# LMP 1210H: Basic Principles of Machine Learning in Biomedical Research

Bo Wang
AI Lead Scientist, PMCC, UHN
CIFAR AI Chair, Vector Institute
Assistant Professor, University of Toronto

## Administrative Stuff

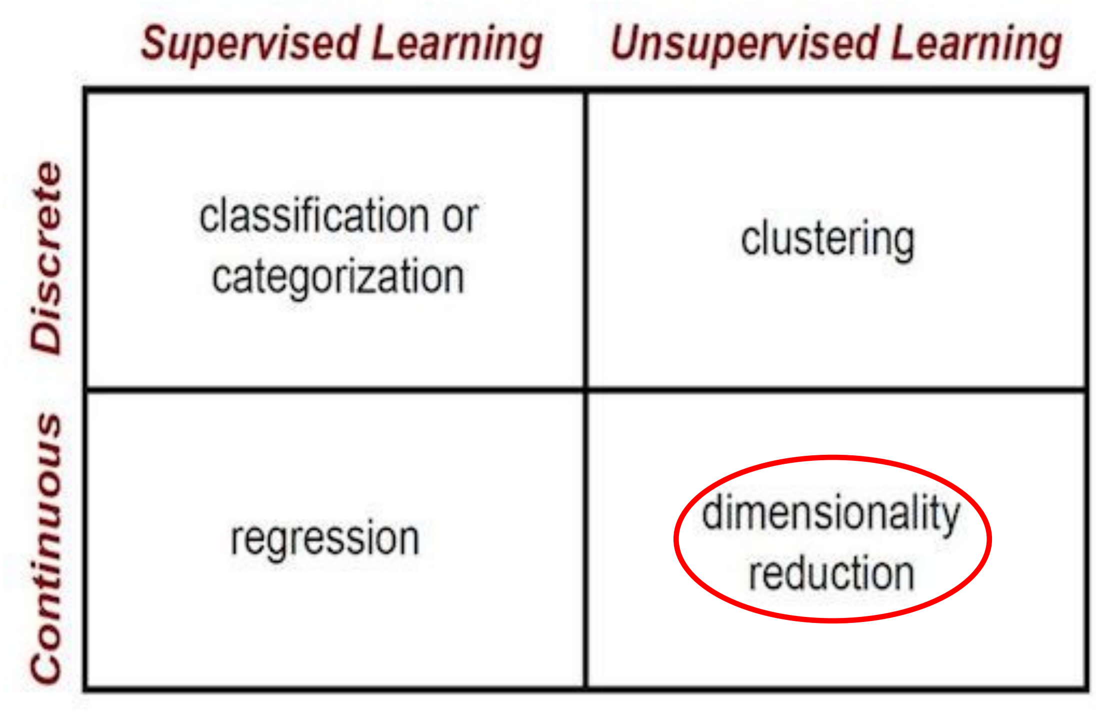
1. Marks of Homework 1 is released!
2. Homework 3 : March 19
3. Schedules of in-class presentations will be released soon.
4. Interesting talks next coming weeks!

## More on the participation marks

In-class participation (**15%**)

Q&A during lectures

# Unsupervised Learning vs Supervised Learning



| | Supervised Learning | Unsupervised Learning |
|---|---|---|
| Discrete | classification or categorization | clustering |
| Continuous | regression | dimensionality reduction |

## Why dimension reduction?

**High-dimensional data is everywhere!**

- High-Dimensions = Lot of Features

  Document classification

  Features per document = thousands of words/unigrams

Source: Nina Balcan

## Why dimension reduction?

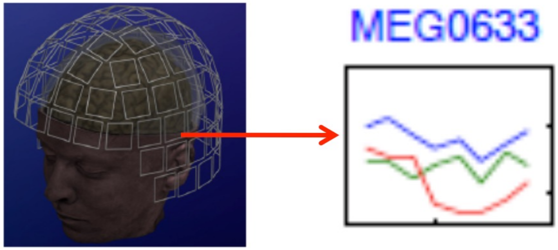
**High-dimensional data is everywhere!**

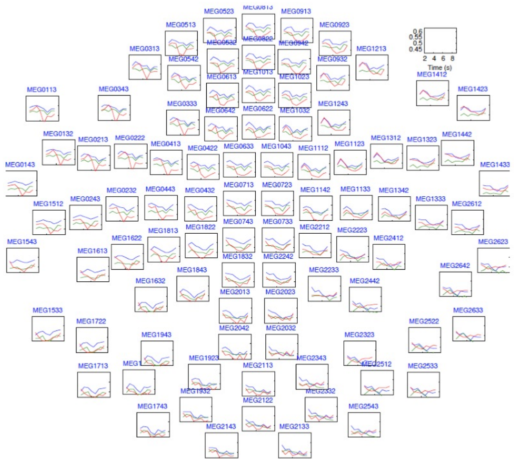
- High-Dimensions = Lot of Features

MEG Brain Imaging

120 locations x 500 time points

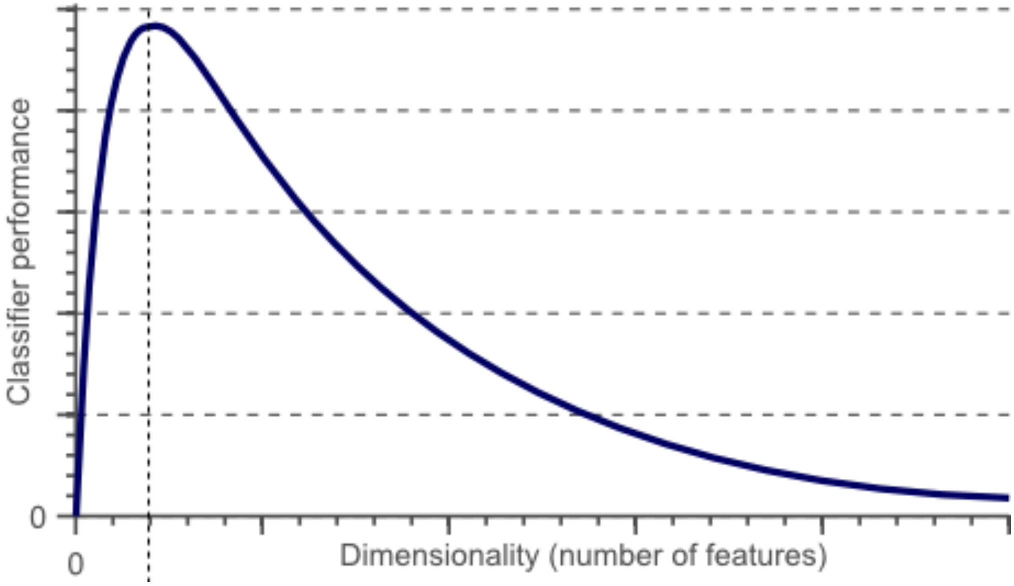
x 20 objects





Source: Nina Balcan

## Why dimension reduction?



**Curse of Dimensionality**

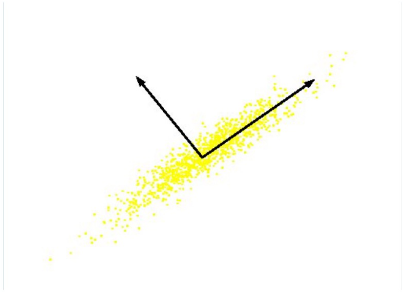
## Why dimension reduction?

