Autoencoders, Extensions, and Applications

Piyush Rai

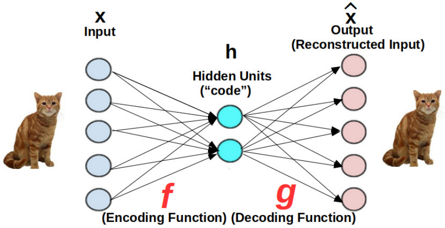
IIT Kanpur

Outline

- Introduction to Autoencoders
- Autoencoder Variants and Extensions
- Some Applications of Autoencoders
- Autoencoders for Recommender Systems

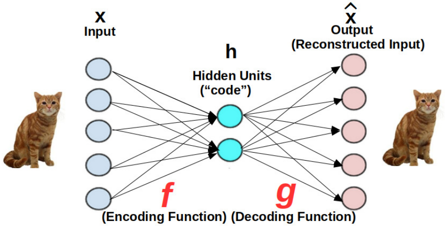
Autoencoder

- Similar to the standard feedforward neural network with a key difference:
  - Unsupervised. No "label" at the output layer; Output layer simply tries to "recreate" the input



Autoencoder

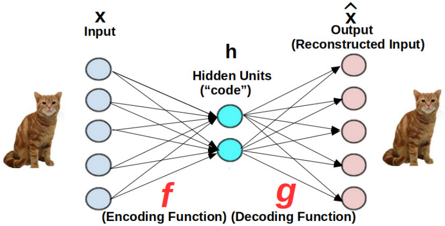
- Similar to the standard feedforward neural network with a key difference:
  - Unsupervised. No "label" at the output layer; Output layer simply tries to "recreate" the input



- Defined by two (possibly nonlinear) mapping functions: Encoding function f, Decoding function g

Autoencoder

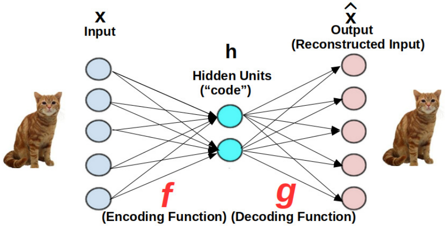
- Similar to the standard feedforward neural network with a key difference:
  - Unsupervised. No "label" at the output layer; Output layer simply tries to "recreate" the input



- Defined by two (possibly nonlinear) mapping functions: Encoding function f, Decoding function g
- $\boldsymbol{h} = f(\boldsymbol{x})$ denotes an encoding (possibly nonlinear) for the input $\boldsymbol{x}$

Autoencoder

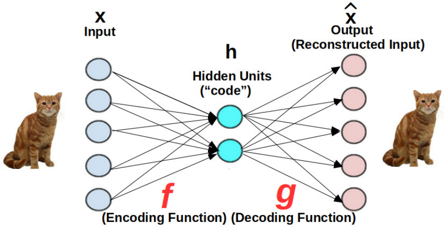
- Similar to the standard feedforward neural network with a key difference:
  - Unsupervised. No "label" at the output layer; Output layer simply tries to "recreate" the input



- Defined by two (possibly nonlinear) mapping functions: Encoding function f, Decoding function g
- $\boldsymbol{h} = f(\boldsymbol{x})$ denotes an encoding (possibly nonlinear) for the input $\boldsymbol{x}$
- $\hat{\boldsymbol{x}} = g(\boldsymbol{h}) = g(f(\boldsymbol{x}))$ denotes the reconstruction (or the "decoding") for the input $\boldsymbol{x}$

Autoencoder

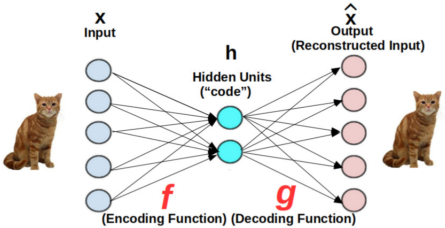
- Similar to the standard feedforward neural network with a key difference:
  - Unsupervised. No "label" at the output layer; Output layer simply tries to "recreate" the input



- Defined by two (possibly nonlinear) mapping functions: Encoding function f, Decoding function g
- $\boldsymbol{h} = f(\boldsymbol{x})$ denotes an encoding (possibly nonlinear) for the input $\boldsymbol{x}$
- $\hat{\boldsymbol{x}} = g(\boldsymbol{h}) = g(f(\boldsymbol{x}))$ denotes the reconstruction (or the "decoding") for the input $\boldsymbol{x}$
- For an Autoencoder, f and g are learned with a goal to minimize the difference between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$

Autoencoder for Feature Learning

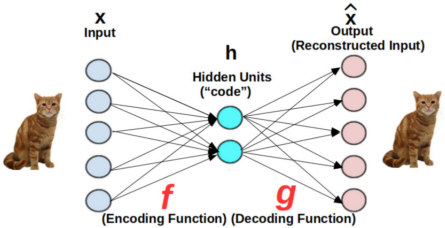
- The learned code $\boldsymbol{h} = f(\boldsymbol{x})$ can be used as a new feature representation of the input $\boldsymbol{x}$



- Therefore autoencoders can also be used for "feature learning"

Autoencoder for Feature Learning

- The learned code $\boldsymbol{h} = f(\boldsymbol{x})$ can be used as a new feature representation of the input $\boldsymbol{x}$



- Therefore autoencoders can also be used for "feature learning"
- Note: Size of the hidden units (encoding) can also be larger than the input

A Simple Autoencoder

- Let's assume a $D \times 1$ input $\boldsymbol{x} \in \mathbb{R}^D$, and a single hidden layer with $K \times 1$ code $\boldsymbol{h} \in \mathbb{R}^K$
- We can then define a simple linear autoencoder as

$$\begin{aligned}\boldsymbol{h} &= f(\boldsymbol{x}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \\ \hat{\boldsymbol{x}} &= g(\boldsymbol{h}) = \mathbf{W}^*\boldsymbol{h} + \boldsymbol{c}\end{aligned}$$

where f is defined by $\mathbf{W} \in \mathbb{R}^{K \times D}$ and $\boldsymbol{b} \in \mathbb{R}^{K \times 1}$

