Lecture 9: Clustering and Generative Models

Matt Ellis and Mike Smith

Session outline

Unsupervised learning

- Recap PCA and Auto-encoders
- Clustering: K-Means
- Clustering: Spectral Clustering

Generative models

- Review of Bayes' theorem and Bayesian inference

Learning Objectives

By the end of this session you should be able to:

1. Explain what is meant by clustering in unsupervised learning.
2. Explain the method and objective function of K-means clustering.
3. Explain the method of spectral clustering.
4. Calculate the cut and normalised cut of a graph.
5. Recall the definition of a generative model.
6. Apply Bayes' theorem for inference.

Unsupervised Learning

Session outline

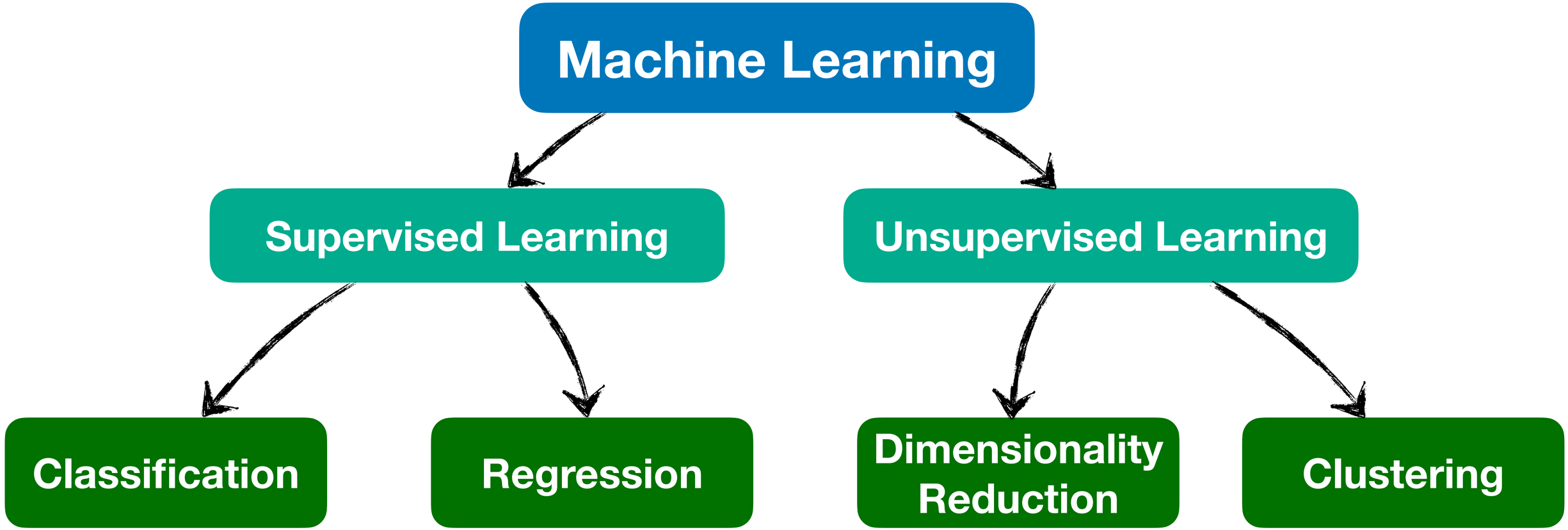
Unsupervised learning

- **Recap PCA and Auto-encoders**
- Clustering: K-Means
- Clustering: Spectral Clustering

Generative models

- Review of Bayes' theorem and Bayesian inference

Supervised vs Unsupervised



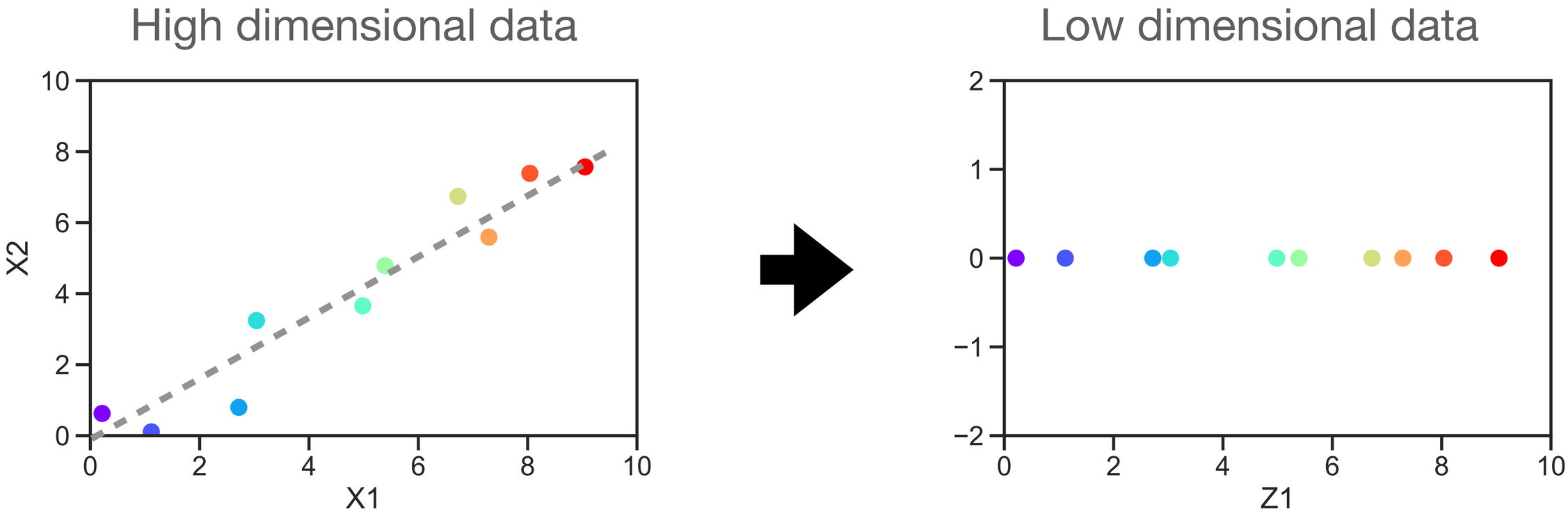
- Click-through-rate
- Image classification
- Diagnostics
- Fraud detection

- Market forecasting
- Ad popularity

- Big data visualisation
- Feature discovery
- Structure discovery

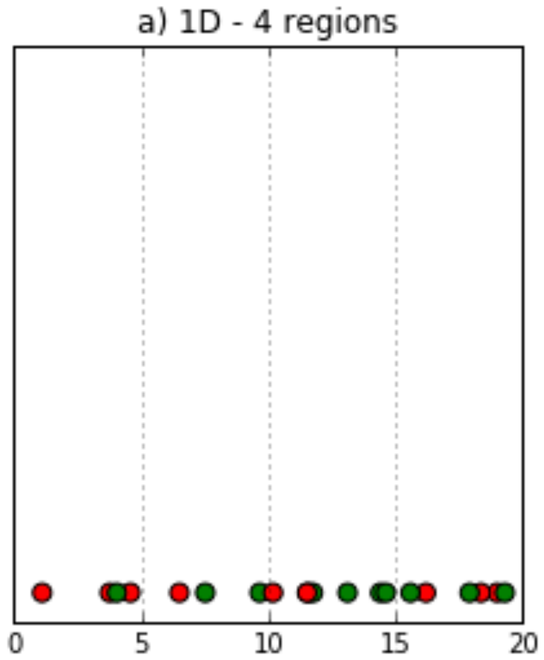
- Customer segmentation
- Targeted marketing
- Recommendation

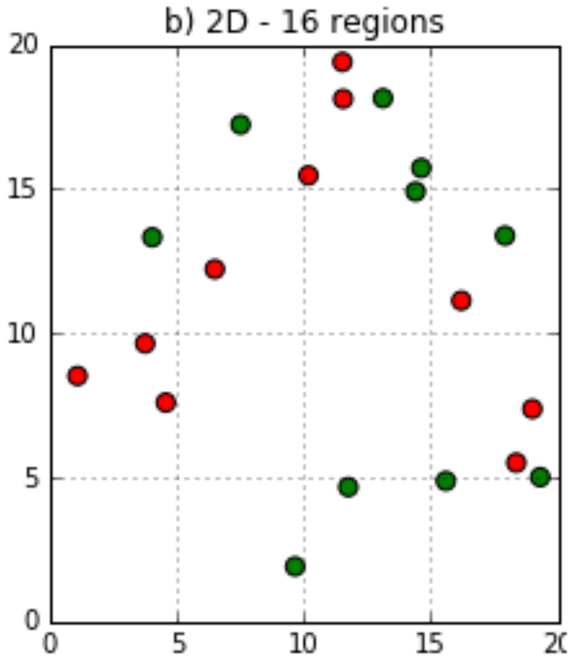
Dimensionality Reduction

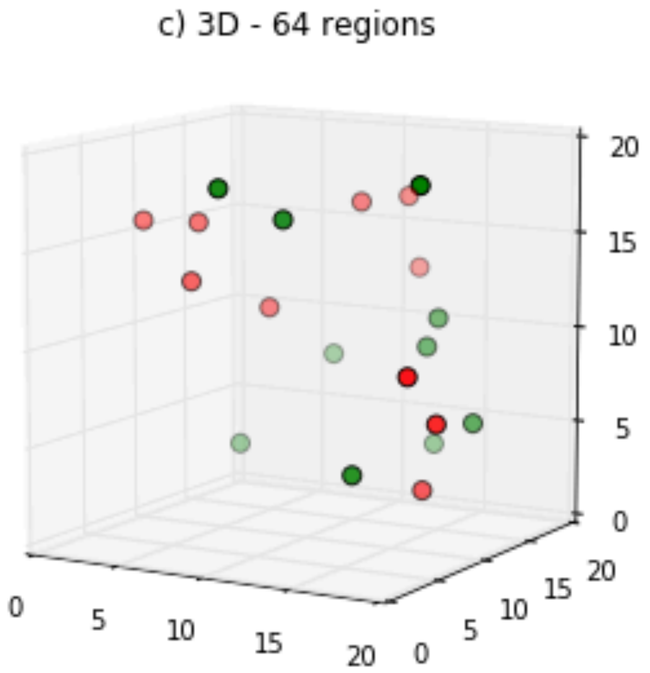


Why apply dimensionality reduction?

- Curse of dimensionality
- Reduce redundancy
- Visualisation







Question

USPS handwritten digit dataset:

Image size: 64 by 57

Black or white pixels (1 bit)

How many dimensions is a single image?

$$64 \times 57 = 3648$$

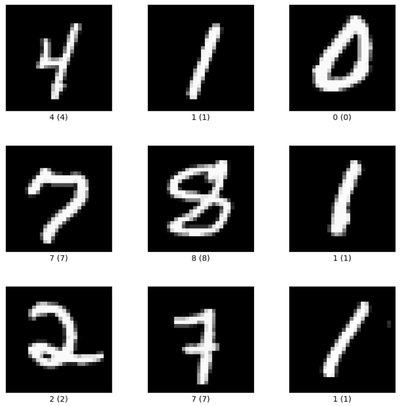
How many possible images?

