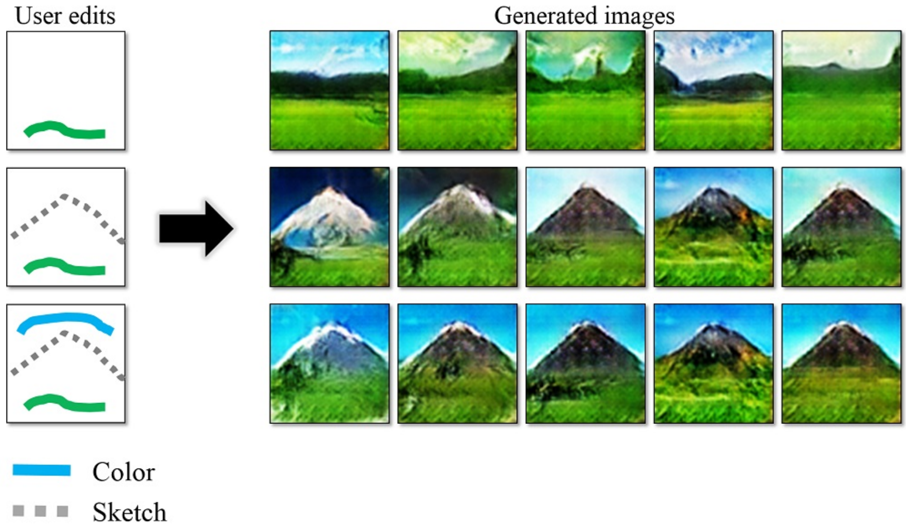
StackGAN


A small yellow bird with a black crown and a short black pointed beak
Stage-I
Stage-II
A white bird with a black crown and yellow beak
Stage-I
Stage-II

StackGAN


This flower has long thin yellow petals and a lot of yellow anthers in the center
Stage-I
Stage-II
This flower is white, pink, and yellow in color, and has petals that are multi colored
Stage-I
Stage-II

iGAN
Zhu *et al.* 2016



iGAN
Zhu *et al.* 2016

Demo

Conclusions

- Conventional autoencoders are important tools for dimension reduction and data representation in general
- Generative models are some very exciting hot topics in deep learning
  - Especially useful for datasets with few or no labels
  - Many other possible applications to be discovered
- We discuss two state-of-the-art generative models
  - Variational autoencoders: autoencoders + variational inference
  - Generative adversarial networks (GANs): more recent and gaining lots of interests

Conclusions

- Conventional autoencoders are important tools for dimension reduction and data representation in general
- Generative models are some very exciting hot topics in deep learning
  - Especially useful for datasets with few or no labels
  - Many other possible applications to be discovered
- We discuss two state-of-the-art generative models
  - Variational autoencoders: autoencoders + variational inference
  - Generative adversarial networks (GANs): more recent and gaining lots of interests

Conclusions

- Conventional autoencoders are important tools for dimension reduction and data representation in general
- Generative models are some very exciting hot topics in deep learning
    - Especially useful for datasets with few or no labels
    - Many other possible applications to be discovered
- We discuss two state-of-the-art generative models
    - Variational autoencoders: autoencoders + variational inference
    - Generative adversarial networks (GANs): more recent and gaining lots of interests