A Simple Autoencoder

- Let's assume a $D \times 1$ input $\boldsymbol{x} \in \mathbb{R}^D$, and a single hidden layer with $K \times 1$ code $\boldsymbol{h} \in \mathbb{R}^K$
- We can then define a simple linear autoencoder as

$$
\begin{aligned}
\boldsymbol{h} &= f(\boldsymbol{x}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \\
\hat{\boldsymbol{x}} &= g(\boldsymbol{h}) = \mathbf{W}^* \boldsymbol{h} + \boldsymbol{c}
\end{aligned}
$$

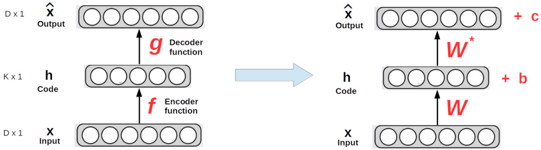
where f is defined by $\mathbf{W} \in \mathbb{R}^{K \times D}$ and $\boldsymbol{b} \in \mathbb{R}^{K \times 1}$, g is defined by $\mathbf{W}^* \in \mathbb{R}^{D \times K}$ and $\boldsymbol{c} \in \mathbb{R}^{D \times 1}$

A Simple Autoencoder

- Let's assume a $D \times 1$ input $\boldsymbol{x} \in \mathbb{R}^D$, and a single hidden layer with $K \times 1$ code $\boldsymbol{h} \in \mathbb{R}^K$
- We can then define a simple linear autoencoder as

$$\begin{aligned}\boldsymbol{h} &= f(\boldsymbol{x}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \\ \hat{\boldsymbol{x}} &= g(\boldsymbol{h}) = \mathbf{W}^*\boldsymbol{h} + \boldsymbol{c}\end{aligned}$$

where f is defined by $\mathbf{W} \in \mathbb{R}^{K \times D}$ and $\boldsymbol{b} \in \mathbb{R}^{K \times 1}$, g is defined by $\mathbf{W}^* \in \mathbb{R}^{D \times K}$ and $\boldsymbol{c} \in \mathbb{R}^{D \times 1}$

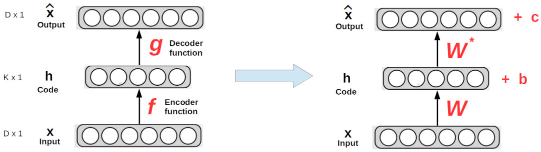


A Simple Autoencoder

- Let's assume a $D \times 1$ input $\boldsymbol{x} \in \mathbb{R}^D$, and a single hidden layer with $K \times 1$ code $\boldsymbol{h} \in \mathbb{R}^K$
- We can then define a simple linear autoencoder as

$$
\begin{aligned}
\boldsymbol{h} &= f(\boldsymbol{x}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \\
\hat{\boldsymbol{x}} &= g(\boldsymbol{h}) = \mathbf{W}^*\boldsymbol{h} + \boldsymbol{c}
\end{aligned}
$$

where f is defined by $\mathbf{W} \in \mathbb{R}^{K \times D}$ and $\boldsymbol{b} \in \mathbb{R}^{K \times 1}$, g is defined by $\mathbf{W}^* \in \mathbb{R}^{D \times K}$ and $\boldsymbol{c} \in \mathbb{R}^{D \times 1}$

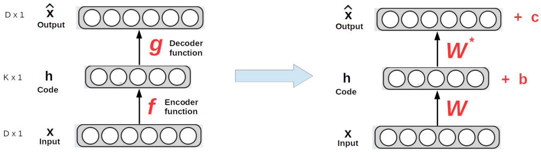


- Note: If we learn f, g to minimize the squared error $||\hat{\boldsymbol{x}} - \boldsymbol{x}||^2$ then the linear autoencoder with $\mathbf{W}^* = \mathbf{W}^\top$ is optimal

- Let's assume a $D \times 1$ input $\boldsymbol{x} \in \mathbb{R}^D$, and a single hidden layer with $K \times 1$ code $\boldsymbol{h} \in \mathbb{R}^K$
- We can then define a simple linear autoencoder as

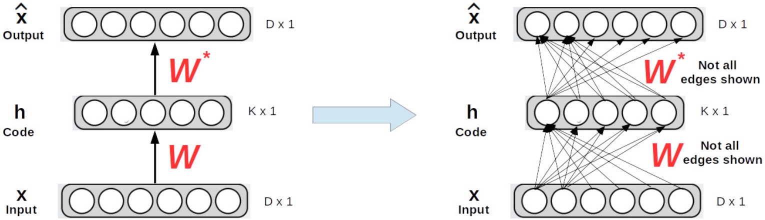
$$\begin{aligned} \boldsymbol{h} &= f(\boldsymbol{x}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \\ \hat{\mathbf{x}} &= g(\boldsymbol{h}) = \mathbf{W}^*\boldsymbol{h} + \boldsymbol{c} \end{aligned}$$

where f is defined by $\mathbf{W} \in \mathbb{R}^{K \times D}$ and $\boldsymbol{b} \in \mathbb{R}^{K \times 1}$, g is defined by $\mathbf{W}^* \in \mathbb{R}^{D \times K}$ and $\boldsymbol{c} \in \mathbb{R}^{D \times 1}$