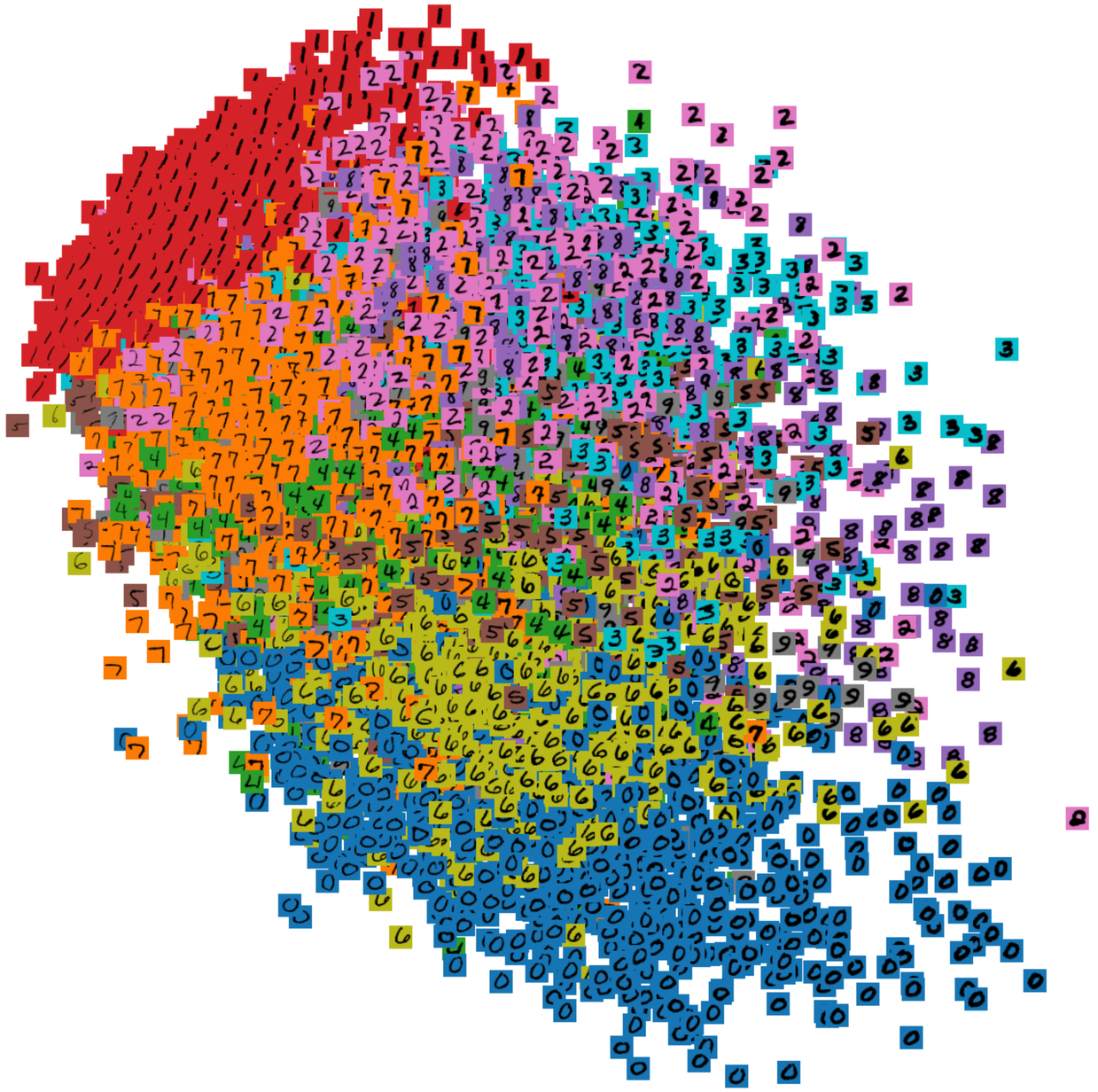
$$2^{64 \times 57} = 2^{3648} = ?$$

Principal Component Analysis

PCA Demo

Demonstration:
https://projector.tensorflow.org/



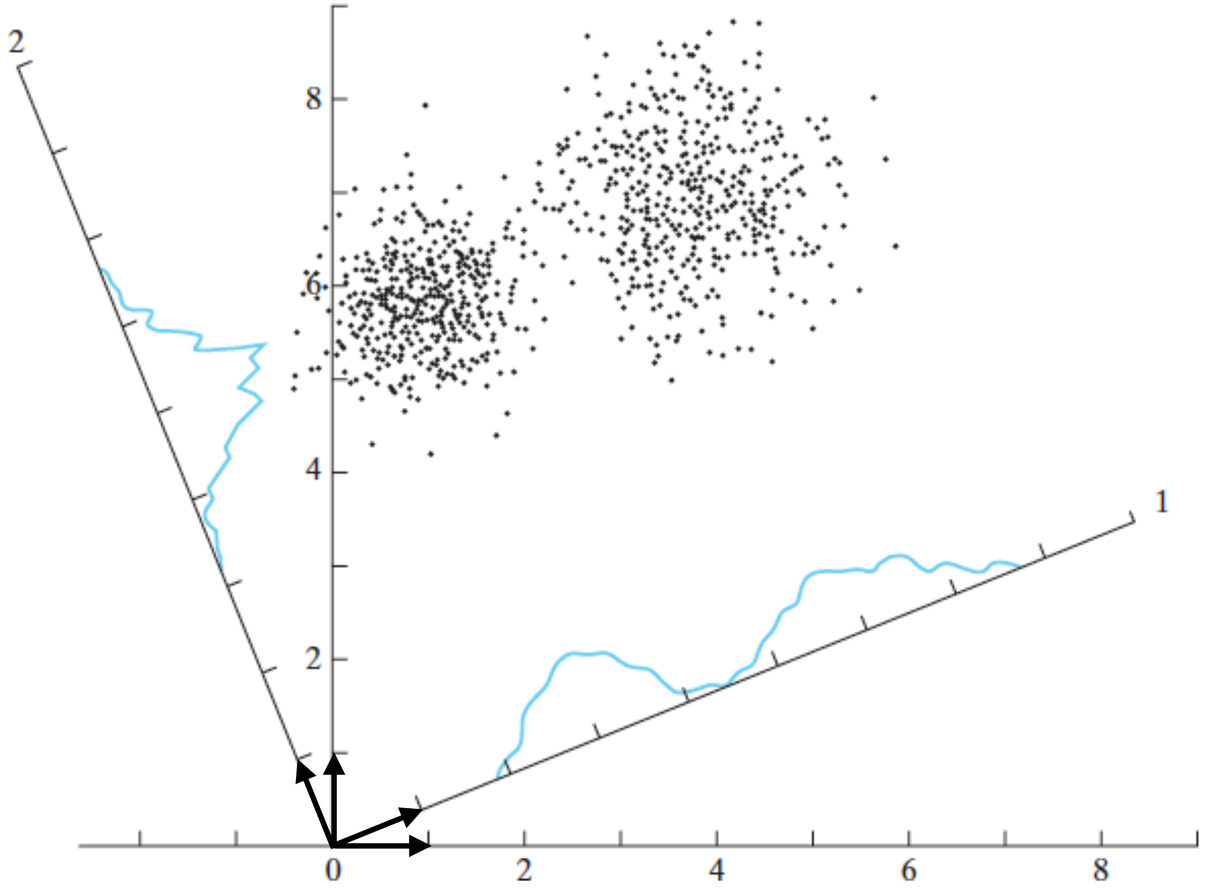


Finding a basis to best represent data

PCA

1. Rotate the data with some **rotation matrix** (linear transformation) so that features are uncorrelated.
2. Keep the dimensions with **highest variance** for DR.

Adapted from Neural Networks and Learning Machines by Simon Haykin (Pearson 2009)



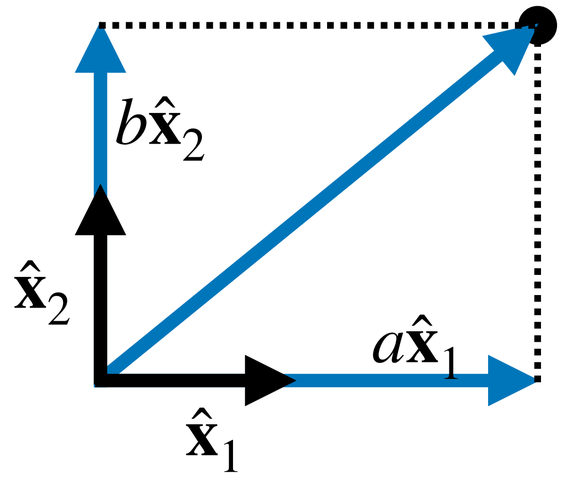
Unit vectors that form the basis

Basis vectors for datapoints

A point in space is described by its projections on a set of basis vectors.

Datapoint

$\mathbf{x} = a\hat{\mathbf{x}}_1 + b\hat{\mathbf{x}}_2$



Projections on basis vectors

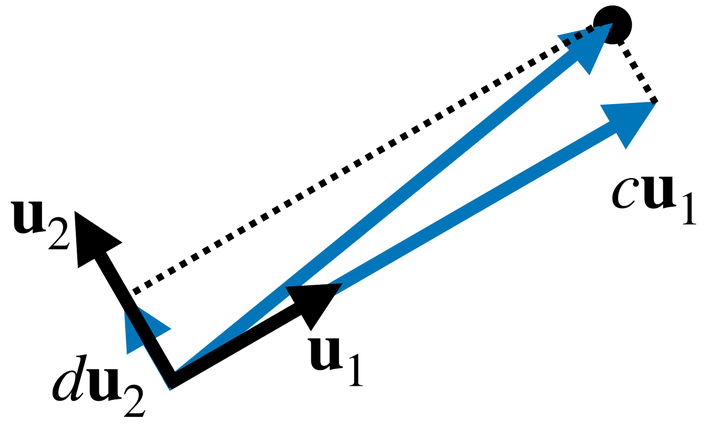
$a = \hat{\mathbf{x}}_1^T \mathbf{x}$

$b = \hat{\mathbf{x}}_2^T \mathbf{x}$

Datapoint vector = $[a, b]^T$

Datapoint in transformed basis

$\mathbf{x} = c\mathbf{u}_1 + d\mathbf{u}_2$



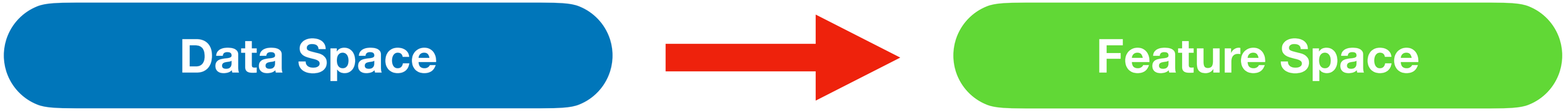
Projections on basis vectors

$c = \mathbf{u}_1^T \mathbf{x}$

$d = \mathbf{u}_2^T \mathbf{x}$

Datapoint vector in new basis = $[c, d]^T$

Rotating the basis as a transformation



$$\mathbf{x} = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1^T \mathbf{x} \\ \mathbf{u}_2^T \mathbf{x} \end{bmatrix}$$

Transformation (rotation) matrix

$$\mathbf{U}^T = \begin{bmatrix} \mathbf{u}_1^T \\ \mathbf{u}_2^T \end{bmatrix}$$

$$\mathbf{y} = \mathbf{U}^T \mathbf{x}$$

Variances and Covariance

PCA: Find directions that **maximises the variance** of the transformed data.

Recall that:

Variance & Covariance - measure of the "spread" of a set of points around their centre (mean).

Variance (**scalar) -** measure of the spread **each dimension separately**.

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

- Covariance is measured between two dimensions.
- Covariances sees if there is a relation between them.
- The covariance of a dimension with itself is the variance

PCA

Find directions that **maximises the variance** of the transformed data.

How do we find these directions?

Eigenvectors of the sample covariance (scatter) matrix

$$C_{ij} = \frac{1}{N} \sum_{n=1}^{N} (x_{ni} - \mu_i)(x_{nj} - \mu_j)$$

Question: what is the covariance matrix in the projected space?

Covariance in the projected space

The covariance is defined as

$$\mathrm{Cov}(y_i, y_j) = \frac{1}{N}\sum_{n=1}^{N}(y_{ni} - \mathbb{E}(y_i))(y_{nj} - \mathbb{E}(y_j))$$

$$y_{ni} = \mathbf{u}_i^T \mathbf{x}_n$$

$$\mathbb{E}(y_i) = \mathbf{u}_i^T \boldsymbol{\mu}$$

$$= \frac{1}{N}\sum_{n=1}^{N} \mathbf{u}_i^T(\mathbf{x}_n - \boldsymbol{\mu})(\mathbf{x}_n - \boldsymbol{\mu})^T\mathbf{u}_j$$

$$= \mathbf{u}_i^T \mathbf{C} \mathbf{u}_j$$

Eigenvectors are directions of maximum variance

Find the first direction $\mathbf{u}_1$ that maximises the variance in the projected space.