**Useful for:**

- Visualization
- More efficient use of resources (e.g., time, memory, communication)
- Statistical: fewer dimensions → better generalization
- Noise removal (improving data quality)
- Further processing by machine learning algorithms

## PCA: Principle Component Analysis

**What is PCA:** Unsupervised technique for extracting variance structure from high dimensional datasets.

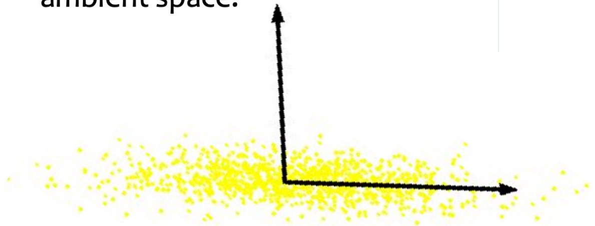


- PCA is an orthogonal projection or transformation of the data into a (possibly lower dimensional) subspace so that the variance of the projected data is maximized.

Source: Nina Balcan

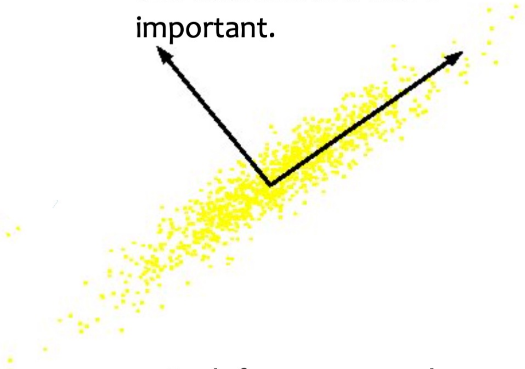
## PCA: Principle Component Analysis

Intrinsically lower dimensional than the dimension of the ambient space.



Only one relevant feature

If we rotate data, again only one coordinate is more important.

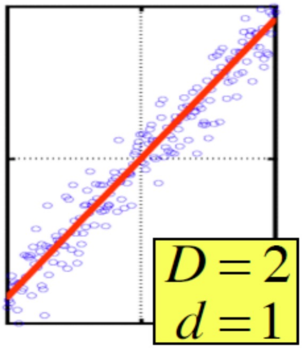


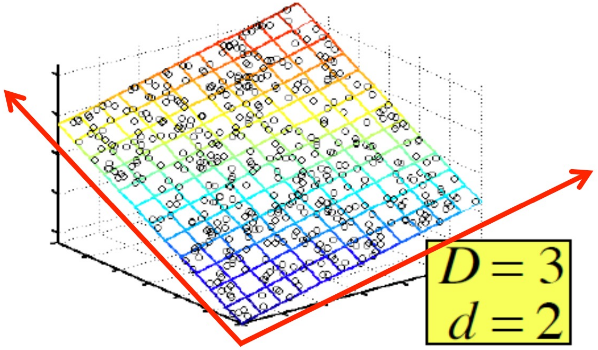
Both features are relevant

Question: Can we transform the features so that we only need to preserve one latent feature?

Source: Nina Balcan

# PCA:  Principle Component Analysis



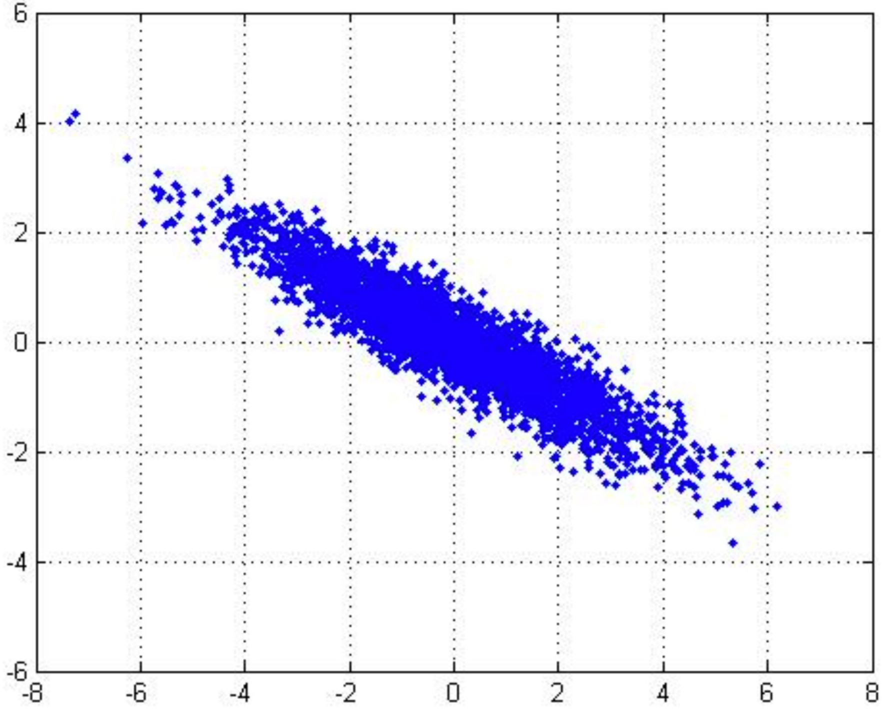


In case where data lies on or near a low d-dimensional linear subspace, axes of this subspace are an effective representation of the data.

Identifying the axes is known as Principal Components Analysis, and can be obtained by using classic matrix computation tools (Eigen or Singular Value Decomposition).