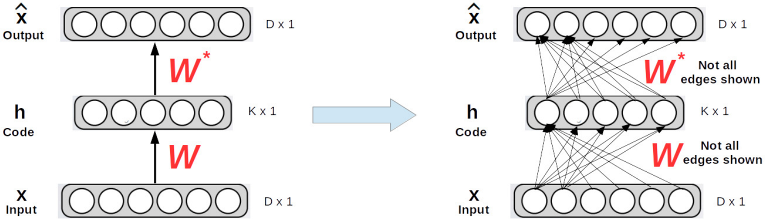


- Note: If we learn f, g to minimize the squared error $||\hat{\mathbf{x}} - \boldsymbol{x}||^2$ then the linear autoencoder with $\mathbf{W}^* = \mathbf{W}^\top$ is optimal, and is equivalent to Principal Component Analysis (PCA)

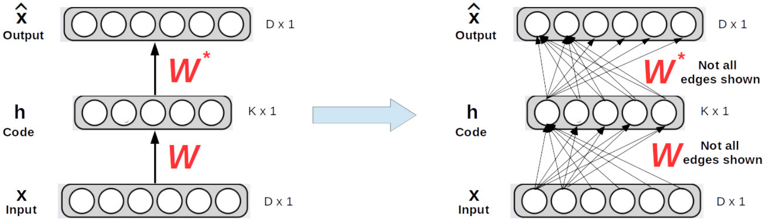
Autoencoder: Zooming in..



Autoencoder: Zooming in..

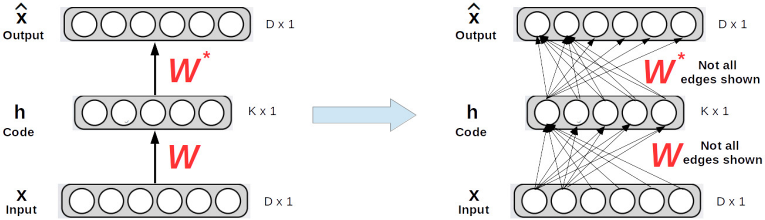


- **W**: $K \times D$ matrix of weights of edges between input and hidden layer



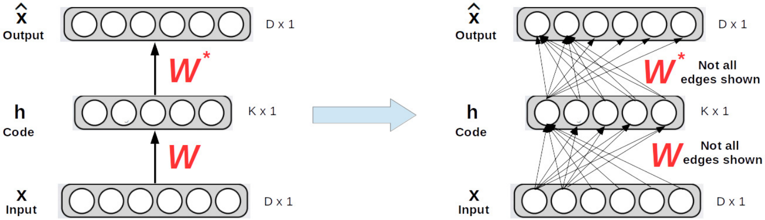
- **W**: $K \times D$ matrix of weights of edges between input and hidden layer
  - W_{kd} is the weight of edge connecting input layer node d to hidden layer node k

Autoencoder: Zooming in..



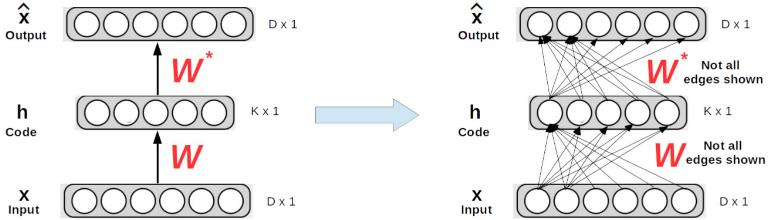
- **W**: $K \times D$ matrix of weights of edges between input and hidden layer
  - W_{kd} is the weight of edge connecting input layer node d to hidden layer node k
- $\mathbf{W}^*$: $D \times K$ matrix of weights of edges between hidden and output layer

Autoencoder: Zooming in..



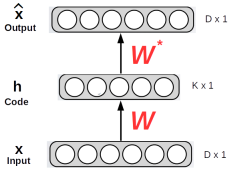
- **W**: $K \times D$ matrix of weights of edges between input and hidden layer
  - W_{kd} is the weight of edge connecting input layer node d to hidden layer node k
- **W***: $D \times K$ matrix of weights of edges between hidden and output layer
  - W^*_{dk} is the weight of edge connecting hidden layer node k to output layer node d

Autoencoder: Zooming in..



- $\mathbf{W}$: $K \times D$ matrix of weights of edges between input and hidden layer
  - W_{kd} is the weight of edge connecting input layer node d to hidden layer node k
- $\mathbf{W}^*$: $D \times K$ matrix of weights of edges between hidden and output layer
  - W^*_{dk} is the weight of edge connecting hidden layer node k to output layer node d
- If $\mathbf{W}^* = \mathbf{W}^\top$, the autoencoder architecture is said to have "tied weights"

Nonlinear Autoencoders

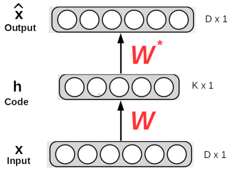


- The hidden nodes can also be nonlinear transforms of the inputs, e.g.,
  - Can define $\boldsymbol{h}$ as a linear transform of $\boldsymbol{x}$ followed by a nonlinearity (e.g., sigmoid, ReLU)

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

where the nonlinearity $\text{sigmoid}(z) = \frac{1}{1+\exp(-z)}$ squashes the real-valued z to lie between 0 and 1

Nonlinear Autoencoders



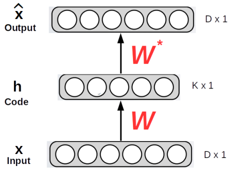
- The hidden nodes can also be nonlinear transforms of the inputs, e.g.,
  - Can define $\boldsymbol{h}$ as a linear transform of $\boldsymbol{x}$ followed by a nonlinearity (e.g., sigmoid, ReLU)

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

  where the nonlinearity $\text{sigmoid}(z) = \frac{1}{1+\exp(-z)}$ squashes the real-valued z to lie between 0 and 1

- Most commonly used autoencoders use such nonlinear transforms

Nonlinear Autoencoders



- The hidden nodes can also be nonlinear transforms of the inputs, e.g.,
  - Can define $\boldsymbol{h}$ as a linear transform of $\boldsymbol{x}$ followed by a nonlinearity (e.g., sigmoid, ReLU)