Optimisation criterion for 1st PCA including a unit-norm constraint via Lagrange multipliers:

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

Variance of the data in direction $\mathbf{u}_1$

Constraint to keep $|\mathbf{u}_1| = 1$

We want to maximise this, so what can we do?

Eigenvectors are directions of maximum variance

Find derivative w.r.t $\mathbf{u}_1$ and set to zero:

$$L(\mathbf{u}_1, \lambda_1) = \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \lambda_1 \left(1 - \mathbf{u}_1^T \mathbf{u}_1\right)$$

$$\frac{dL(\mathbf{u}_1, \lambda_1)}{d\mathbf{u}_1} = 2\mathbf{C}\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1 = \mathbf{0}$$

$$\mathbf{C}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

Eigenvalue problem!

Multiple solutions, which do we pick?

Selecting other transformation vectors

Further directions: **Orthogonal** (uncorrelated) to the first and each other

Use the remaining **eigenvectors** of **C:**

- **Eigenvectors** - basis vectors, principal components
- **Eigenvalues** - the variance of the data captured by that direction

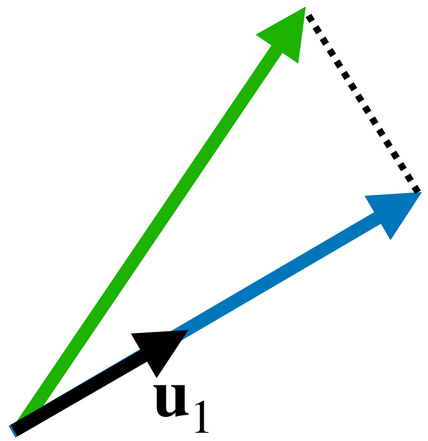
For a D dimensional input, we will have D eigenvectors.
Create transformation matrix by forming these as columns.

$$\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_D]$$

For DR, we select the *k* eigenvectors with the highest eigenvalues to keep, drop the rest.

Reconstruction error vs maximum variance

We could minimise the error when transforming back (reconstruction).



Transform $y_{n1} = \mathbf{u}_1^T \mathbf{x}_n$

Inverse Transform $\tilde{\mathbf{x}}_n = \mathbf{u}_1 y_{n1}$

$$E = \frac{1}{2N} \sum_n \left(\mathbf{x}_n - \tilde{\mathbf{x}}_n \right)^2$$

$$= \frac{1}{2N} \sum_n \left(\mathbf{x}_n - \mathbf{u}_1 (\mathbf{u}_1^T \mathbf{x}_n) \right)^2$$

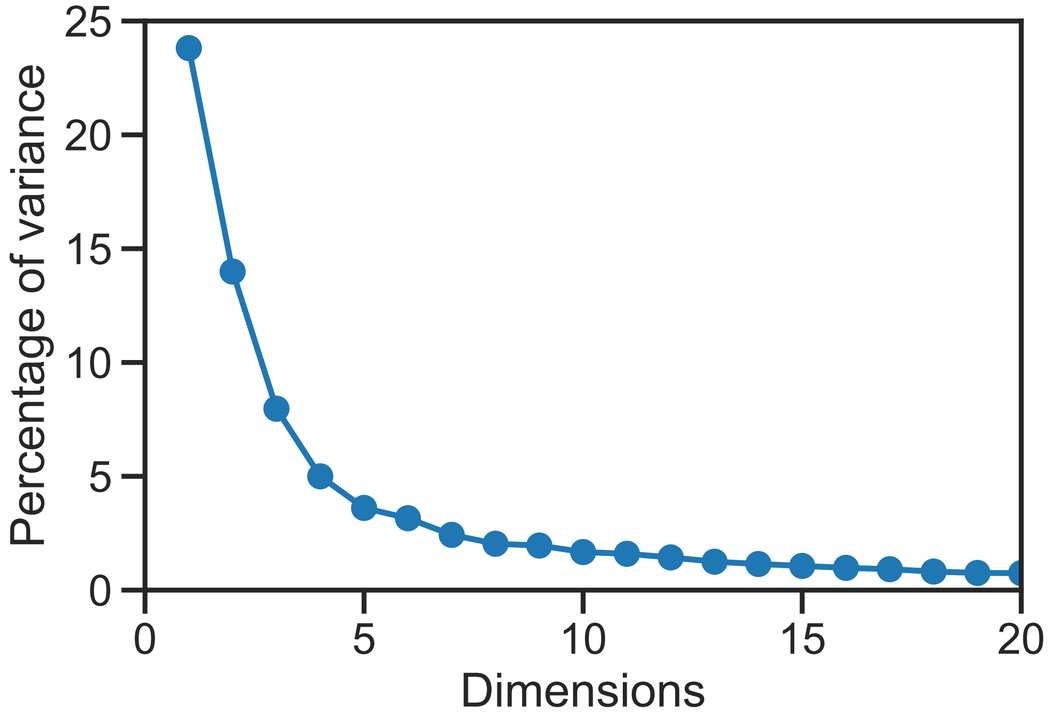
$$= - \mathbf{u}_1^T \mathbf{C} \mathbf{u}_1 + \frac{1}{N} \sum_n |\mathbf{x}_n|^2$$

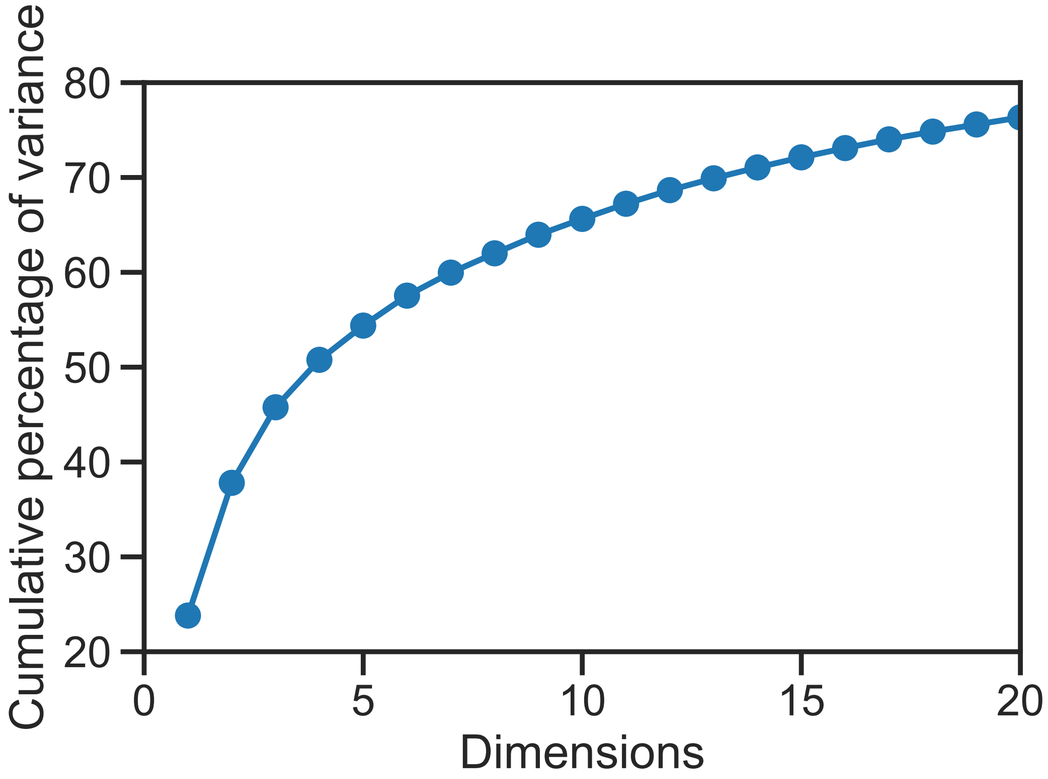
So min reconstruction error = max variance

How many dimensions to keep?

Pick based on percentage of variance kept/lost

Look for an ‘elbow’ in a **scree plot** (plot of explained variance or eigenvalues).





Representation and Reconstruction

Example: Olivetti faces dataset.

Representation:
Face $\mathbf{x}$ in 'face space' coordinates.

$$\mathbf{y} = \mathbf{U}^T(\mathbf{x} - \boldsymbol{\mu}) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Reconstruction: use representation to rebuild using eigenvectors as basis.

$=$

$+$

$$\tilde{\mathbf{x}} = \boldsymbol{\mu} + y_1\mathbf{u}_1 + y_2\mathbf{u}_2 + y_3\mathbf{u}_3 + \ldots$$

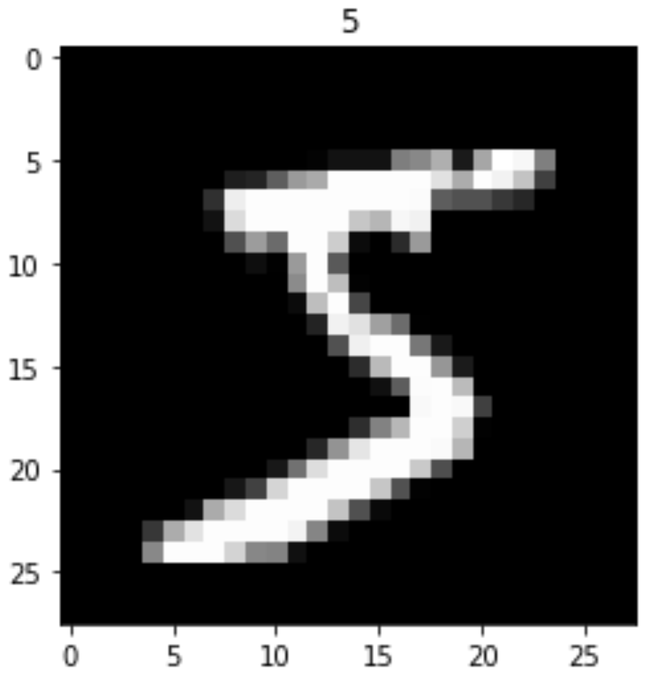
PCA ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: Linear projection layer $\mathbf{y} = \mathbf{U}^T\mathbf{x}$
  - **Parameters**: The principal components (eigenvectors)
  - **Hyper-parameters**: Number of components to keep, *k*
- **Evaluation metric**: Variance explained by the PCs
- **Optimisation**: Eigen-decomposition