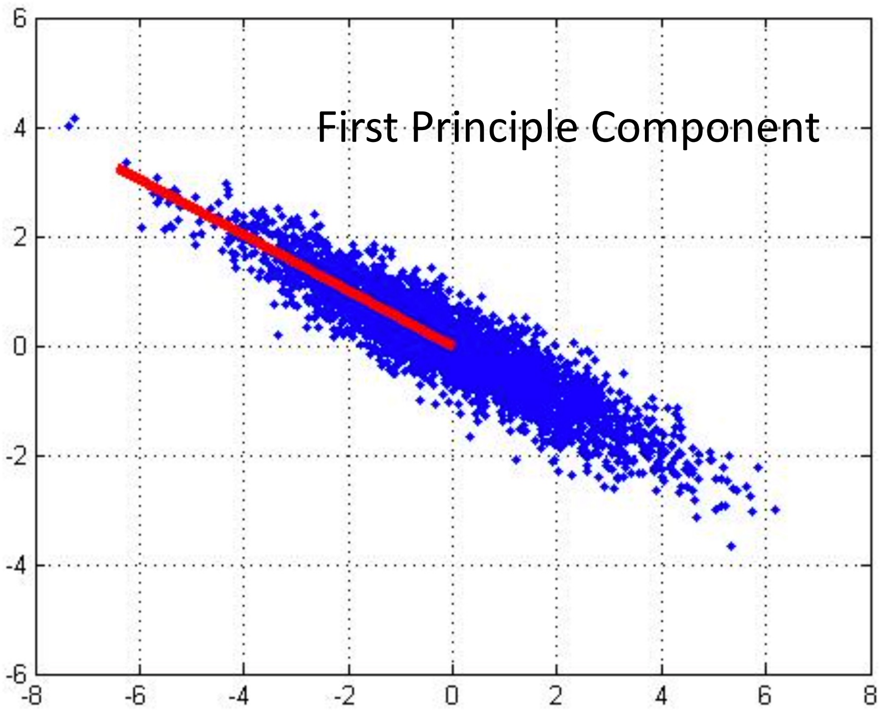
Source: Nina Balcan

## Example: 2D Gaussian



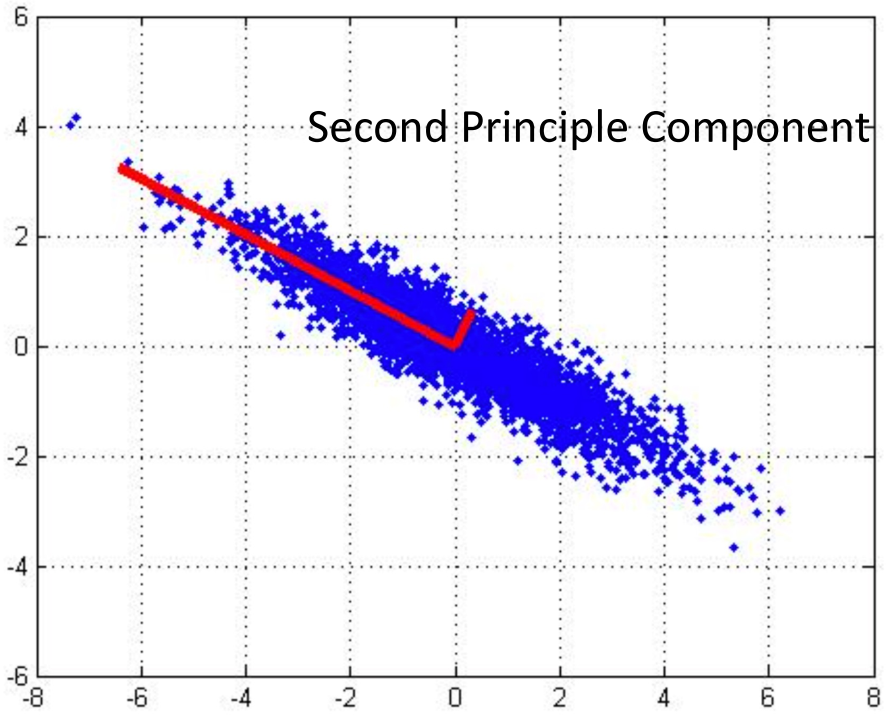
Source: Barnabas Poczos

## Example: 2D Gaussian



Source: Barnabas Poczos

## Example: 2D Gaussian



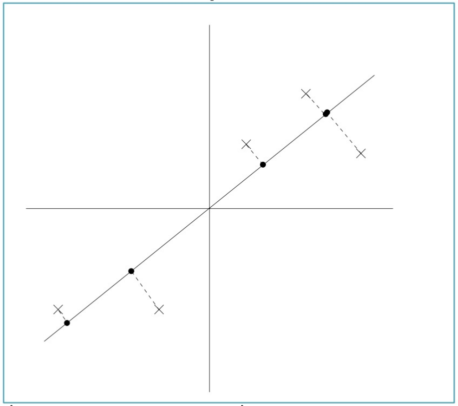
Source: Barnabas Poczos

## How PCA?

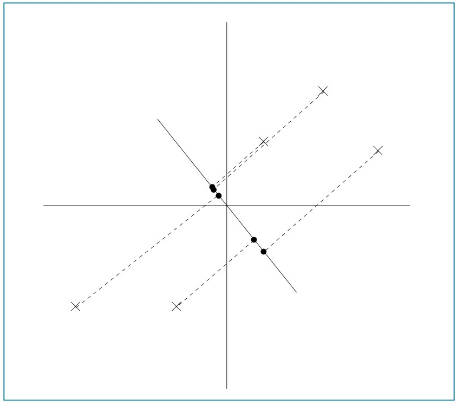
**Intuition: Maximizing the Variances**

- Consider the two projections below
- Which maximizes the variance?

Option A



Option B



Source: Andrew Ng (CS229 Lecture Notes)

## How PCA?

$\boxed{\text{Intuition: Maximizing the Variances}}$

Maximise $\mathbf{u}^T\mathbf{X}\mathbf{X}^T\mathbf{u}$

s.t $\mathbf{u}^T\mathbf{u} = 1$ $\rightarrow$ $\boxed{\text{First PC}}$

Construct Langrangian $\mathbf{u}^T\mathbf{X}\mathbf{X}^T\mathbf{u} - \lambda\mathbf{u}^T\mathbf{u}$

Vector of partial derivatives set to zero

$$\mathbf{x}\mathbf{x}^T\mathbf{u} - \lambda\mathbf{u} = (\mathbf{x}\mathbf{x}^T - \lambda\mathbf{I})\,\mathbf{u} = 0$$

As $\mathbf{u} \neq \mathbf{0}$ then $\mathbf{u}$ must be an eigenvector of $\mathbf{X}\mathbf{X}^T$ with eigenvalue $\lambda$

Source: Barnabas Poczos

## How PCA?

Intuition: Maximizing the Variances

- Given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, compute covariance matrix $\Sigma$

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T$$

where

$$\bar{\mathbf{x}} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_i$$

- **PCA** basis vectors = the eigenvectors of $\Sigma$

We get the eigvectors using an eigendecomposition.
Power iteration (Von Mises iteration is a standard algorithm for this)

- Larger eigenvalue $\Rightarrow$ more important eigenvectors

Source: Barnabas Poczos

## More on Co-Variance

- Covariance as a measure of how much each of the dimensions vary from the mean with respect to each other.

- What is the interpretation of covariance calculations?

  e.g.: 2 dimensional data set

  x: number of hours studied for a subject

  y: marks obtained in that subject

  covariance value is say: 104.53

  what does this value mean?

source: Robert Collins, CSE/EE586

## More on Co-Variance

- Covariance as a measure of how much each of the dimensions vary from the mean with respect to each other.
- A positive value of covariance indicates both dimensions increase or decrease together e.g. as the number of hours studied increases, the marks in that subject increase.
- A negative value indicates while one increases the other decreases, or vice-versa
- If covariance is zero: the two dimensions are independent of each other e.g. heights of students vs the marks obtained in a subject

source: Robert Collins, CSE/EE586

## More on Co-Variance

- Covariance as a measure of how much each of the dimensions vary from the mean with respect to each other.

- A positive value of covariance indicates both dimensions increase or decrease together e.g. as the number of hours studied increases, the marks in that subject increase.

- A negative value indicates while one increases the other decreases, or vice-versa

- If covariance is zero: the two dimensions are independent of each other e.g. heights of students vs the marks obtained in a subject

source: Robert Collins, CSE/EE586

## More on Co-Variance

- By finding the eigenvalues and eigenvectors of the covariance matrix, we find that the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset.
- This is the principal component.
- PCA is a useful statistical technique that has found application in:
  - fields such as face recognition and image compression
  - finding patterns in data of high dimension.

source: Robert Collins, CSE/EE586

## PCA Summary

**Goal:** Find $r$-dim projection that best preserves variance

1. Compute mean vector $\mu$ and covariance matrix $\Sigma$ of original points
2. Compute eigenvectors and eigenvalues of $\Sigma$
3. Select top $r$ eigenvectors
4. Project points onto subspace spanned by them:

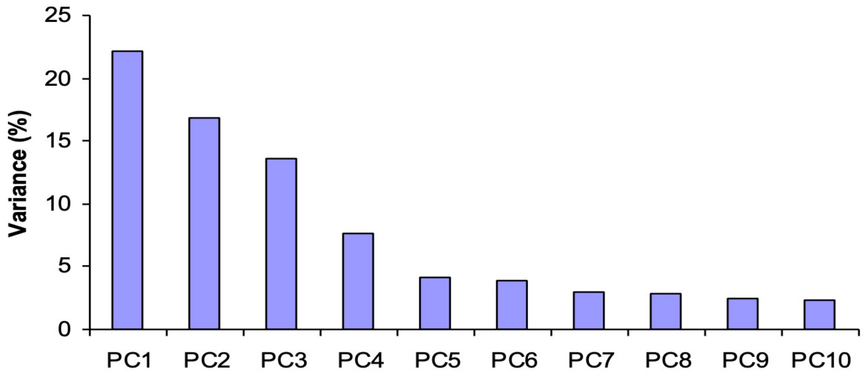
$$y = A(x - \mu)$$

where $y$ is the new point, $x$ is the old one, and the rows of $A$ are the eigenvectors

## How many PCs

- For n original dimensions, sample covariance matrix is nxn, and has up to n eigenvectors. So n PCs.
- Where does dimensionality reduction come from?