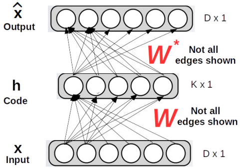
$$\boldsymbol{h} = \text{sigmoid}(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

  where the nonlinearity $\text{sigmoid}(z) = \frac{1}{1+\exp(-z)}$ squashes the real-valued z to lie between 0 and 1

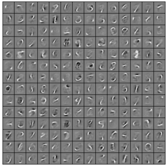
- Most commonly used autoencoders use such nonlinear transforms
- Note: If inputs $\boldsymbol{x} \in \{0,1\}^D$ are binary, it may be more appropriate to also define $\hat{\boldsymbol{x}}$ as

$$\hat{\boldsymbol{x}} = \text{sigmoid}(\mathbf{W}^*\boldsymbol{h} + \boldsymbol{c})$$

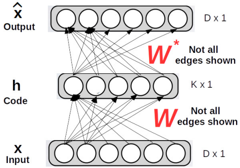
What's Learned by an Autoencoder?



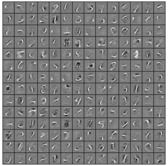
- Figure below: The $K \times D$ matrix $\mathbf{W}$ learned on digits data. Each tiny block visualizes a row of $\mathbf{W}$



What's Learned by an Autoencoder?

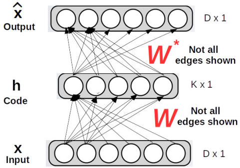


- Figure below: The $K \times D$ matrix $\mathbf{W}$ learned on digits data. Each tiny block visualizes a row of $\mathbf{W}$

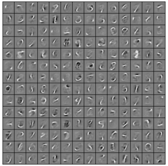


- Thus $\mathbf{W}$ captures the possible "patterns" in the training data (akin to the K basis vectors in PCA)

What's Learned by an Autoencoder?



- Figure below: The $K \times D$ matrix $\mathbf{W}$ learned on digits data. Each tiny block visualizes a row of $\mathbf{W}$



- Thus $\mathbf{W}$ captures the possible "patterns" in the training data (akin to the K basis vectors in PCA)
- For any input $\boldsymbol{x}$, the encoding $\boldsymbol{h}$ tells us how much each of these K features in present in $\boldsymbol{x}$

Training the Autoencoder

- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$

Training the Autoencoder

- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$
- The loss function (a function of parameters $\mathbf{W}$, $\boldsymbol{b}$, $\mathbf{W}^*$, $\boldsymbol{c}$) can be defined using various ways

Training the Autoencoder

- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$
- The loss function (a function of parameters $\mathbf{W}$, $\boldsymbol{b}$, $\mathbf{W}^*$, $\boldsymbol{c}$) can be defined using various ways
- In general, it is defined in terms of the difference between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$ (reconstruction error)

Training the Autoencoder

- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$
- The loss function (a function of parameters $\mathbf{W}, \boldsymbol{b}, \mathbf{W}^*, \boldsymbol{c}$) can be defined using various ways
- In general, it is defined in terms of the difference between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$ (reconstruction error)
- For a single input $\boldsymbol{x} = [x_1, \ldots, x_D]$ and its reconstruction $\hat{\boldsymbol{x}} = [\hat{x}_1, \ldots, \hat{x}_D]$

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = \sum_{d=1}^{D} (\hat{x}_d - x_d)^2$$

(squared loss; used if input are real-valued)

Training the Autoencoder

- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$
- The loss function (a function of parameters $\mathbf{W}, \boldsymbol{b}, \mathbf{W}^*, \boldsymbol{c}$) can be defined using various ways
- In general, it is defined in terms of the difference between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$ (reconstruction error)
- For a single input $\boldsymbol{x} = [x_1, \ldots, x_D]$ and its reconstruction $\hat{\boldsymbol{x}} = [\hat{x}_1, \ldots, \hat{x}_D]$

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = \sum_{d=1}^{D} (\hat{x}_d - x_d)^2$$

(squared loss; used if input are real-valued)

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = -\sum_{d=1}^{D} [x_d \log(\hat{x}_d) + (1 - x_d)\log(1 - \hat{x}_d)]$$

(cross-entropy loss; used if input are binary)

Training the Autoencoder

- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$
- The loss function (a function of parameters $\mathbf{W}, \boldsymbol{b}, \mathbf{W}^*, \boldsymbol{c}$) can be defined using various ways
- In general, it is defined in terms of the difference between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$ (reconstruction error)
- For a single input $\boldsymbol{x} = [x_1, \ldots, x_D]$ and its reconstruction $\hat{\boldsymbol{x}} = [\hat{x}_1, \ldots, \hat{x}_D]$

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = \sum_{d=1}^{D} (\hat{x}_d - x_d)^2$$

(squared loss; used if input are real-valued)

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = -\sum_{d=1}^{D} [x_d \log(\hat{x}_d) + (1 - x_d) \log(1 - \hat{x}_d)]$$

(cross-entropy loss; used if input are binary)

- We find $(\mathbf{W}, \boldsymbol{b}, \mathbf{W}^*, \boldsymbol{c})$ by minimizing the reconstruction error (summed over all training data)

Training the Autoencoder

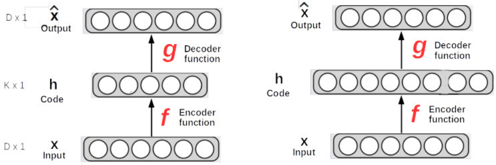
- To train the autoencoder, we need to define a loss function $\ell(\hat{\boldsymbol{x}}, \boldsymbol{x})$
- The loss function (a function of parameters $\mathbf{W}, \boldsymbol{b}, \mathbf{W}^*, \boldsymbol{c}$) can be defined using various ways
- In general, it is defined in terms of the difference between $\hat{\boldsymbol{x}}$ and $\boldsymbol{x}$ (reconstruction error)
- For a single input $\boldsymbol{x} = [x_1, \ldots, x_D]$ and its reconstruction $\hat{\boldsymbol{x}} = [\hat{x}_1, \ldots, \hat{x}_D]$