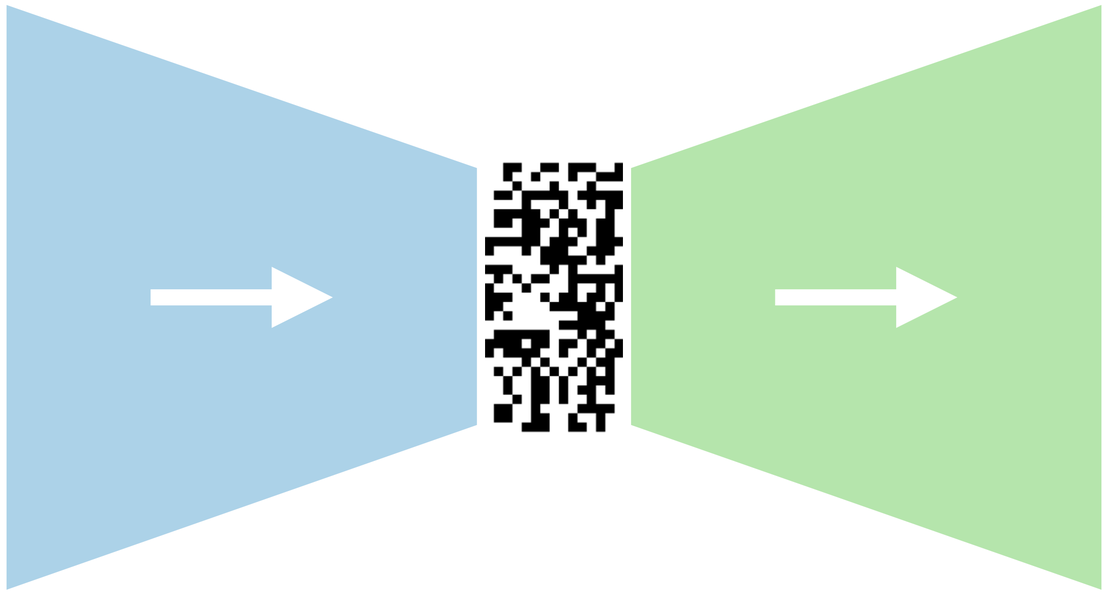
Auto-encoders

DR as encoding

Original data



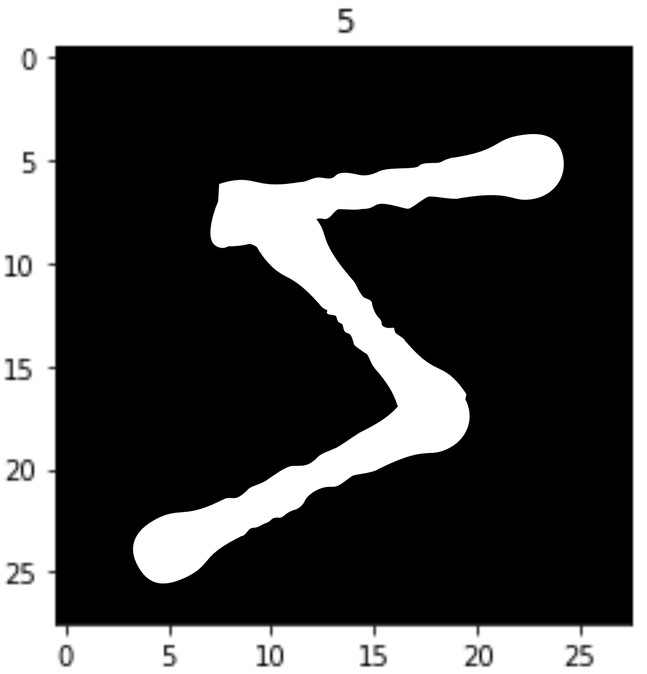
‘Code’
Useful representation



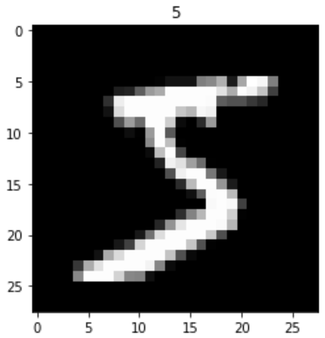
Encoder
Compress or extract features

Decoder
Recover data from code

Decoded data

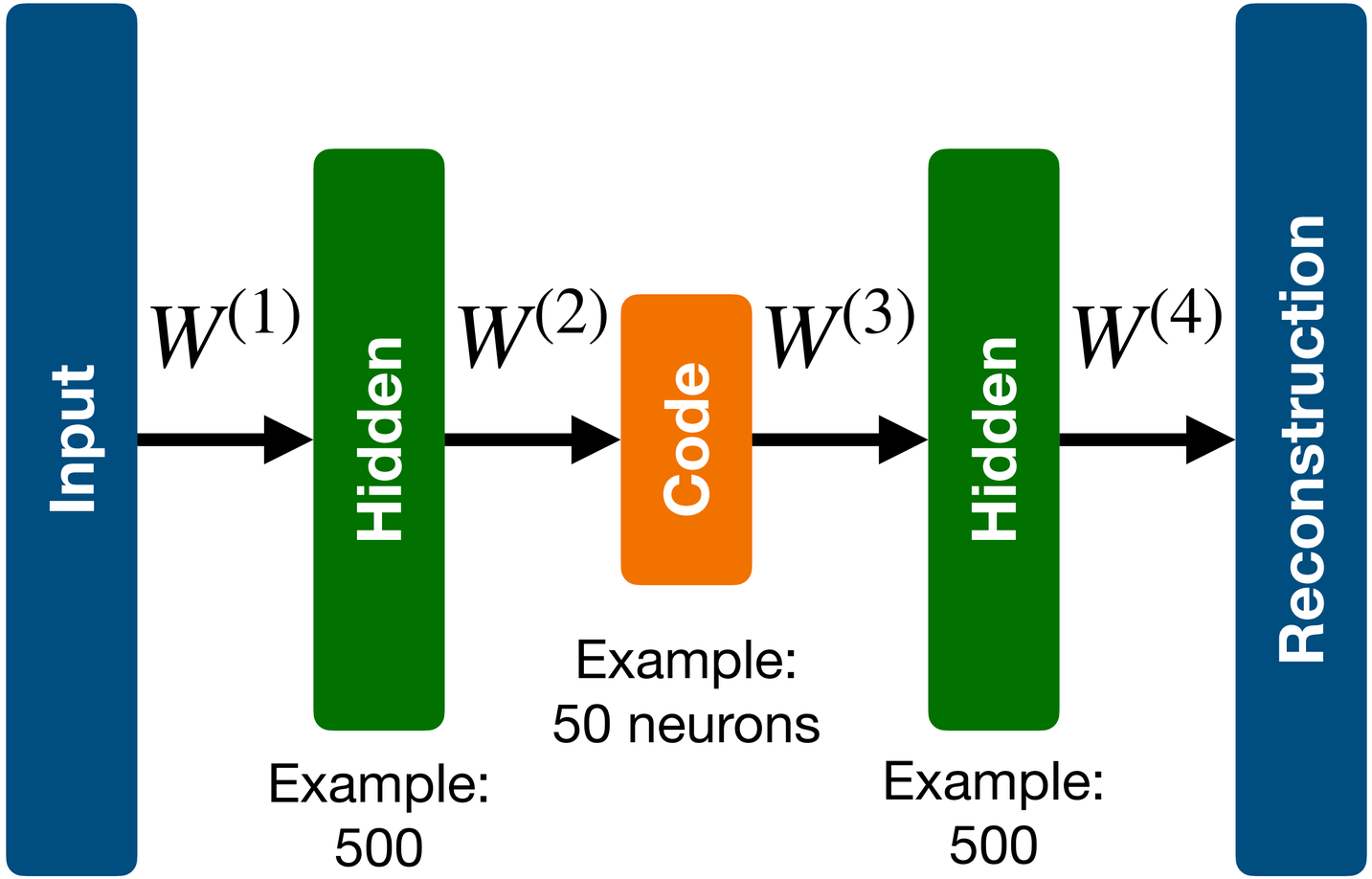


Auto-encoder structure



Example:
782 neurons

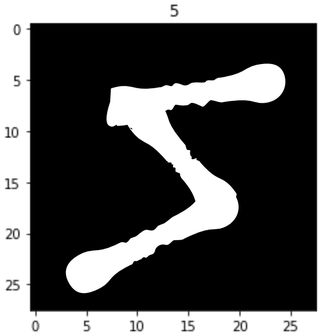
$\mathbf{x}_n$



Example:
500
neurons

Example:
50 neurons

Example:
500
neurons



Example:
782 neurons

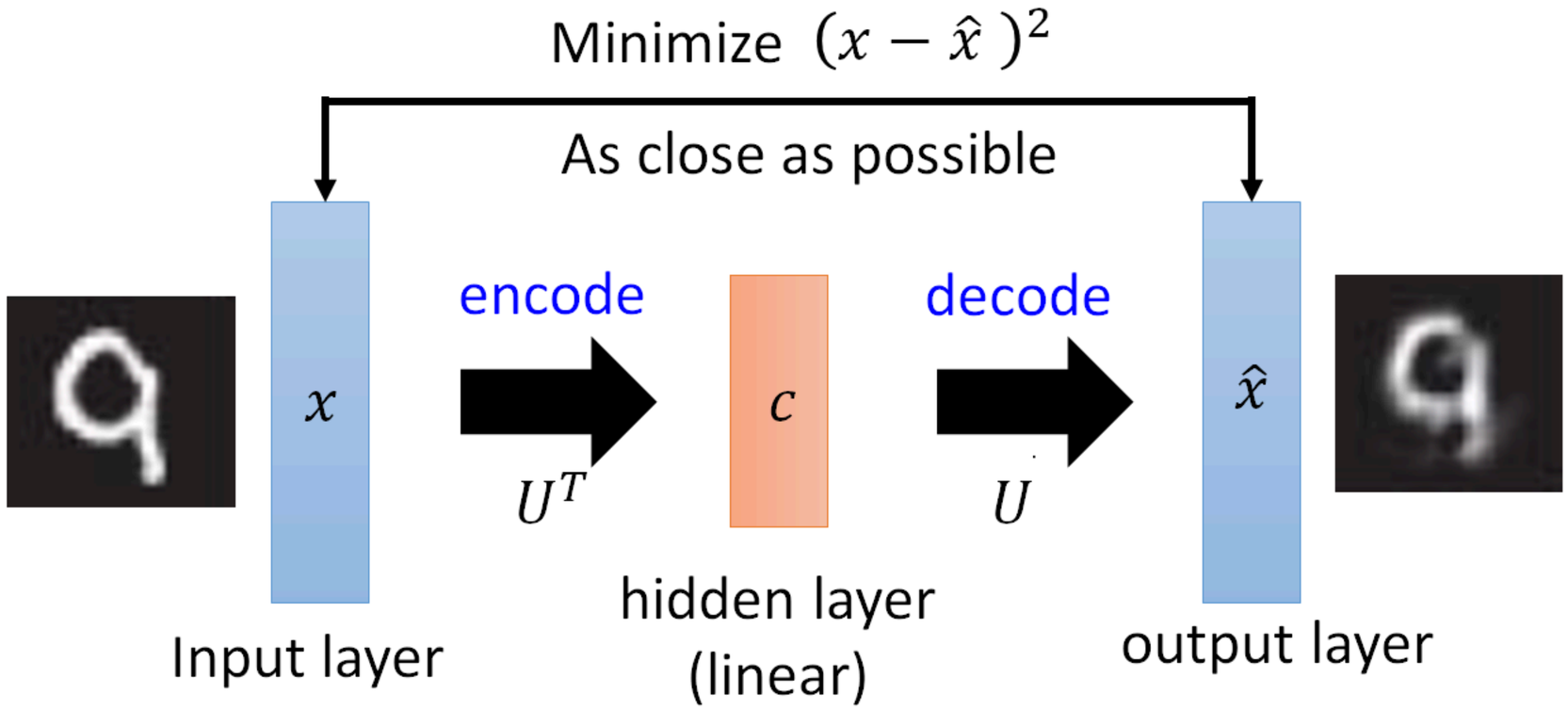
$\tilde{\mathbf{x}}_n$

What will our objective (error or loss) function be?

$$E = \frac{1}{2N} \sum_n \left(\mathbf{x}_n - \tilde{\mathbf{x}}_n \right)^2$$

PCA as a linear Auto-encoder

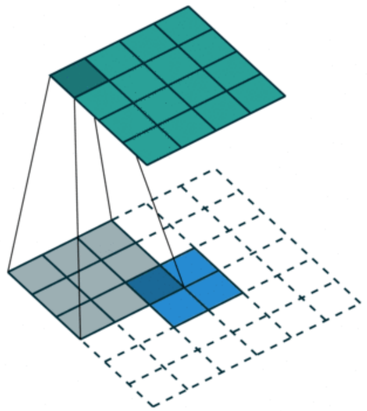
PCA: single weight layer shared as encoder and decoder.



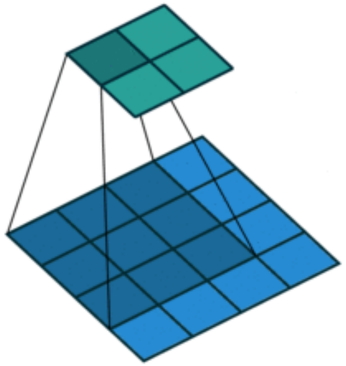
Transposed Convolutional Layer

Need to ‘decode’ the output of the convolutions.