Can *ignore* the components of lesser significance.



You do lose some information, but if the eigenvalues are small, you don't lose much

- n dimensions in original data
- calculate n eigenvectors and eigenvalues
- choose only the first p eigenvectors, based on their eigenvalues

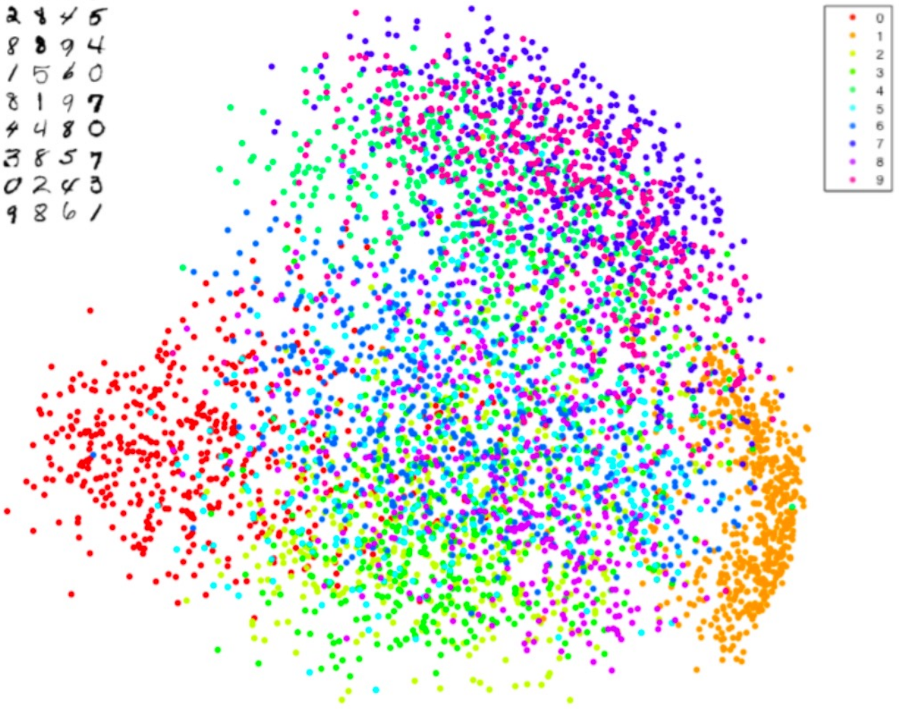
## Breaks

## Visualization of data using t-SNE

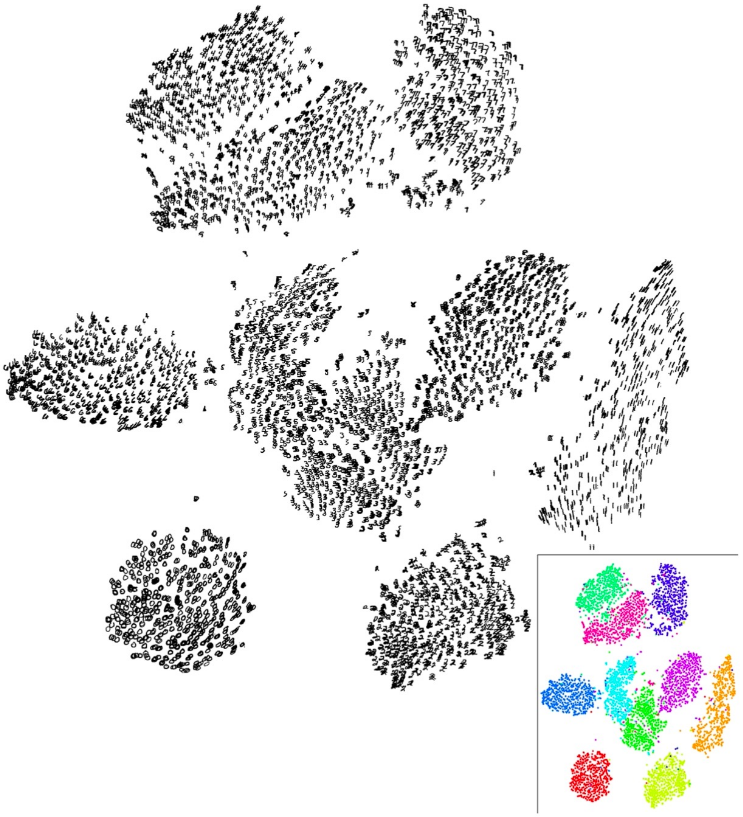
SNE - Stochastic Neighbor Embedding

- t-SNE is an alternative dimensionality reduction algorithm.
- PCA tries to find a global structure
  - Low dimensional subspace
  - Can lead to local inconsistencies
    - Far away point can become nearest neighbors
- t-SNE tries to perserve local structure
  - Low dimensional neighborhood should be the same as original neighborhood.
- Unlike PCA almost only used for visualization
  - No easy way to embed new points

# Visualization of MNIST using PCA



## Visualization of MNIST using t-SNE



## SNE -- Stochastic Neighboring Embedding

SNE basic idea:

- "Encode" high dimensional neighborhood information as a distribution
- Intuition: Random walk between data points.
  - High probability to jump to a close point
- Find low dimensional points such that their neighborhood distribution is similar.
- How do you measure distance between distributions?
  - Most common measure: KL divergence

## SNE -- Stochastic Neighboring Embedding

- Consider the neighborhood around an input data point $\mathbf{x}_i \in \mathbb{R}^d$
- Imagine that we have a Gaussian distribution centered around $\mathbf{x}_i$
- Then the probability that $\mathbf{x}_i$ chooses some other datapoint $\mathbf{x}_j$ as its neighbor is in proportion with the density under this Gaussian
- A point closer to $\mathbf{x}_i$ will be more likely than one further away

## SNE -- Stochastic Neighboring Embedding

The $i \rightarrow j$ probability (should be familiar from A1Q2), is the probability that point $\mathbf{x}_i$ chooses $\mathbf{x}_j$ as its neighbor

$$P_{j|i} = \frac{\exp\left(-||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2/2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-||\mathbf{x}^{(i)} - \mathbf{x}^{(k)}||^2/2\sigma_i^2\right)}$$

With $P_{i|i} = 0$

- The parameter $\sigma_i$ sets the size of the neighborhood
    - Very low $\sigma_i$ - all the probability is in the nearest neighbor.
    - Very high $\sigma_i$ - Uniform weights.
- Here we set $\sigma_i$ differently for each data point
- Results depend heavily on $\sigma_i$ - it defines the neighborhoods we are trying to preserve.
- Final distribution over pairs is symmetrized: $P_{ij} = \frac{1}{2N}(P_{i|j} + P_{j|i})$

## SNE -- Stochastic Neighboring Embedding

Given $\mathbf{x}^{(1)}, .., \mathbf{x}^{(N)} \in \mathbb{R}^D$ we define the distribution $P_{ij}$

Goal: Find good embedding $\mathbf{y}^{(1)}, .., \mathbf{y}^{(N)} \in \mathbb{R}^d$ for some $d < D$ (normally
or 3)

How do we measure an embedding quality?