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = \sum_{d=1}^{D} (\hat{x}_d - x_d)^2 \quad \text{(squared loss; used if input are real-valued)}$$

$$\ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) = -\sum_{d=1}^{D} [x_d \log(\hat{x}_d) + (1 - x_d) \log(1 - \hat{x}_d)] \quad \text{(cross-entropy loss; used if input are binary)}$$

- We find $(\mathbf{W}, \boldsymbol{b}, \mathbf{W}^*, \boldsymbol{c})$ by minimizing the reconstruction error (summed over all training data)
- This can be done using backpropagation

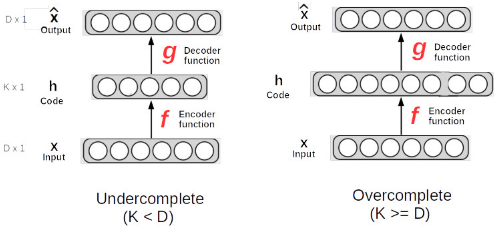
Undercomplete, Overcomplete, and Need for Regularization



Undercomplete (K < D)

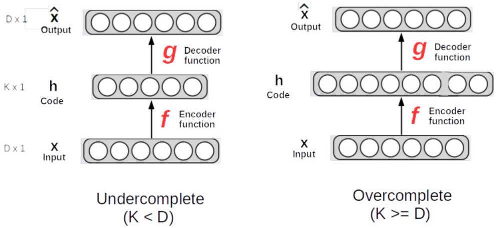
Overcomplete (K >= D)

Undercomplete, Overcomplete, and Need for Regularization



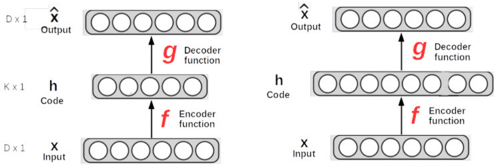
- In both cases, it is important to control the capacity of encoder and decoder

Undercomplete, Overcomplete, and Need for Regularization



- In both cases, it is important to control the capacity of encoder and decoder
- **Undercomplete:** Imagine $K = 1$ and very powerful f and g. Can achieve very small reconstruction error but the learned code will not capture any interesting properties in the data

Undercomplete, Overcomplete, and Need for Regularization

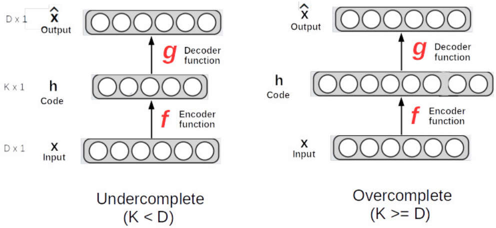


Undercomplete (K < D)

Overcomplete (K >= D)

- In both cases, it is important to control the capacity of encoder and decoder
- **Undercomplete:** Imagine $K = 1$ and very powerful f and g. Can achieve very small reconstruction error but the learned code will not capture any interesting properties in the data
- **Overcomplete:** Imagine $K \geq D$ and trivial (identity) functions f and g. Can achieve even zero reconstruction error but the learned code will not capture any interesting properties in the data

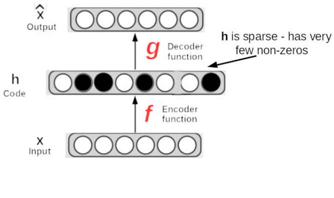
Undercomplete, Overcomplete, and Need for Regularization



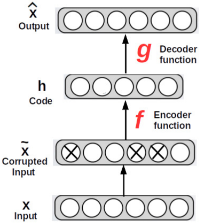
- In both cases, it is important to control the capacity of encoder and decoder
- **Undercomplete:** Imagine $K = 1$ and very powerful f and g. Can achieve very small reconstruction error but the learned code will not capture any interesting properties in the data
- **Overcomplete:** Imagine $K \geq D$ and trivial (identity) functions f and g. Can achieve even zero reconstruction error but the learned code will not capture any interesting properties in the data
- It is therefore important to regularize the functions as well as the learned code, and not just focus on minimizing the reconstruction error

Regularized Autoencoders

- Several ways to regularize the model, e.g.
  - Make the learned code sparse (Sparse Autoencoders)
  - Make the model robust against noisy/incomplete inputs (Denoising Dutoencoders)



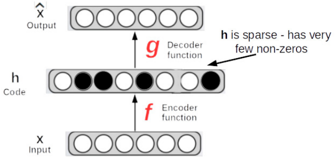
Sparse Autoencoder



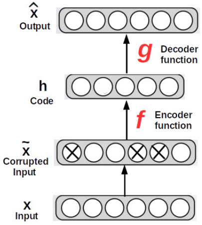
Denoising Autoencoder

Regularized Autoencoders

- Several ways to regularize the model, e.g.
  - Make the learned code sparse (Sparse Autoencoders)
  - Make the model robust against noisy/incomplete inputs (Denoising Dutoencoders)



Sparse Autoencoder



Denoising Autoencoder

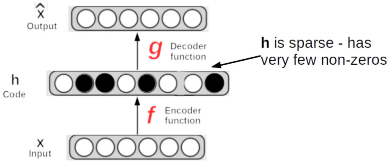
  - Make the model robust against small changes in the input (Contractive Autoencoders)

Sparse Autoencoders

- Make the learned code sparse (Sparse Autoencoders). Done by adding a sparsity penalty on $\boldsymbol{h}$

$$\text{Loss Function:} \quad \ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) + \Omega(\boldsymbol{h})$$

where $\Omega(\boldsymbol{h}) = \sum_{k=1}^{K} |h_k|$ is the ℓ_1 norm of $\boldsymbol{h}$

