DSC 140B

Representation Learning

Lecture 24 | Part 1

Autoencoders

Generalizing PCA

- We started the quarter with PCA.
- PCA is a **linear** method.
- We can generalize upon PCA to derive nonlinear representation learners.

Representation Learning

- At a high level, representation learning finds an **encoding function** $\text{encode}(\vec{x}) : \mathbb{R}^d \to \mathbb{R}^k$.
- Ideally, this function captures useful aspects of the data distribution.

Example: PCA

- In PCA, we **encode** a point $\vec{x}$ by projecting it onto the top k eigenvectors of data covariance matrix:

$$\text{encode}(\vec{x}) = U^T \vec{x}$$

Decoding

- Encoding can decrease dimensionality.
- Intuitively, we may want to preserve as much "information" about $\vec{x}$ as possible.
- We should be able to **decode** the encoding and **reconstruct** the original point, approximately.

$$\vec{x} \approx \text{decode}(\text{encode}(\vec{x}))$$

Example: PCA

- In PCA, given a point $\vec{z} \in \mathbb{R}^k$ in the new representation, the **reconstruction** is:

$$\text{decode}(\vec{z}) = U\vec{z}$$

Representation Learning

- **Goal:** find an encoder (and decoder) such that

$$\text{encode}(\text{decode}(\vec{x})) \approx \vec{x}$$

Reconstruction Error

- In general, decode(encode($\vec{x}$)) will not be exactly equal to $\vec{x}$.

- One way of quantifying the difference w.r.t. data is the (ℓ_2) **reconstruction error**:

$$\sum_{i=1}^{n} \|\vec{x}^{(i)} - \text{decode}(\text{encode}(\vec{x}^{(i)}))\|^2$$

Note

- Of course, it is trivial to find an encoder/decoder with zero reconstruction error:

$$\text{encode}(\vec{x}) = \vec{x} = \text{decode}(\vec{x})$$

- Such an encoder is not useful.

- Instead, we constrain the form of the encoder so that it cannot simply copy the input.

Example: PCA

- Assume encode($\vec{x}$) = $U\vec{x}$, for some matrix U whose $k \leq d$ columns are orthonormal.
    - That is, the encoding is an orthogonal projection.

- **Goal:** find U to minimize reconstruction error on a dataset $\vec{x}^{(1)}, \ldots, \vec{x}^{(d)}$.

- **Solution:** pick columns of U to be top k eigenvectors of data covariance matrix.

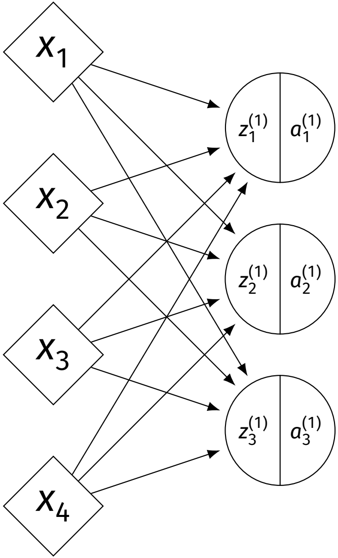
Now

- $\text{encode}(\vec{x}) = U\vec{x}$ is a linear encoding function.
- What if we let encode be nonlinear?
- That is, let's generalize PCA.

Encoder as a Neural Network

- Assume encode($\vec{x}$) is a (deep) **neural network**.
- Output is not a single number, but k numbers.
  - I.e., a vector in $\mathbb{R}^k$
- Can use nonlinear activations, have more than one layer.

Encoder as a Neural Network



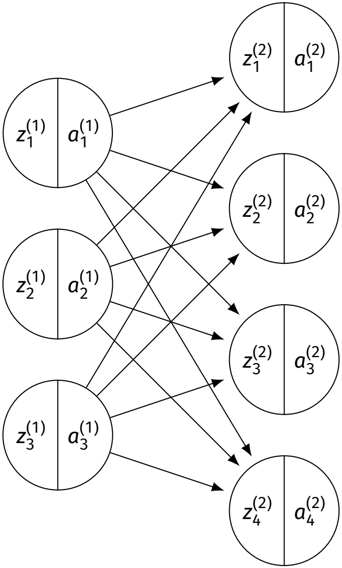
Encoder as a Neural Network

- The output of the encoder is the new representation.
- To train the encoder, we’ll need a **decoder**.