For points $\mathbf{y}^{(1)}, .., \mathbf{y}^{(N)} \in \mathbb{R}^d$ we can define distribution $Q$ similarly the sa
(notice no $\sigma_i^2$ and not symmetric)

$$Q_{ij} = \frac{\exp\left(-||\mathbf{y}^{(i)} - \mathbf{y}^{(j)}||^2\right)}{\sum_k \sum_{l \neq k} \exp\left(-||\mathbf{y}^{(l)} - \mathbf{y}^{(k)}||^2\right)}$$

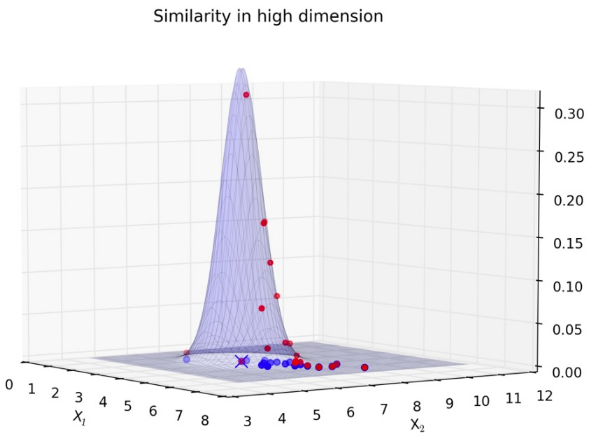
Optimize $Q$ to be close to $P$

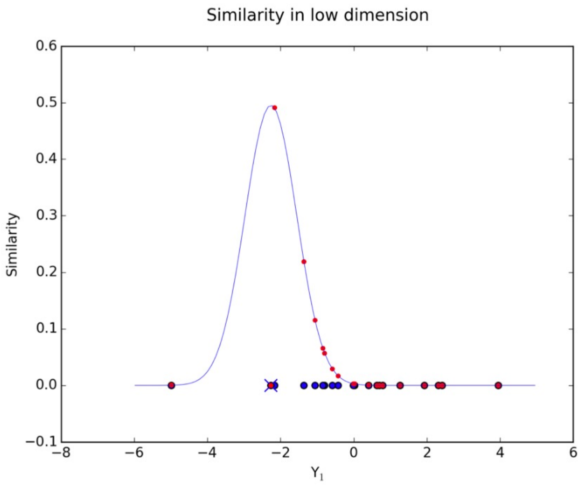
- Minimize KL-divergence

## SNE pitfall: Crowding problem

- In high dimension we have more room, points can have a lot of different neighbors
- In 2D a point can have a few neighbors at distance one all far from each other - what happens when we embed in 1D?
- This is the "crowding problem" - we don't have enough room to accommodate all neighbors.
- This is one of the biggest problems with SNE.
- t-SNE solution: Change the Gaussian in $Q$ to a heavy tailed distribution.
  - if $Q$ changes slower, we have more "wiggle room" to place points at.

## SNE pitfall: Crowding problem





There is much more space in high dimensions.

## t-Distributed SNE

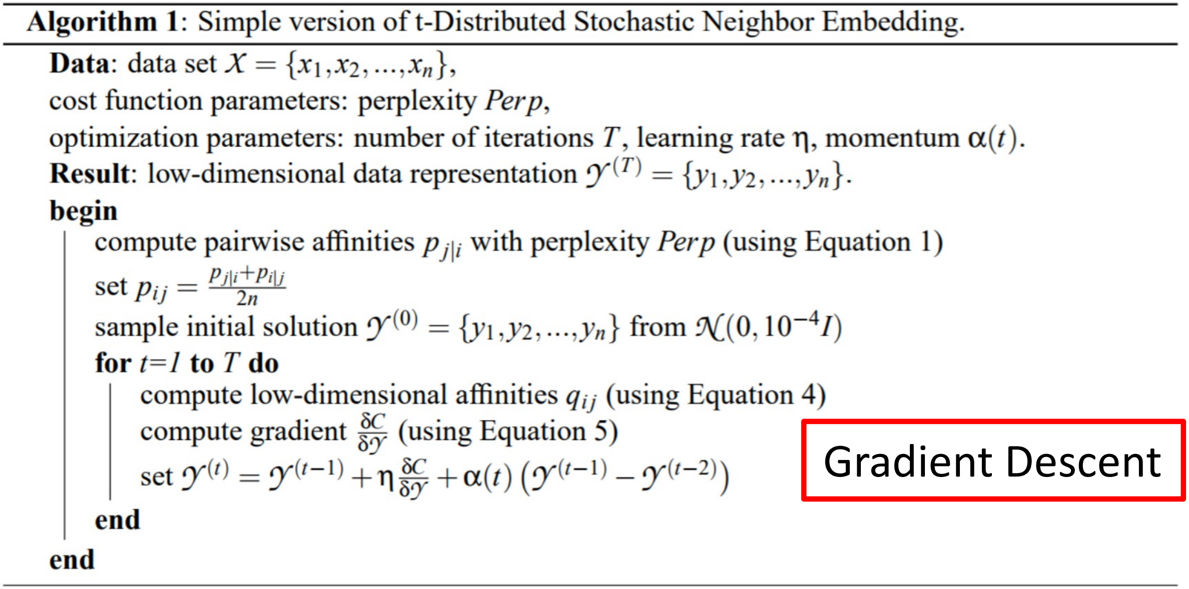
t-Distributed Stochastic Neighbor Embedding

- Student-t Probability density $p(x) \propto (1 + \frac{x^2}{v})^{-(v+1)/2}$
  - for $v = 1$ we get $p(x) \propto \frac{1}{1+x^2}$
- Probability goes to zero much slower then a Gaussian.
- Can show it is equivalent to averaging Gaussians with some prior over $\sigma^2$
- We can now redefine $Q_{ij}$ as

$$Q_{ij} = \frac{(1 + ||\mathbf{y}_i - \mathbf{y}_j||^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + ||\mathbf{y}_k - \mathbf{y}_l||^2)^{-1}}$$

- We leave $P_{ij}$ as is!