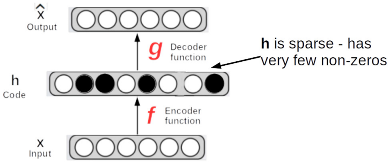


Sparse Autoencoders

- Make the learned code sparse (Sparse Autoencoders). Done by adding a sparsity penalty on $\boldsymbol{h}$

$$\text{Loss Function:} \quad \ell(\hat{\boldsymbol{x}}, \boldsymbol{x}) + \Omega(\boldsymbol{h})$$

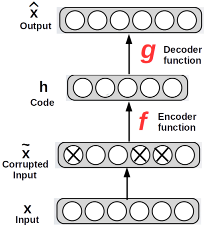
where $\Omega(\boldsymbol{h}) = \sum_{k=1}^{K} |h_k|$ is the ℓ_1 norm of $\boldsymbol{h}$



- Sparse autoencoder is learned by minimizing the above regularized loss function

Denoising Autoencoders

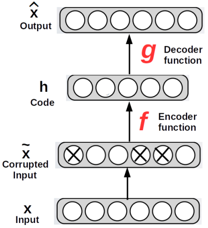
- First add some noise (e.g., Gaussian noise) to the original input $\boldsymbol{x}$
- Let's denote $\tilde{\boldsymbol{x}}$ as the corrupted version of $\boldsymbol{x}$
- The encoder f operates on $\tilde{\boldsymbol{x}}$, i.e., $\boldsymbol{h} = f(\tilde{\boldsymbol{x}})$



Denoising Autoencoder

Denoising Autoencoders

- First add some noise (e.g., Gaussian noise) to the original input $\boldsymbol{x}$
- Let's denote $\tilde{\boldsymbol{x}}$ as the corrupted version of $\boldsymbol{x}$
- The encoder f operates on $\tilde{\boldsymbol{x}}$, i.e., $\boldsymbol{h} = f(\tilde{\boldsymbol{x}})$

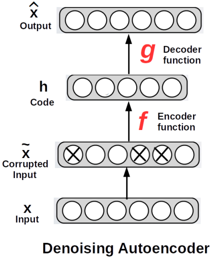


Denoising Autoencoder

- However, we still want to reconstruction $\hat{\boldsymbol{x}}$ to be close to the original uncorrupted input $\boldsymbol{x}$

Denoising Autoencoders

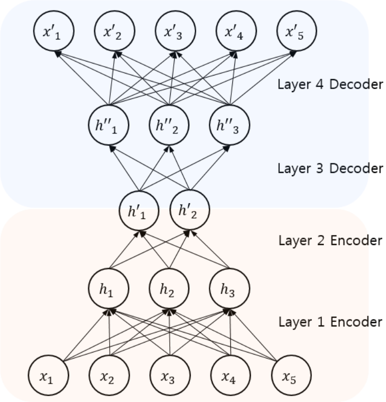
- First add some noise (e.g., Gaussian noise) to the original input $\boldsymbol{x}$
- Let's denote $\tilde{\boldsymbol{x}}$ as the corrupted version of $\boldsymbol{x}$
- The encoder f operates on $\tilde{\boldsymbol{x}}$, i.e., $\boldsymbol{h} = f(\tilde{\boldsymbol{x}})$



- However, we still want to reconstruction $\hat{\boldsymbol{x}}$ to be close to the original uncorrupted input $\boldsymbol{x}$
- Since the corruption is stochastic, we minimize the expected loss: $\mathbb{E}_{\tilde{\boldsymbol{x}} \sim p(\tilde{\boldsymbol{x}}|\boldsymbol{x})}[\ell(\hat{\boldsymbol{x}}, \tilde{\boldsymbol{x}})] + \Omega(\boldsymbol{h})$

Deep/Stacked Autoencoders

- Most autoencoders can be extended to have more than one hidden layer



Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings

Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings
- This gives a probabilistic approach for designing autoencoders

Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings
- This gives a probabilistic approach for designing autoencoders
- Negative log-likelihood $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ is equivalent to the reconstruction error

Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings
- This gives a probabilistic approach for designing autoencoders
- Negative log-likelihood $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ is equivalent to the reconstruction error
- Can also use a prior distribution $p(\boldsymbol{h})$ on the encodings (equivalent to regularizer)

Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings
- This gives a probabilistic approach for designing autoencoders
- Negative log-likelihood $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ is equivalent to the reconstruction error
- Can also use a prior distribution $p(\boldsymbol{h})$ on the encodings (equivalent to regularizer)
- Such ideas have been used to design generative models for autoencoders

Stochastic Autoencoders

- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings
- This gives a probabilistic approach for designing autoencoders
- Negative log-likelihood $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ is equivalent to the reconstruction error
- Can also use a prior distribution $p(\boldsymbol{h})$ on the encodings (equivalent to regularizer)
- Such ideas have been used to design generative models for autoencoders
  - Variational Autoencoder (VAE) is a popular example of such a model

Stochastic Autoencoders

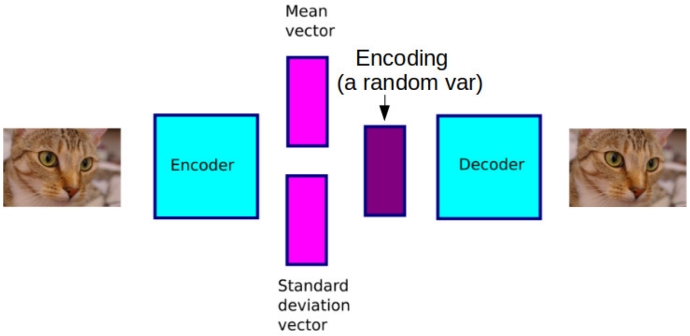
- Can also define the encoder and decoder functions using probability distributions

$$p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$$
$$p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$$