No padding or strides

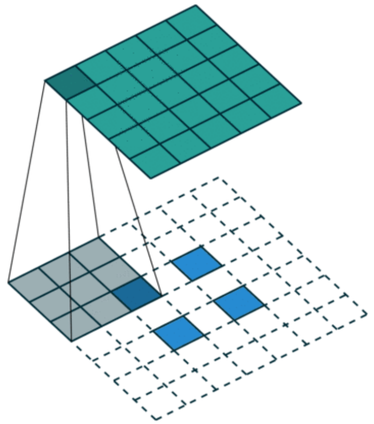


Reconstructed input needs to be 4 by 4, expand from 2 by 2

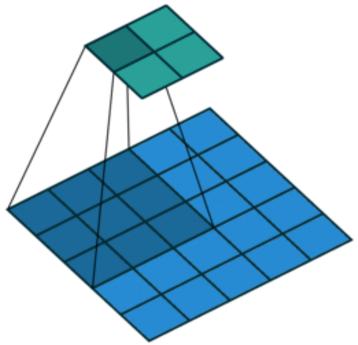


Original input is 4 by 4
Convolution result is 2 by 2

No padding, stride = 2



Reconstructed input needs to be 5 by 5, expand from 2 by 2



Original input is 5 by 6
Convolutional result is 2 by 2

More at https://github.com/vdumoulin/conv_arithmetic

Auto-encoder ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: encoder + decoder, multiple neural network layers
  - **Parameters**: layer weights and biases
  - **Hyper-parameters**: number of layers, layer parameters, learning rate
- **Evaluation metric**: Mean-Squared Error (reconstruction error)
- **Optimisation**: Gradient descent (back-prop) or similar

PyTorch Auto-encoder

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            # 1 input image channel, 16 output channel, 3x3 square convolution
            nn.Conv2d(1, 16, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, 7)
        )
        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 7),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, 3, stride=2, padding=1, output_padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, 1, 3, stride=2, padding=1, output_padding=1),
            nn.Sigmoid()  #to range [0, 1]
        )

    def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

Take Home Messages

- Unsupervised learning: **no labels**, learn from **data only**.
- Dimensionality reduction: useful for visualisation, feature discovery.
- Principal component analysis (PCA): use **eigenvectors** of the **covariance matrix** to find directions of **maximum variance.**
- Auto-encoder: use **neural network** to **encode data into a 'code'**, use a mirrored network to **decode** it to **reconstruct original data**.
- **Reconstruction error** - mean squared difference of our recovered data (after transforming back) against the original data.

Further Reading

Chapter 7 (sections 7.1 and 7.2) in A First Course in Machine Learning by Rogers and Girolami.

Chapter 14 of Deep Learning by Goodfellow.

Available at https://www.deeplearningbook.org/

Clustering

Session outline

Unsupervised learning

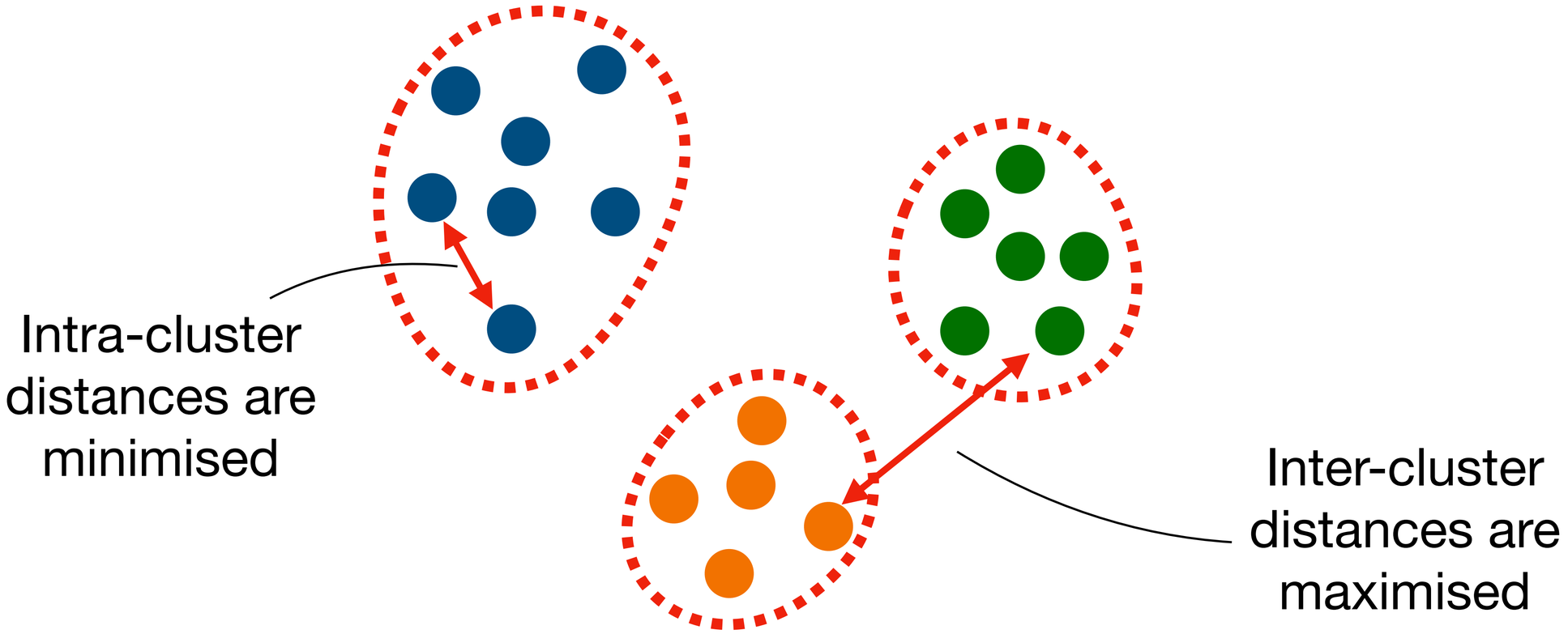
- Recap PCA and Auto-encoders
- **Clustering: K-Means**
- Clustering: Spectral Clustering

Generative models

- Review of Bayes' theorem and Bayesian inference

What is clustering?

Find groups of objects such that objects in one group will be more similar (more related) to one another and less similar to objects in other groups.



K-Means Clustering

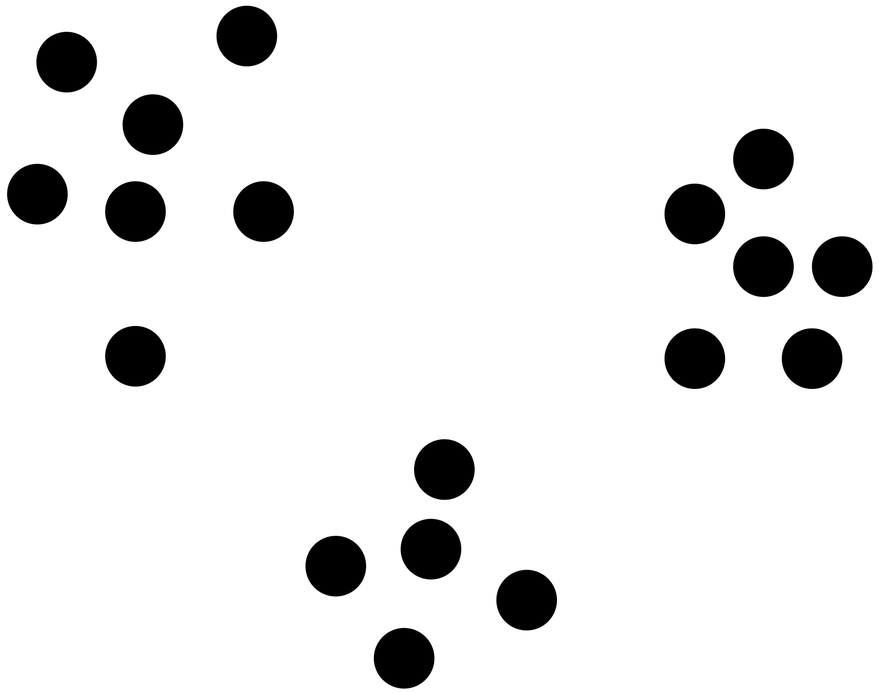
Minimise objective function: with respect to what?

Reconstruction Error (Within Cluster Scatter):

$$E_{\mathrm{KM}} = \frac{1}{2} \sum_{k=1}^{K} \sum_{n \in C_k} \left(\mathbf{x}_n - \mathbf{m}_k \right)^2$$

K	$\mathbf{m}_k$	C_k	n_k
number of clusters (hyper-parameter)	**centroid of cluster k (i.e weights)**	**List of data points in cluster k**	**Number of data points in cluster k**

K-Means Clustering

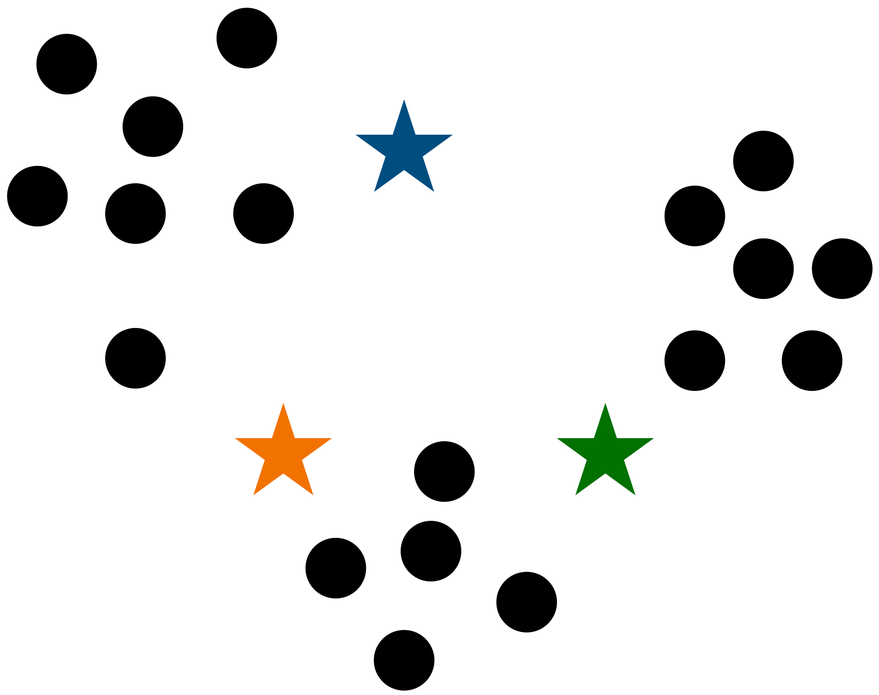


Given data $\mathbf{x}$, and k clusters to find

1. Pick random starting centroids
2. Repeat until no more changes:
   - 2.1. Assign datapoints to nearest centroid
   - 2.2. Update centroids to mean of assigned datapoints

$$\mathbf{m}_k = \frac{1}{n_k} \sum_{n \in C_k} \mathbf{x}_n$$

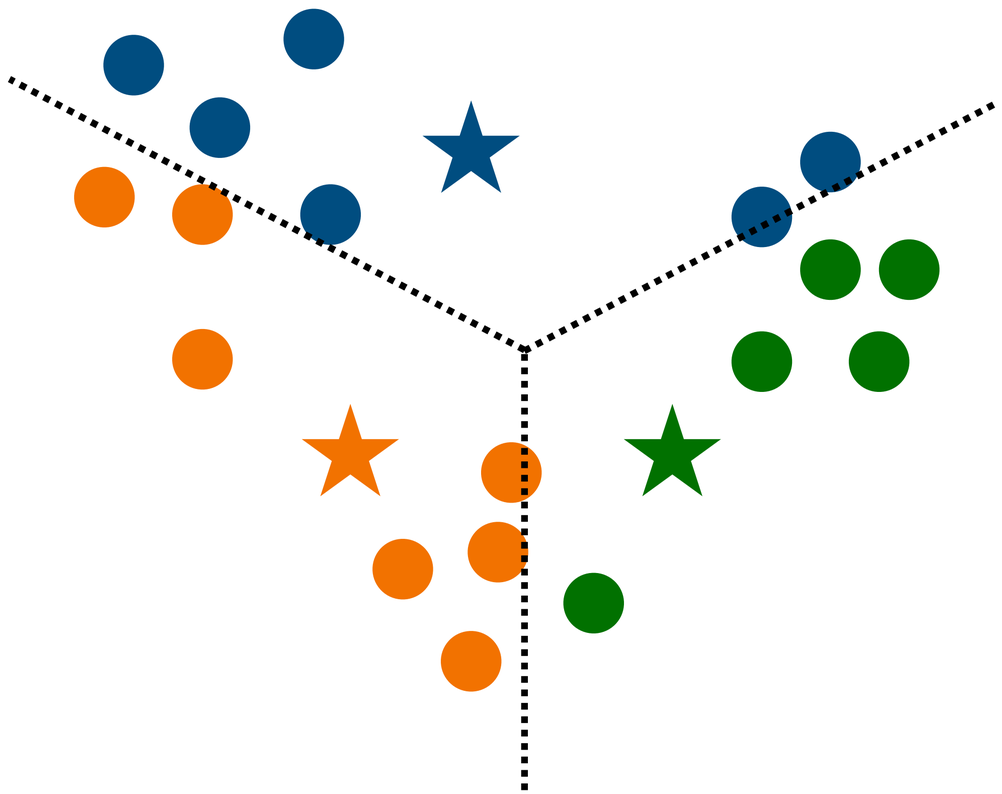
K-Means Clustering



Given data $\mathbf{x}$, and k clusters to find

1. **Pick random starting centroids**
2. Repeat until no more changes:
    2.1. Assign datapoints to nearest centroid
    2.2. Update centroids to mean of assigned datapoints