# How t-SNE? (Optional)



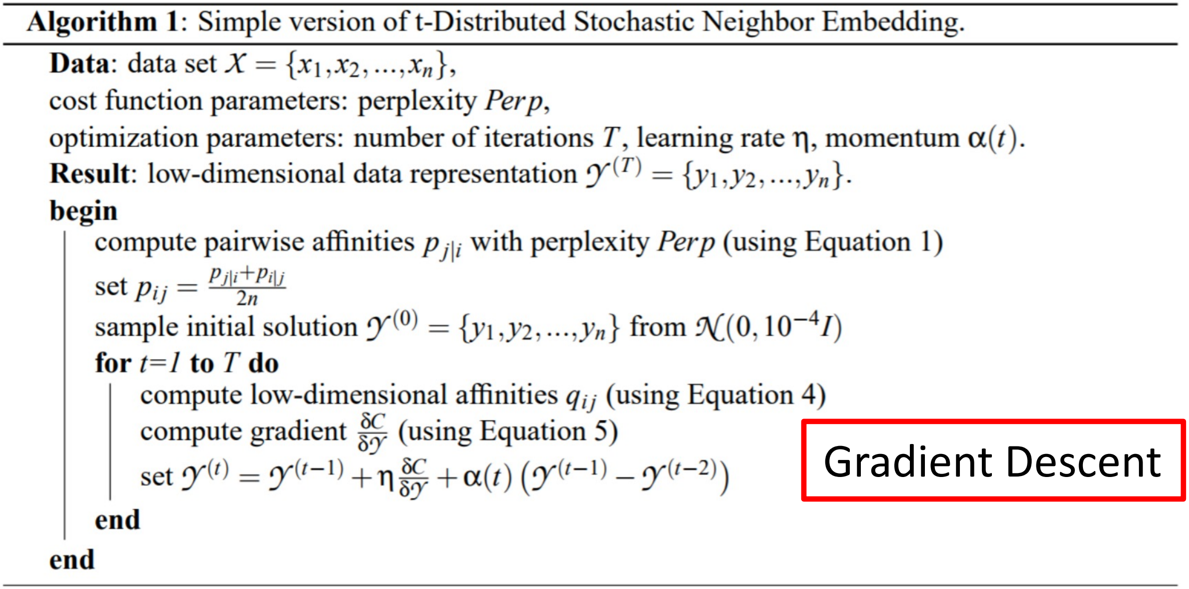
**Algorithm 1**: Simple version of t-Distributed Stochastic Neighbor Embedding.

```
Data: data set $X = \{x_1, x_2, \ldots, x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
begin
    compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
    set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
    sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
    for t=1 to T do
        compute low-dimensional affinities $q_{ij}$ (using Equation 4)
        compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
        set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t) \left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
    end
end
```

Gradient Descent

[Slide credit: "Visualizing Data using t-SNE"]

## How t-SNE? (Optional)



**Algorithm 1**: Simple version of t-Distributed Stochastic Neighbor Embedding.

```
Data: data set $X = \{x_1, x_2, \dots, x_n\}$,
cost function parameters: perplexity $Perp$,
optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$.
Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \dots, y_n\}$.
begin
    compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
    set $p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$
    sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \dots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
    for $t$=1 to $T$ do
        compute low-dimensional affinities $q_{ij}$ (using Equation 4)
        compute gradient $\frac{\delta C}{\delta \mathcal{Y}}$ (using Equation 5)
        set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$
    end
end
```

Gradient Descent

[Slide credit: "Visualizing Data using t-SNE"]

## PCA vs t-SNE

- PCA
  - Requires more than 2 dimensions
  - Thrown off by quantised data
  - Expects linear relationships

- tSNE
  - Can’t cope with noisy data
  - Loses the ability to cluster

## PCA vs t-SNE

- PCA
  - Requires more than 2 dimensions
  - Thrown off by quantised data
  - Expects linear relationships

- tSNE
  - Can't cope with noisy data
  - Loses the ability to cluster

**Answer: Combine the two methods, get the best of both worlds**

- PCA
  - Good at extracting signal from noise
  - Extracts informative dimensions

- tSNE
  - Can reduce to 2D well
  - Can cope with non-linear scaling

## So PCA + t-SNE ?

tSNE is slow.  This is probably it’s biggest crime

- tSNE doesn’t scale well to large numbers of cells (10k+)

tSNE only gives reliable information on the closest neighbours  large distance information is almost irrelevant

source: Simon Andrews

## UMAP: Uniform Manifold Approximation and Projection

- UMAP is a replacement for tSNE to fulfil the same role
- Conceptually very similar to tSNE, but with a couple of relevant (and somewhat technical) changes
- Practical outcome is:
  - UMAP is quite a bit quicker than tSNE
  - UMAP can preserve more global structure than tSNE*
  - UMAP can run on raw data without PCA preprocessing*
  - UMAP can allow new data to be added to an existing projection

* In theory, but possibly not in practice

source: Simon Andrews

## UMAP: Uniform Manifold Approximation and Projection

- Instead of the single perplexity value in tSNE, UMAP defines
  - **Nearest neighbours**: the number of expected nearest neighbours – basically the same concept as perplexity
  - **Minimum distance**: how tightly UMAP packs points which are close together
- Nearest neighbours will affect the influence given to global vs local information. Min dist will affect how compactly packed the local parts of the plot are.

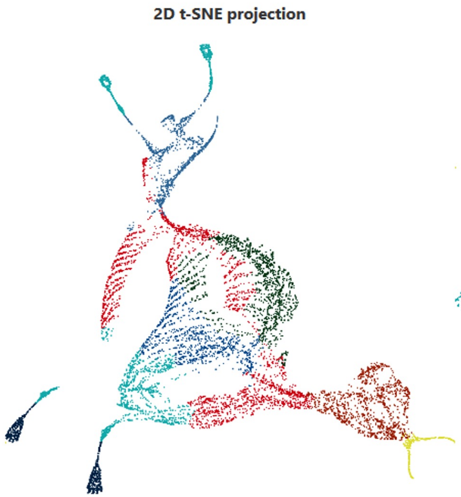
source: Simon Andrews

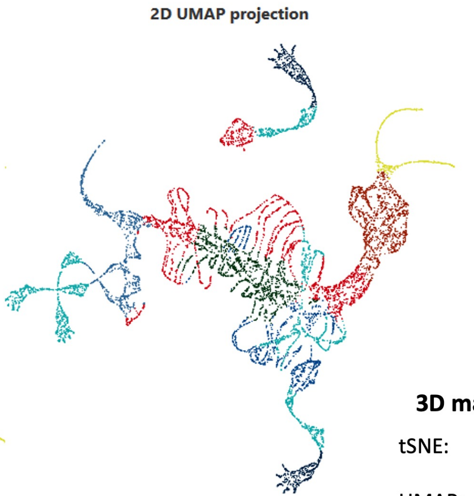
## UMAP: Uniform Manifold Approximation and Projection

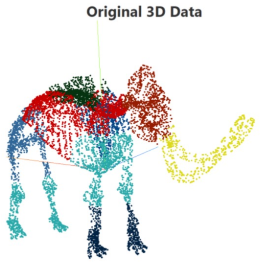
- Speed – mostly a level of maths I'm not going to get into!
  - UMAP skips a normalisation step in the calculation of high dimensional distances which speeds it up
  - In the 2D projection UMAP uses a more efficient method to shuffle the cells into their final position
    - Doesn't have to measure every cell to decide on what to move
    - Uses an algorithm which can be multi-threaded
    - Algorithm is more deterministic, allowing more data to be projected later

source: Simon Andrews

# UMAP is better than tSNE?







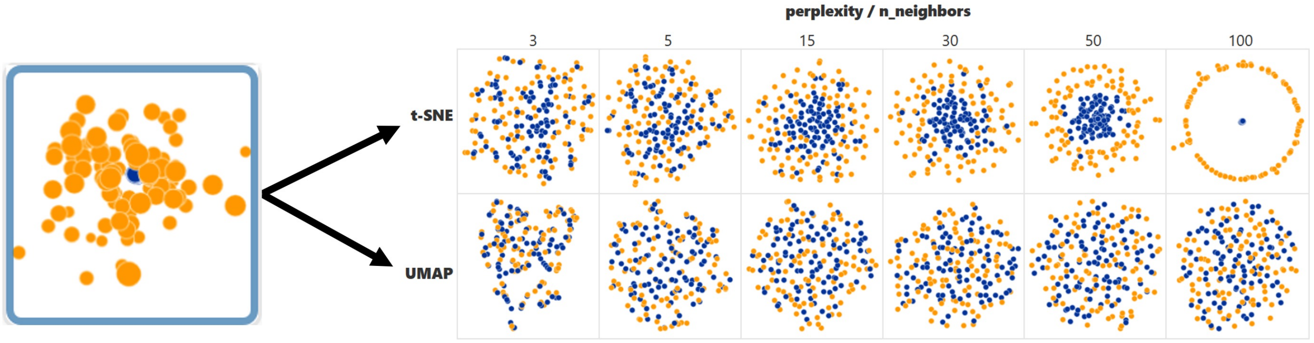
**3D mammoth skeleton projected into 2D**