Decoder as a Neural Network

- Assume decode($\vec{z}$) is a (deep) **neural network**.
- Output is not a single number, but d numbers.
  - Same dimensionality as original input, $\vec{x}$.
  - I.e., a vector in $\mathbb{R}^d$
- Can use nonlinear activations, have more than one layer.

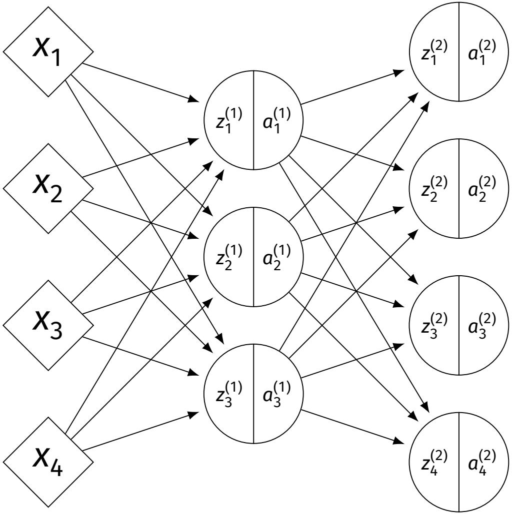
Decoder as a Neural Network



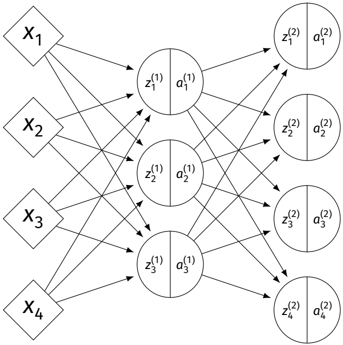
decode(encode($\vec{x}$)) **as a NN**

- Together, decode(encode($\vec{x}$)) is a neural network $H(\vec{x}) : \mathbb{R}^d \to \mathbb{R}^d$.

decode(encode($\vec{x}$)) **as a NN**



Training



- We want $H(\vec{x}) \approx \vec{x}$
- One approach: train network to minimize reconstruction error.

$$\sum_{i=1}^{n} \|\vec{x}^{(i)} - H(\vec{x}^{(i)})\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{d} (\vec{x}_j^{(i)} - (H(\vec{x}^{(i)}))_j)^2$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{d} (\vec{x}_j^{(i)} - a_j^{(2)}(\vec{x}^{(i)}))^2$$

Training

- The network can be trained using gradient-based methods.
  - E.g., stochastic gradient descent.
- **Note:** this is an **unsupervised** learning problem.

Autoencoders

- When the encoder/decoder are NNs, $H(\vec{x}) = \text{decode}(\text{encode}(\vec{x}))$ is an **autoencoder**.

Generalizing PCA

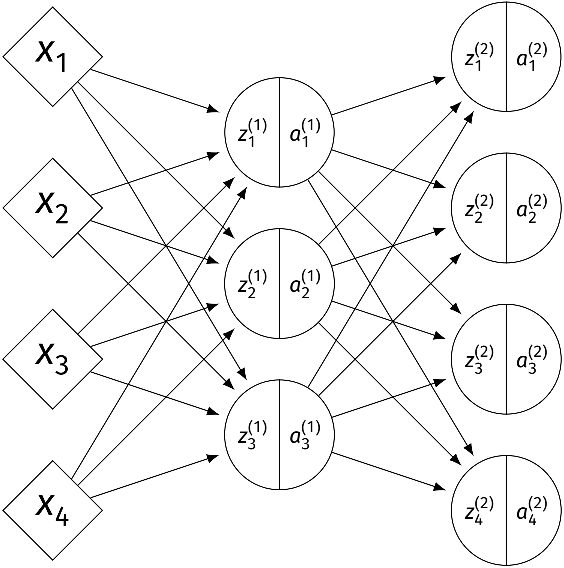
- We can view autoencoders as generalizations of PCA.