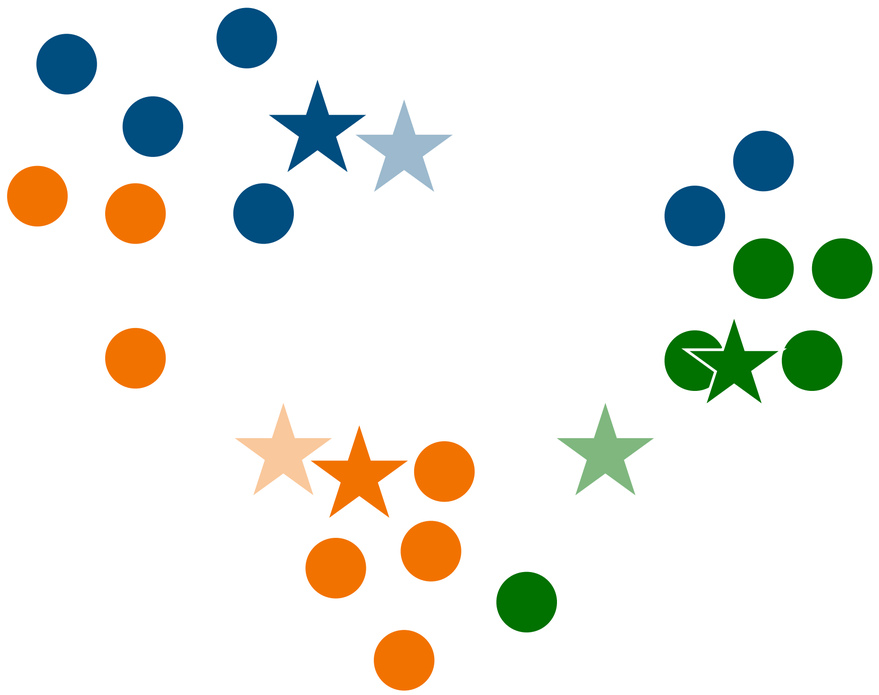
K-Means Clustering



Given data $\mathbf{x}$, and k clusters to find

1. Pick random starting centroids
2. Repeat until no more changes:
    2.1. **Assign datapoints to nearest centroid**
    2.2. Update centroids to mean of assigned datapoints

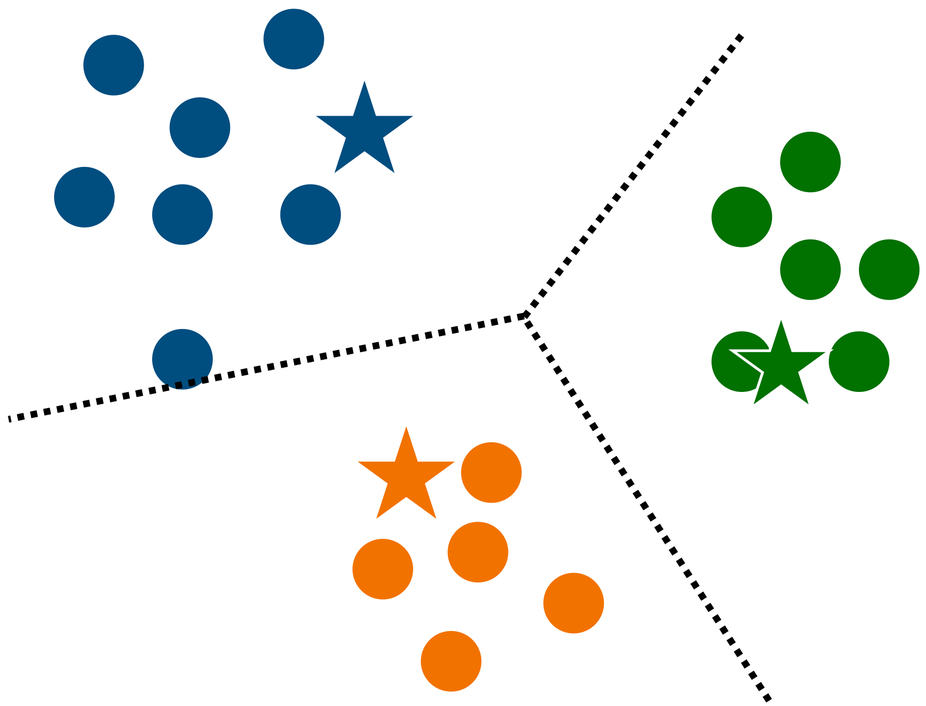
K-Means Clustering



Given data $\mathbf{x}$, and k clusters to find

1. Pick random starting centroids
2. Repeat until no more changes:
    2.1. Assign datapoints to nearest centroid
    2.2. Update centroids to mean of assigned datapoints

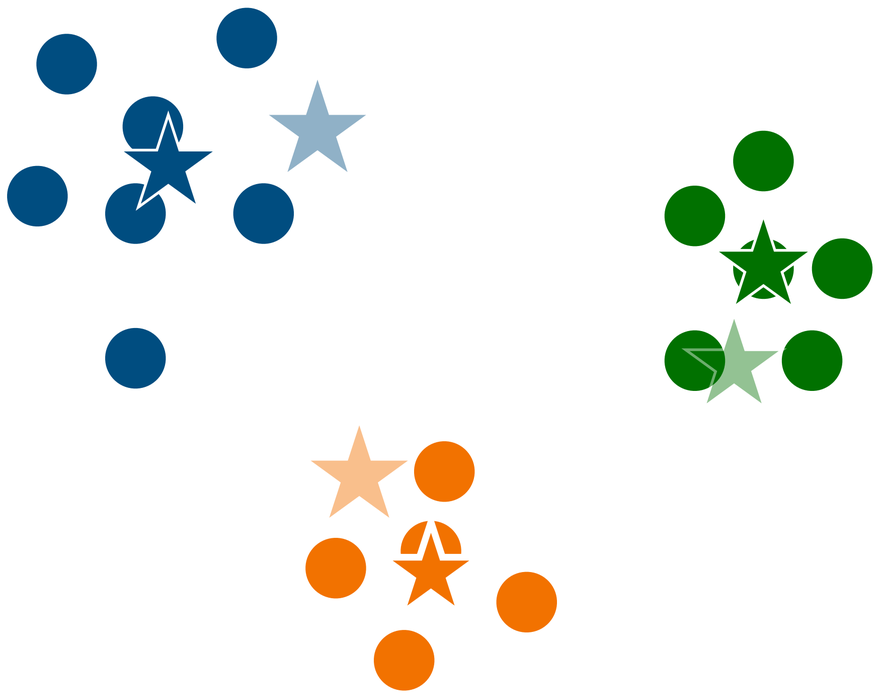
K-Means Clustering



Given data $\mathbf{x}$, and k clusters to find

1. Pick random starting centroids
2. Repeat until no more changes:
   - **2.1. Assign datapoints to nearest centroid**
   - 2.2. Update centroids to mean of assigned datapoints

K-Means Clustering



Given data $\mathbf{x}$, and k clusters to find

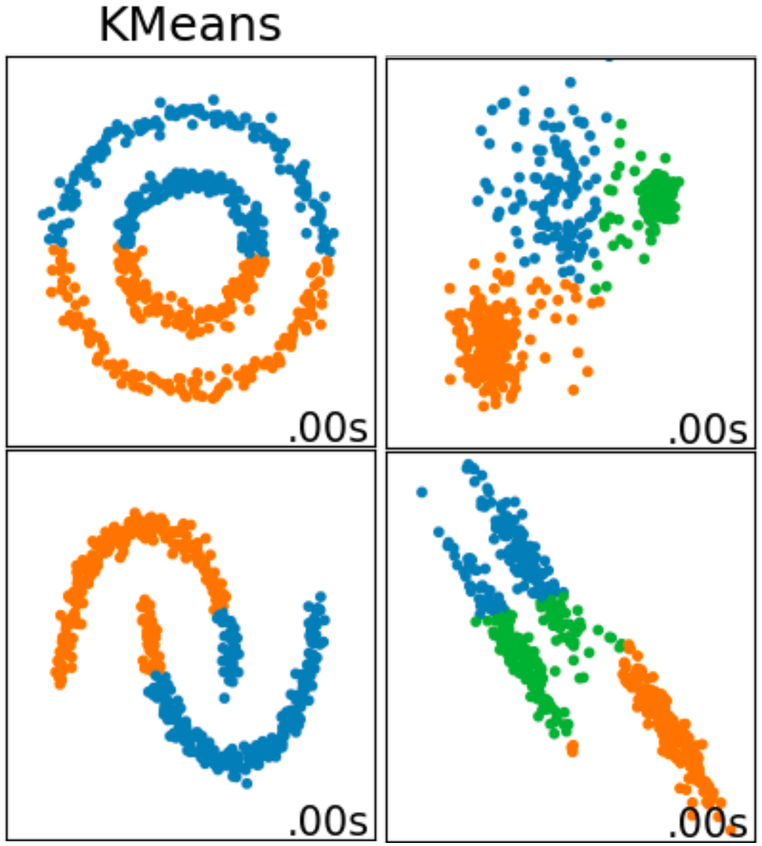
1. Pick random starting centroids
2. **Repeat until no more changes:**
    2.1. Assign datapoints to nearest centroid
    2.2. Update centroids to mean of assigned datapoints

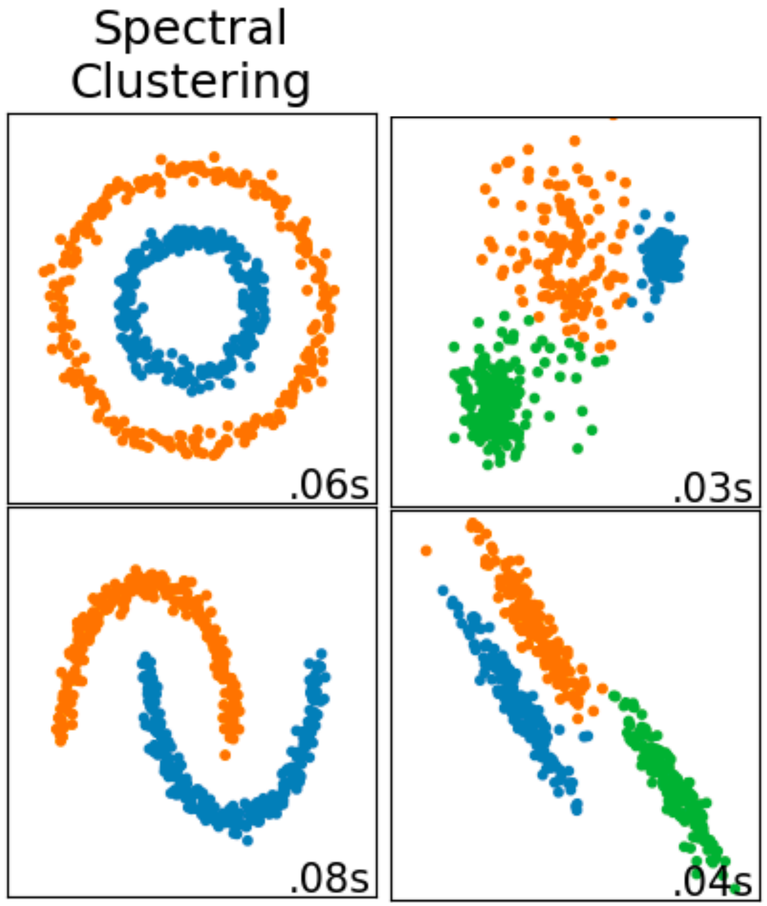
K-means Ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: linear separation between clusters
  - **Parameters**: position of cluster centroids
  - **Hyper-parameters**: number of clusters
- **Evaluation metric**: within cluster scatter
- **Optimisation**: Gradient descent (back-prop) or similar