| | | |
|---|---|---|
| tSNE: | Perplexity 2000 | 2h 5min |
| UMAP: | Nneigh 200, mindist 0.25, | 3min |

https://pair-code.github.io/understanding-umap/

source: Simon Andrews

## UMAP is better than tSNE?

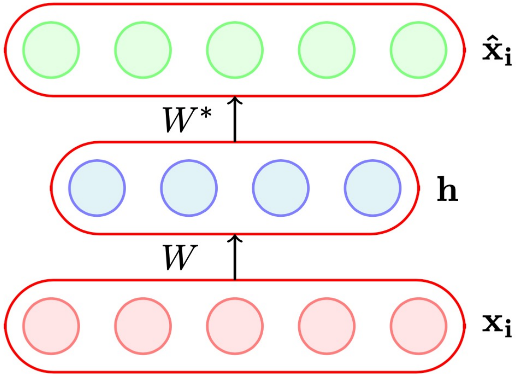


- It may perform better on more complex datasets
- It’s certainly quicker

https://pair-code.github.io/understanding-umap/

source: Simon Andrews

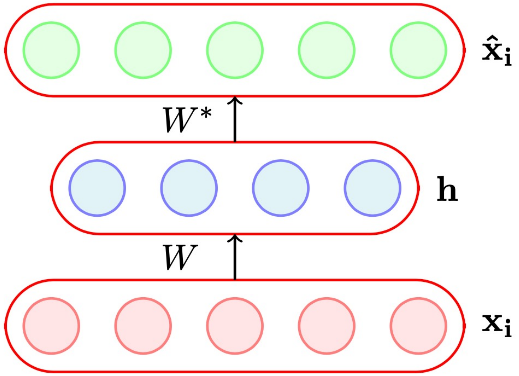
## Dimension Reduction using Deep Learning: Auto-Encoder



- An autoencoder is a special type of feed forward neural network which does the following

source: Mitesh M. Khapra

## Dimension Reduction using Deep Learning: Auto-Encoder

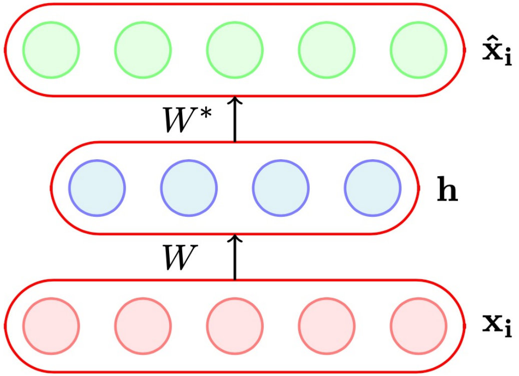


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$

source: Mitesh M. Khapra

## Dimension Reduction using Deep Learning: Auto-Encoder

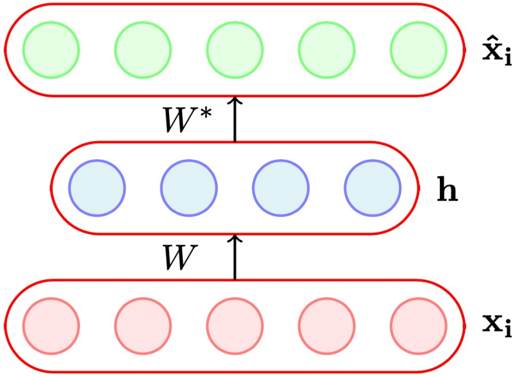


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$
- Decodes the input again from this hidden representation

source: Mitesh M. Khapra

## Dimension Reduction using Deep Learning: Auto-Encoder

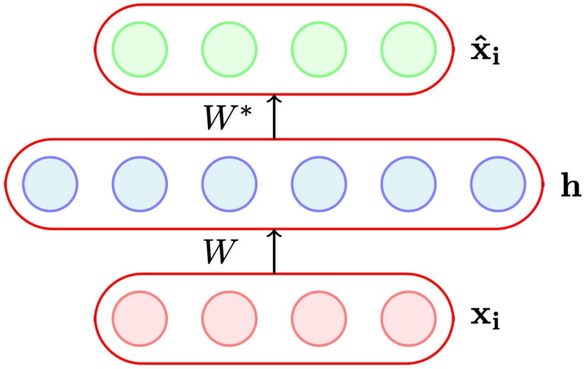


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- An autoencoder is a special type of feed forward neural network which does the following
- Encodes its input $\mathbf{x}_\mathbf{i}$ into a hidden representation $\mathbf{h}$
- Decodes the input again from this hidden representation
- The model is trained to minimize a certain loss function which will ensure that $\hat{\mathbf{x}}_\mathbf{i}$ is close to $\mathbf{x}_\mathbf{i}$ (we will see some such loss functions soon)

source: Mitesh M. Khapra

## How about a fat auto-encoder?

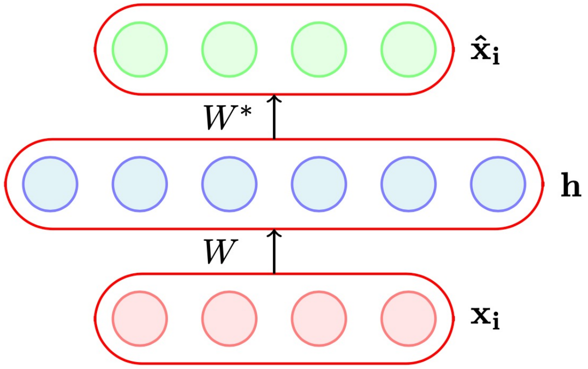


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_\mathbf{i})$
- In such a case the autoencoder could learn a trivial encoding by simply copying $\mathbf{x}_\mathbf{i}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\hat{\mathbf{x}}_\mathbf{i}$

source: Mitesh M. Khapra

## How about a fat auto-encoder?

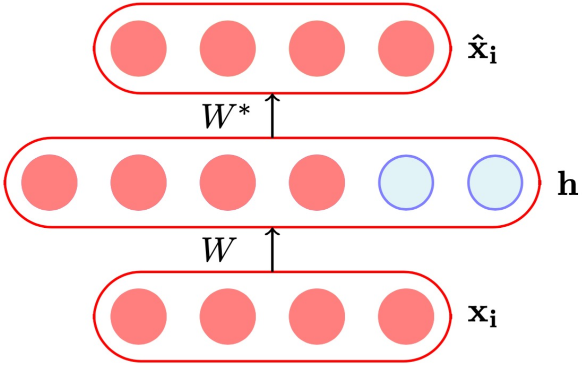


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_\mathbf{i})$
- In such a case the autoencoder could learn a trivial encoding by simply copying $\mathbf{x}_\mathbf{i}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\hat{\mathbf{x}}_\mathbf{i}$