- Consider again the encoder that performs an orthogonal projection:

$$\text{encode}(\vec{x}) = U^T\vec{x}$$

$$\text{decode}(\vec{z}) = U\vec{z}$$

- encode/decode are neural networks (with linear activations).


x_1
x_2
x_3
x_4
$z_1^{(1)}$ $a_1^{(1)}$
$z_2^{(1)}$ $a_2^{(1)}$
$z_3^{(1)}$ $a_3^{(1)}$
$z_1^{(2)}$ $a_1^{(2)}$
$z_2^{(2)}$ $a_2^{(2)}$
$z_3^{(2)}$ $a_3^{(2)}$
$z_4^{(2)}$ $a_4^{(2)}$

Exercise

True/False: training an autoencoder to minimize reconstruction error will result in the same encode($\vec{x}$) function as PCA.

Answer: False

- PCA minimizes reconstruction error **subject to** the constraint that the columns of U are orthonormal.
- Without the orthonormality constraint, the autoencoder learns a different encoding.
- *However*, the autoencoder learns a (non-orthogonal) projection into the same space as PCA.

In other words...

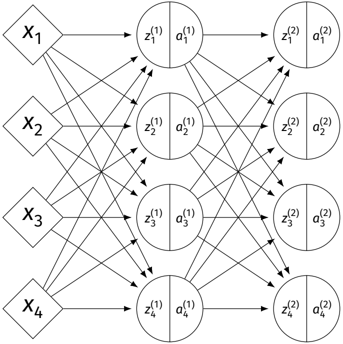
- PCA is an autoencoder trained with an additional orthonormality constraint.
- Cannot easily be learned by gradient descent; find eigenvectors instead.

Uses of Autoencoders

- Like PCA, autoencoders can be used for **dimensionality reduction**.
- Unlike PCA, autoencoders can learn nonlinear maps.
- Encoded data can be used as input to predictive model, etc.

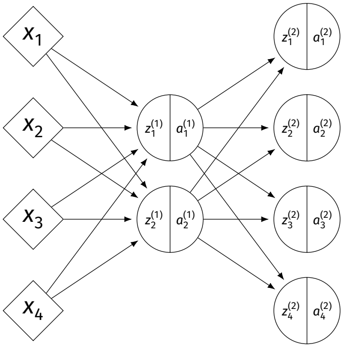
Dimensionality Reduction

- If the dimensionality of the encoder is the same as the dimensionality of $\vec{x}$, the autoencoder can learn to simply reproduce the input.



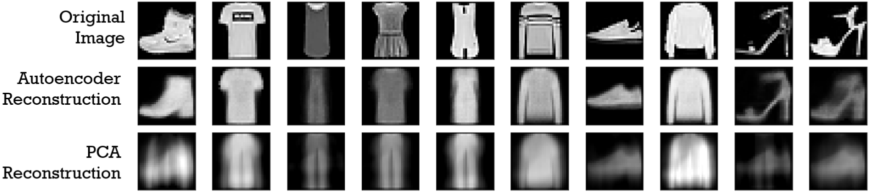
Dimensionality Reduction

- As such, we choose number of hidden nodes < d.



- Called an **undercomplete autoencoder**.

Example

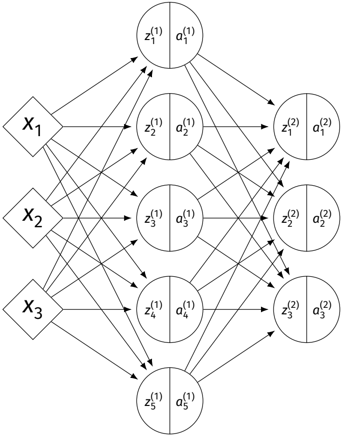


[1]

[1] By Michela Massi - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=80176900

Other Uses

- However, sometimes it is useful for hidden layer to have **greater** dimensionality.



Denoising Autoencoders

- One such case is in **denoising autoencoders**.
- **Idea:** train an autoencoder to remove noise.
- Add random noise to each $\vec{x}^{(i)}$ to get $\tilde{x}^{(i)}$.
- Train network so that $H(\tilde{x}^{(i)}) \approx \vec{x}$.

DSC 140B

Representation Learning

Lecture 24 | Part 2

Conclusion of DSC 140B

Recap

- DSC 140B was about **representation learning**.
- We saw PCA, Laplacian Eigenmaps, RBF Networks, neural networks and deep learning
- Learned ML methods, but also theoretical tools for understanding why other ML methods work

More Deep Learning

- We have only scratched the surface of deep learning.
  - LSTMs, transformer models, graph neural networks, deep RL, GANs, etc.

- In this class, we focused on the fundamental principles behind NNs.

- You might consider taking CSE 151B.

Thanks!