- The choice of distributions depends on the type of data being modeled and of the encodings
- This gives a probabilistic approach for designing autoencoders
- Negative log-likelihood $-\log p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$ is equivalent to the reconstruction error
- Can also use a prior distribution $p(\boldsymbol{h})$ on the encodings (equivalent to regularizer)
- Such ideas have been used to design generative models for autoencoders
  - Variational Autoencoder (VAE) is a popular example of such a model
  - Generative models like VAE can be used to "generate" new data using a random code $\boldsymbol{h}$

Variational Autoencoders (VAE)

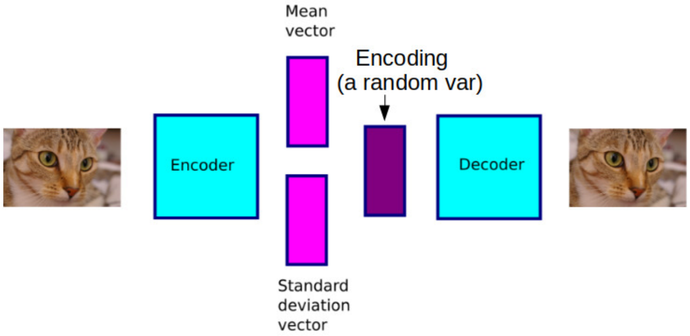
- Learns a distribution (e.g., a Gaussian) on the encoding[1]



[1] http://www.birving.com/presentations/autoencoders/

Variational Autoencoders (VAE)

- Learns a distribution (e.g., a Gaussian) on the encoding[1]



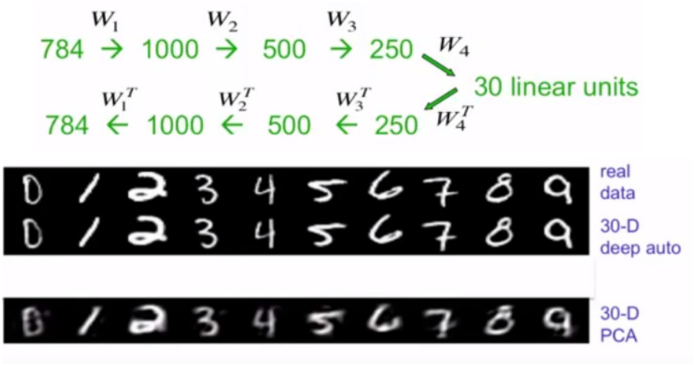
- Unlike standard AE, a VAE model learns to generate plausible data from random encodings

[1] http://www.birving.com/presentations/autoencoders/

Some Applications of Autoencoders

- (Unsupervised) Feature learning and Dimensionality reduction
- Denoising and inpainting
- Pre-training of deep neural networks
- Recommender systems applications

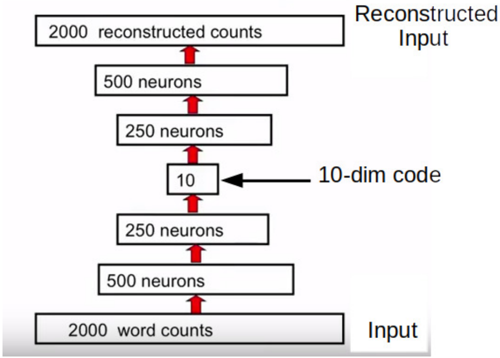
- Example: A deep AE for low-dim feature learning for 784-dimensional MNIST images[2]



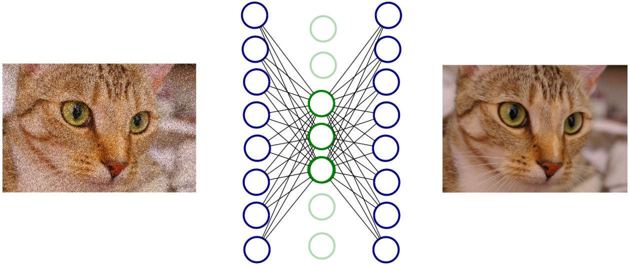
[2] Figure credit: Hinton and Salakhutdinov

Feature learning and Dimensionality Reduction

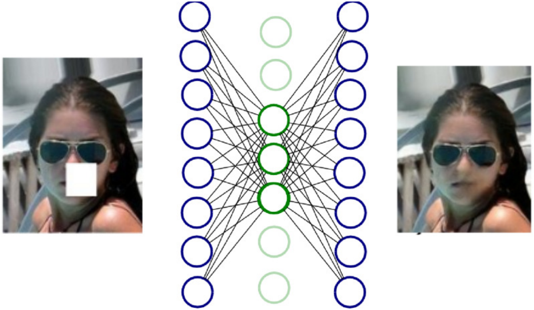
- Example: Low-dim feature learning for 2000-dimensional bag-of-words documents



Denoising and Inpainting



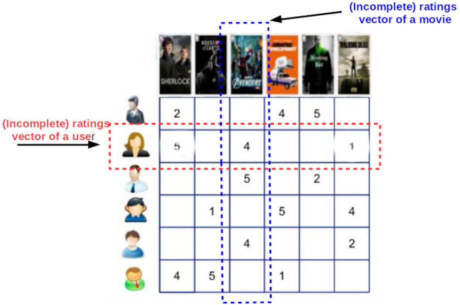
Denoising and Inpainting



Applications in Recommender Systems

Recommender Systems

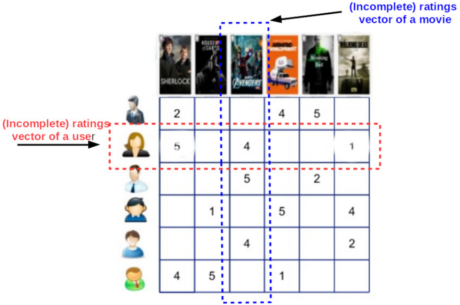
- Assume we are given a partially known $N \times M$ ratings matrix $\mathbf{R}$ of N users on M items (movies)