K-Means has a problem

K-means works best for ‘spherical’ data, struggles with complex distributions.





https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html#sphx-glr-auto-examples-cluster-plot-cluster-comparison-py

Session outline

Unsupervised learning

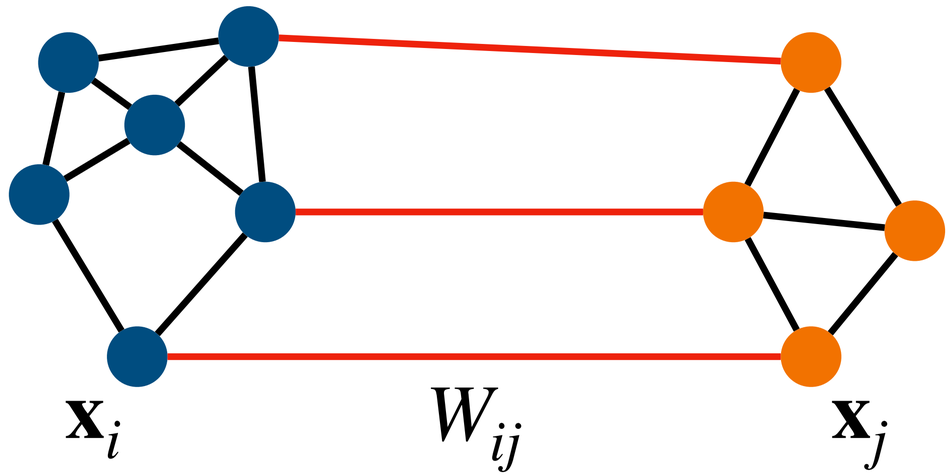
- Recap PCA and Auto-encoders
- Clustering: K-Means
- **Clustering: Spectral Clustering**

Generative models

- Review of Bayes' theorem and Bayesian inference

Spectral Clustering

Group points based on **links** on a **graph**



Similarity matrix $\mathbf{W}$

How to create the graph?

Gaussian kernel - compute similarity between objects:

$$W_{ij} = \exp\left(-\frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2\sigma^2}\right)$$

You could create:

- A fully connected graph - every datapoint is connected to every other
- K-nearest neighbour graph - each node is only connected to its K neighbours (local connectivity)

Clustering - Graph cut / partitioning

Objective: **Minimise cut**

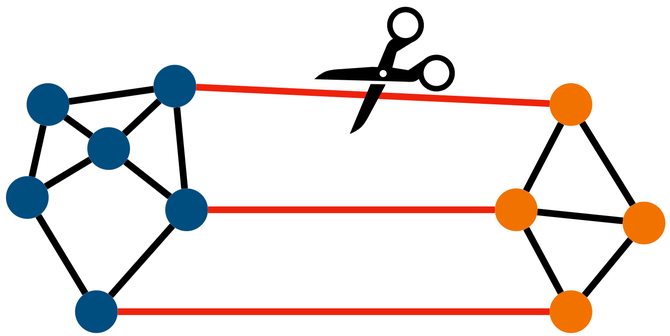
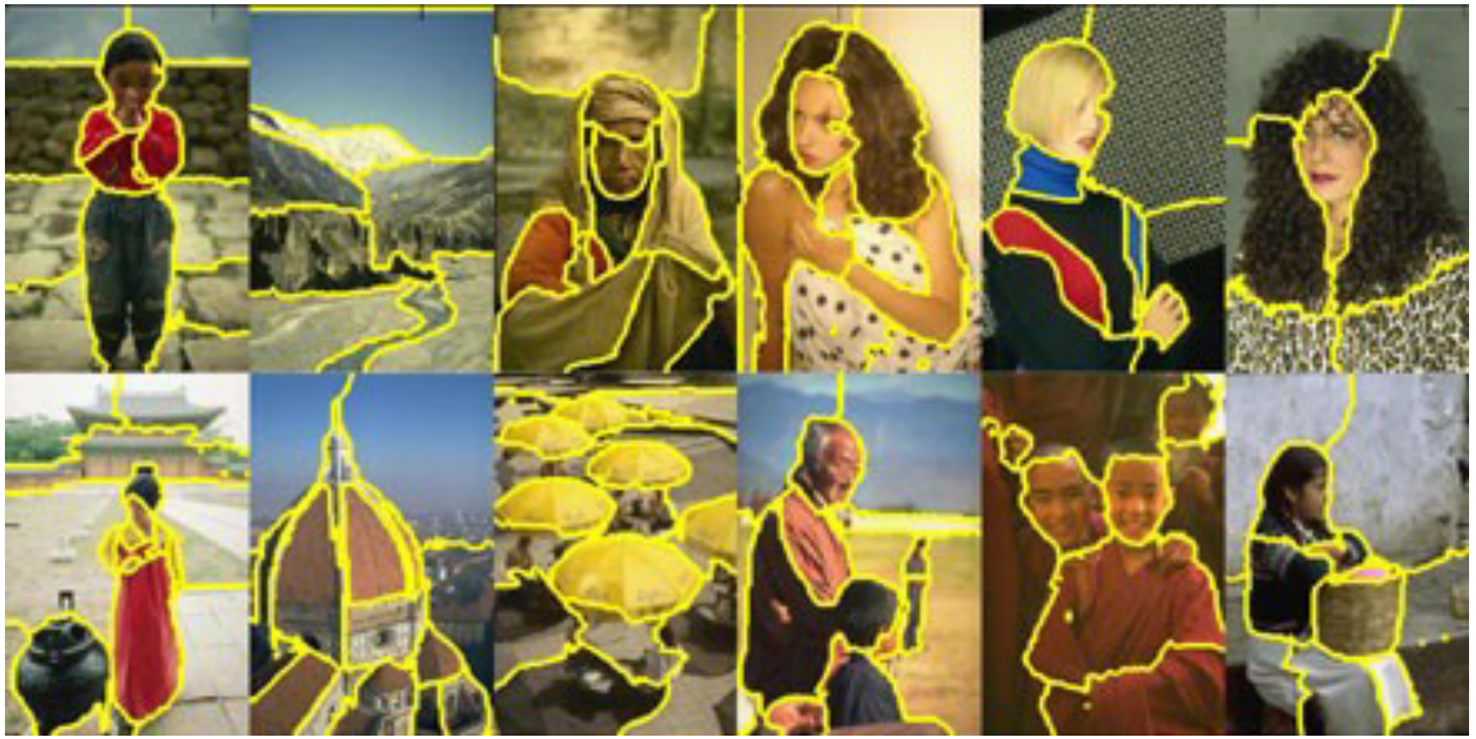


Image segmentation

Image as a graph: treat segmentation as clustering

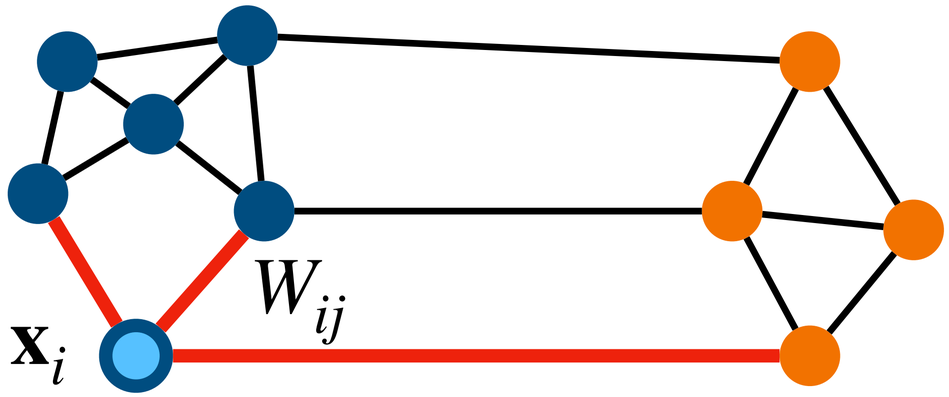
$$W_{ij} = f(\text{pixel separation, pixel colour})$$



Graph Terminologies

Node degree:

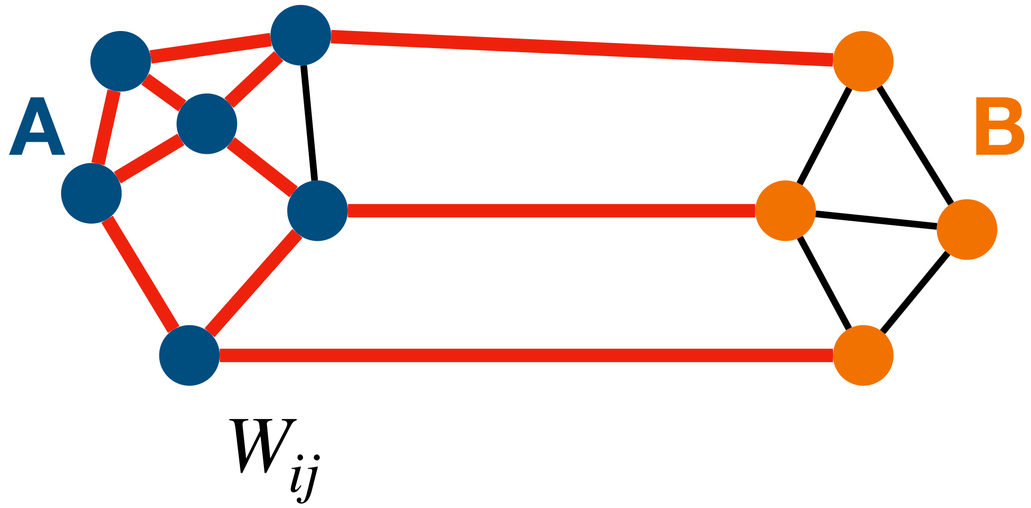
$$d_i = \sum_j W_{ij}$$



Volume of a set:

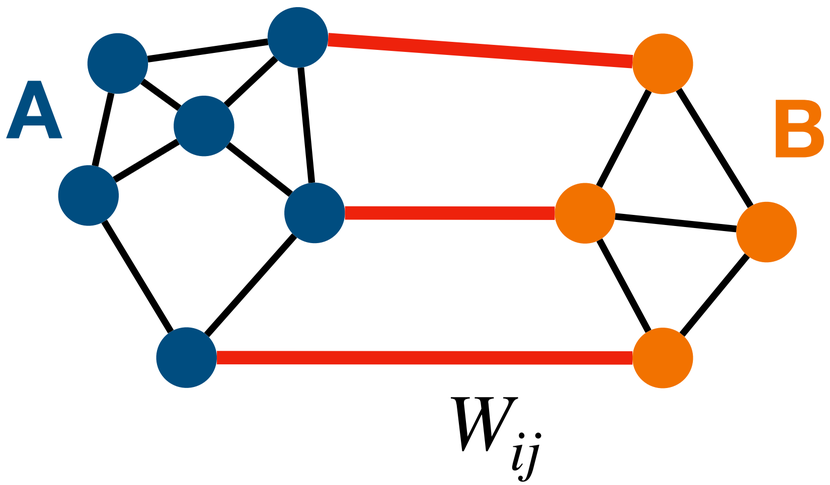
$$\mathrm{vol}(A) = \sum_{i \in A} d_i$$