source: Mitesh M. Khapra

## How about a fat auto-encoder?

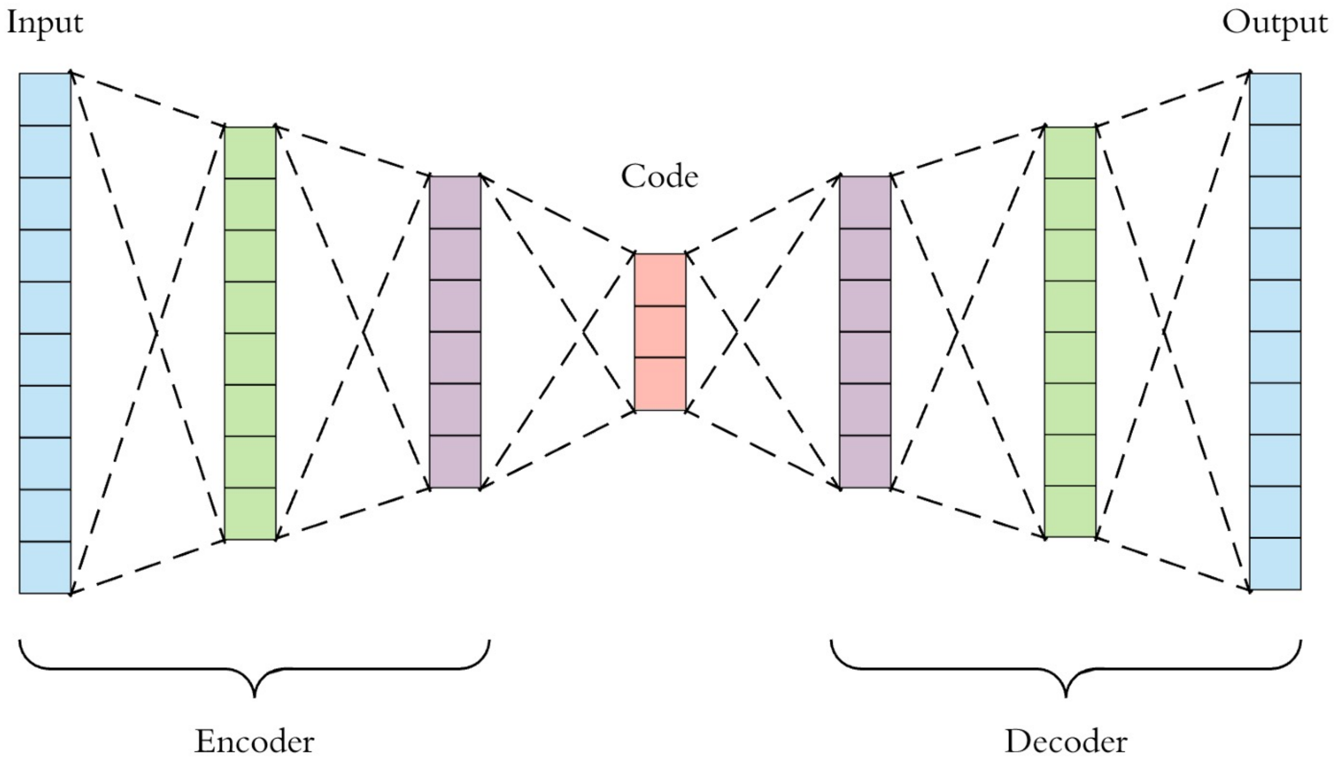


$$\mathbf{h} = g(W\mathbf{x}_\mathbf{i} + \mathbf{b})$$
$$\hat{\mathbf{x}}_\mathbf{i} = f(W^*\mathbf{h} + \mathbf{c})$$

- Let us consider the case when $\dim(\mathbf{h}) \geq \dim(\mathbf{x}_\mathbf{i})$
- In such a case the autoencoder could learn a trivial encoding by simply copying $\mathbf{x}_\mathbf{i}$ into $\mathbf{h}$ and then copying $\mathbf{h}$ into $\hat{\mathbf{x}}_\mathbf{i}$
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data

source: Mitesh M. Khapra

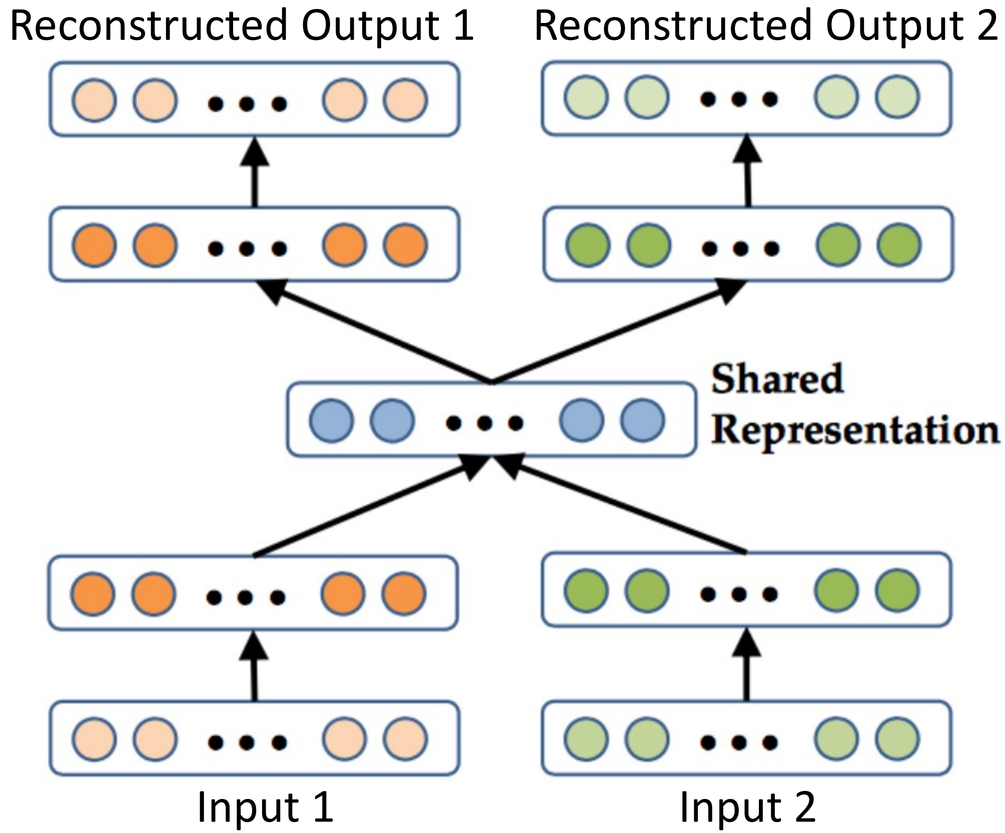
## Going Deeper: Stacked Auto-encoder



## How to choose encoder/decoder

- Tabular Data (e.g., clinical variables, gene expression profiles etc.) : Multi-layer Perceptron/Fully Connected Layers
- Imaging Data (e.g., MRI, CT scans etc.) : Convolutional Neural Networks
- Sequence Data (e.g., Texts, ECG etc.) : Recurrent Neural Networks

## Multi-Modal Auto-encoder



## Take-Aways

- Dimension Reduction is useful for removing noise, visualization, reducing computational cost, data compression.
- PCA is a linear dimension reduction method that tries to maximize the variances in low-dimensional space.
- T-SNE and UMAP are non-linear visualization approaches that aim to preserve neighboring similarities.
- Deep Auto-Encoder is a non-linear representation learning approach that aims to reconstruct the inputs.