- Denote by $\boldsymbol{r}^{(u)}$ the (partially known) $M \times 1$ ratings vector of user u
- Denote by $\boldsymbol{r}^{(i)}$ the (partially known) $N \times 1$ ratings vector of item i

Recommender Systems

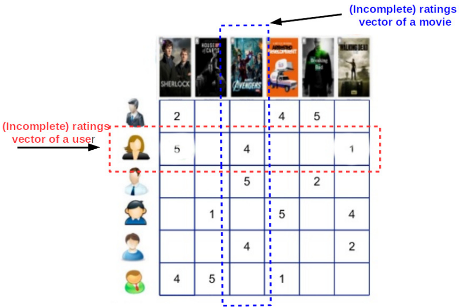
- Assume we are given a partially known $N \times M$ ratings matrix $\mathbf{R}$ of N users on M items (movies)



- Denote by $\boldsymbol{r}^{(u)}$ the (partially known) $M \times 1$ ratings vector of user u
- Denote by $\boldsymbol{r}^{(i)}$ the (partially known) $N \times 1$ ratings vector of item i
- How can we use this data to build a recommender system?

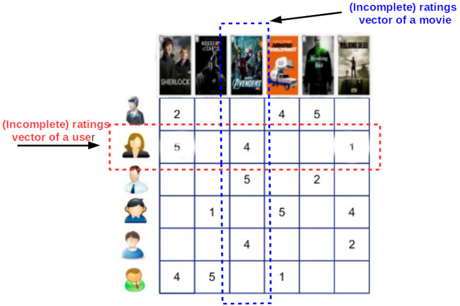
Recommender Systems via Matrix Completion

- An idea: If the predicted value of a user's rating for a movie is high, then we should ideally recommend this movie to the user



Recommender Systems via Matrix Completion

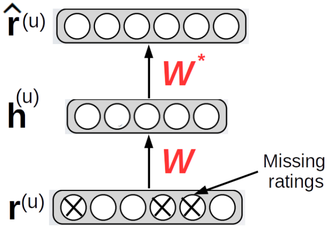
- An idea: If the predicted value of a user's rating for a movie is high, then we should ideally recommend this movie to the user



- Thus if we can "reconstruct" the missing entries in $\mathbf{R}$, we can use this method to recommend movies to users. Using an autoencoders can help us do this!

An Autoencoder based Approach

- Using the rating vectors $\{\boldsymbol{r}^{(u)}\}_{u=1}^{N}$ of all users, can learn an autoencoder



- Note: During backprop, only update weights in $\mathbf{W}$ that are connected to the observed ratings[3]
- Once learned, the model can predict (reconstruct) the missing ratings

[3] AutoRec: Autoencoders Meet Collaborative Filtering (Sedhain et al, WWW 2015)

Another Autoencoder based Approach

- Another approach is to combine (denoising) autoencoders with a matrix factorization model[4]

[4] Deep Collaborative Filtering via Marginalized Denoising Auto-encoder (Li et al, CIKM 2015)

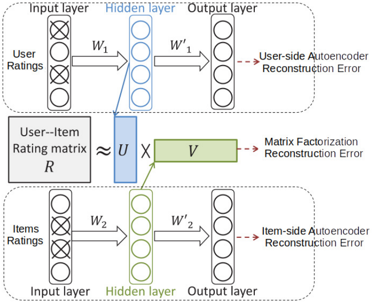
Another Autoencoder based Approach

- Another approach is to combine (denoising) autoencoders with a matrix factorization model[4]
- Idea: Rating of a user u on an item i can be defined using the inner-product based similarity of their features learned via an autoencoder: $R_{ui} = f(\boldsymbol{h}^{(u)^\top}\boldsymbol{h}^{(i)})$ where f is some compatibity function

[4] Deep Collaborative Filtering via Marginalized Denoising Auto-encoder (Li et al, CIKM 2015)

Another Autoencoder based Approach

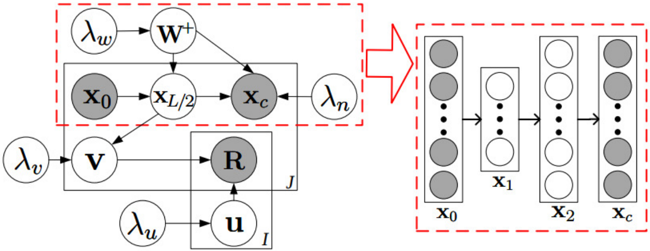
- Another approach is to combine (denoising) autoencoders with a matrix factorization model[4]
- Idea: Rating of a user u on an item i can be defined using the inner-product based similarity of their features learned via an autoencoder: $R_{ui} = f(\boldsymbol{h}^{(u)\top}\boldsymbol{h}^{(i)})$ where f is some compatibity function
- Denoting $\{\boldsymbol{h}^{(u)}\}_{u=1}^{N} = \mathbf{U}$ and $\{\boldsymbol{h}^{(i)}\}_{i=1}^{M} = \mathbf{V}$, we can write $\mathbf{R} = \mathbf{U}\mathbf{V}^\top$



[4]Deep Collaborative Filtering via Marginalized Denoising Auto-encoder (Li et al, CIKM 2015)

Other Approaches on Autoencoders for Recommender Systems

- Several recent papers on similar autoencoder based ideas
  - Collaborative Denoising Auto-Encoders for Top-N Recommender Systems (Wu et al, WSDM 2016)
  - Collaborative Deep Learning for Recommender Systems (Wang et al, KDD 2015)



  - Also possible to incorporate side information about the users and/or items (Wang et al, KDD 2015)

Autoencoders: Summary

- Simple and powerful for (nonlinear) feature learning
- Learned features are able to capture salient properties of data
- Several extensions (sparse, denoising, stochastic, etc.)
- Can also be stacked to create "deep" autoencoders
- Recent focus on autoencoders that are based on generative models of data
  - Example: Variational Autoencoders