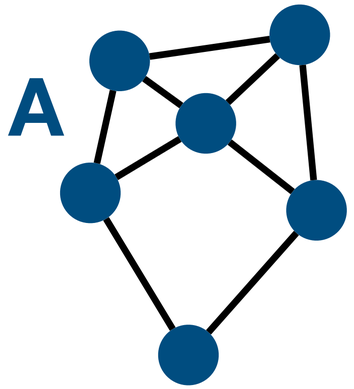
$$= \sum_{i \in A} \sum_j W_{ij}$$

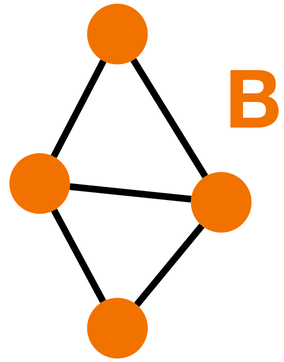


Cutting the graph

Remove connections so that the graph is now two disjoint sets:







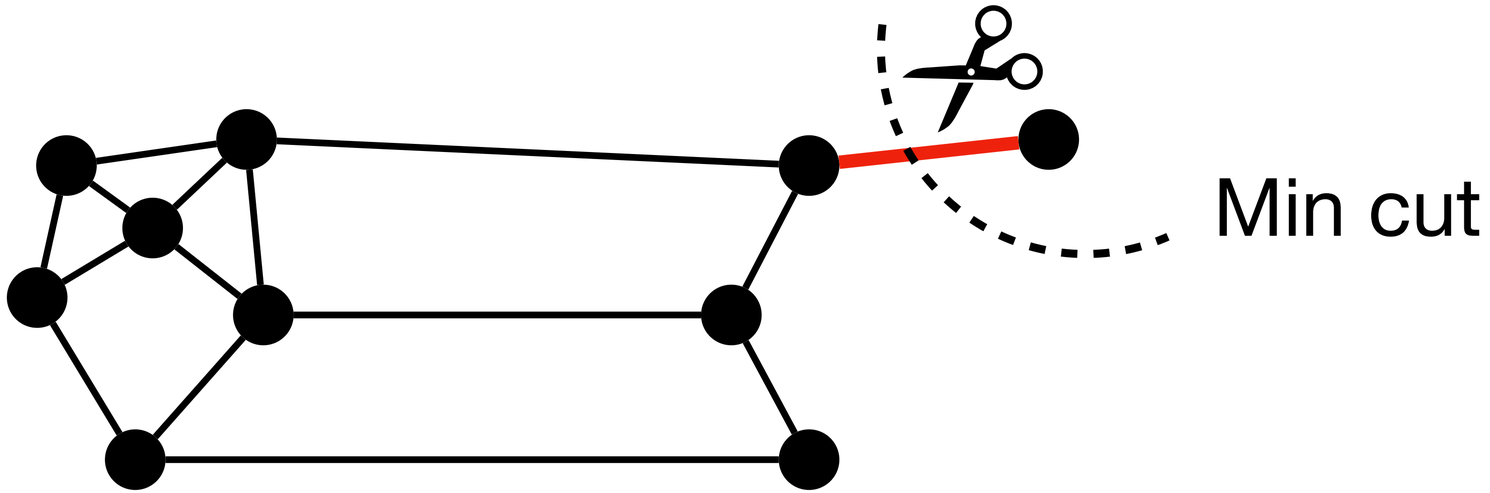
What will $\text{cut}(A, B)$ be?

Graph cut: sum of the connections (edges) that connect the groups.

$$\text{cut}(A, B) = \sum_{i \in A} \sum_{j \in B} W_{ij}$$

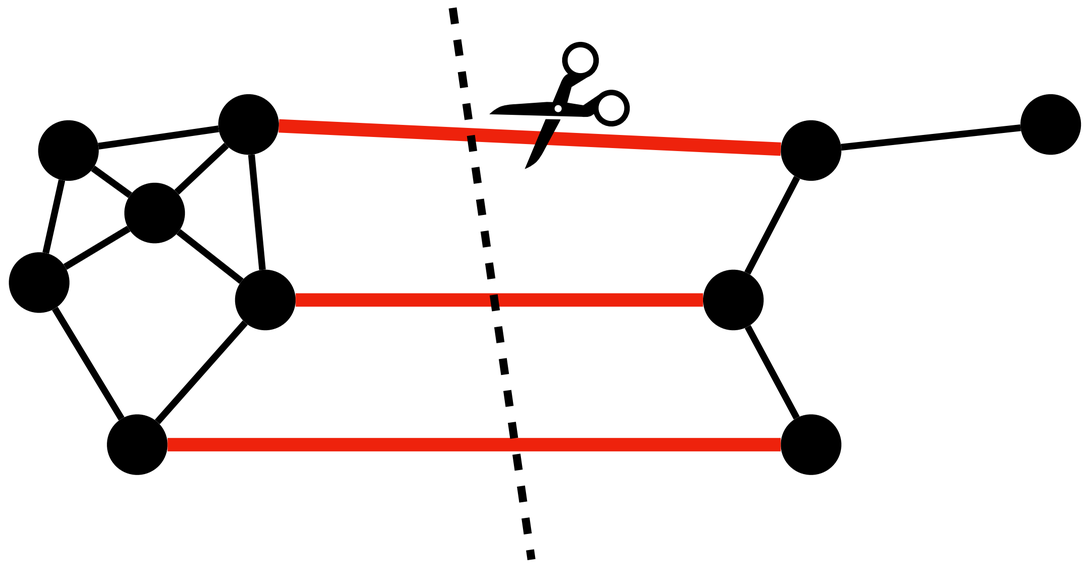
The minimum cut is not always the best

The minimum cut can be biased by outliers with fewer connections:



The minimum cut is not always the best

The minimum cut can be biased by outliers with fewer connections:



Min Normalised cut

Solution: **Normalise** the cut!

The Normalised Cut

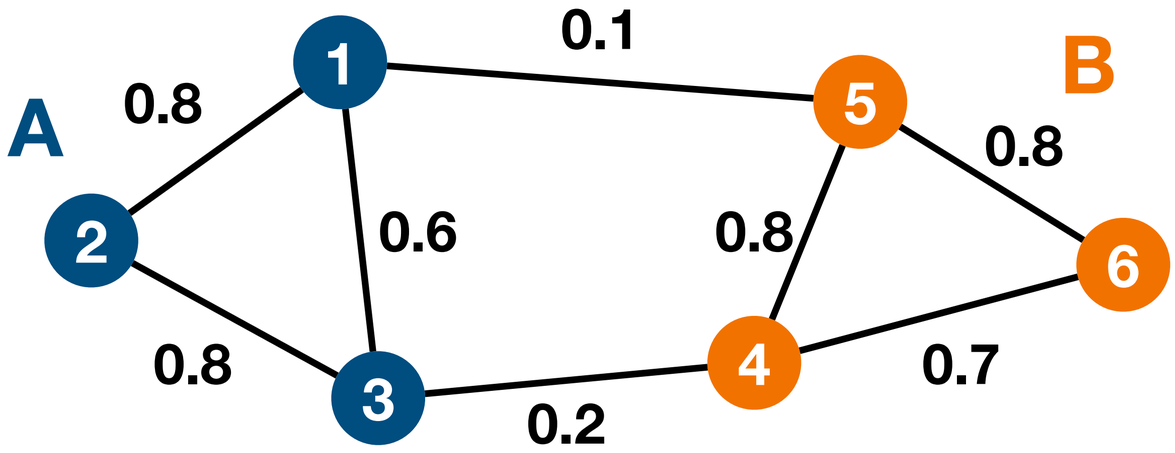
Consider the **connectivity** of two groups relative to the **volumes** of each group:

$$\text{Ncut}(A, B) = \frac{\text{Cut}(A, B)}{\text{Vol}(A)} + \frac{\text{Cut}(A, B)}{\text{Vol}(B)}$$

$$\text{Ncut}(A, B) = \text{Cut}(A, B)\left(\frac{\text{Vol}(A) + \text{Vol}(B)}{\text{Vol}(A)\text{Vol}(B)}\right)$$

Exercise

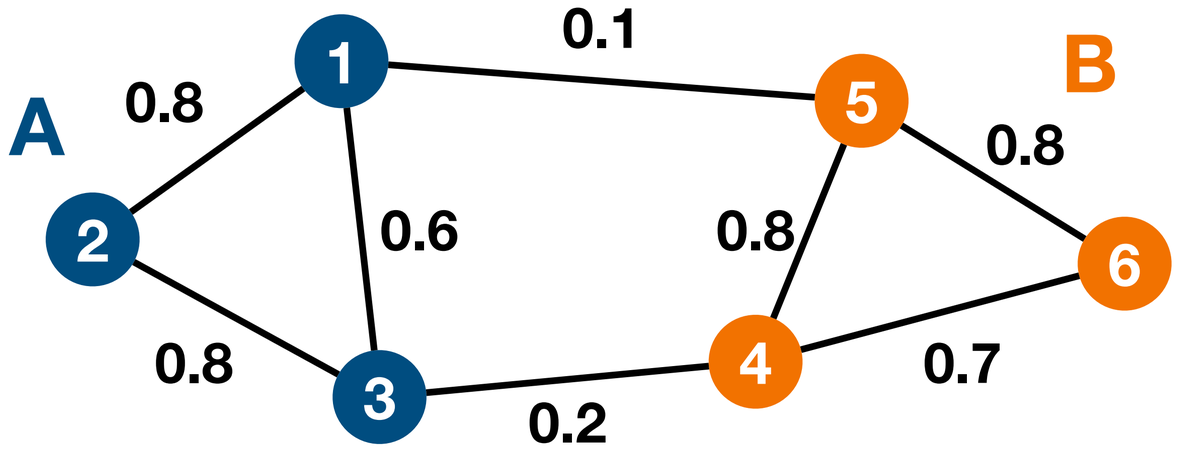
Consider cutting the graph into two parts A & B:



Calculate the node degrees d_i, cut(A,B) and Ncut(A,B).

Exercise

Consider cutting the graph into two parts A & B:



$$d_1 = 0.8 + 0.6 + 0.1 = 1.5$$
$$d_2 = 0.8 + 0.8 = 1.6$$
$$d_3 = 0.6 + 0.8 + 0.2 = 1.6$$
$$d_4 = 0.2 + 0.8 + 0.7 = 1.7$$
$$d_5 = 0.1 + 0.8 + 0.8 = 1.7$$
$$d_6 = 0.8 + 0.7 = 1.5$$

$$\text{vol}(A) = d_1 + d_2 + d_3 = 4.7$$

$$\text{vol}(B) = d_4 + d_5 + d_6 = 4.9$$

$$\text{cut}(A, B) = 0.1 + 0.2 = 0.3$$

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, B)}{\text{vol}(A)} + \frac{\text{cut}(A, B)}{\text{vol}(B)} = \frac{0.3}{4.7} + \frac{0.3}{4.9} = 0.1251$$

Finding A and B to minimise the normalised cut

Shi and Malik (2000) show that

$$\min_{\mathbf{x}} \mathrm{Ncut}(\mathbf{x}) = \min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}}$$

Which can be minimised by solving the generalise eigenvalue system

$$(\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}$$

$\mathbf{x}$ and $\mathbf{y}$ are vectors indicate which cluster a data point is in.

Degree matrix $\mathbf{D}$:

$$D_{ii} = d_i = \sum_j W_{ij}$$

The **second lowest eigenvalue** (and eigenvector) correspond to the cut needed to split the graph into **two** (bipartitioning).

Spectral Clustering Ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: non-linear separation between clusters
  - **Parameters**: (generalised eigenvectors (cluster assignment vector)
  - **Hyper-parameters**: number of clusters, kernel length scale
- **Evaluation metric**: normalised graph cut
- **Optimisation**: Eigen-decomposition (and expectation maximisation in k-means)

Take home messages

- Clustering - group based on **similarity** not labels.
- K-means - represent clusters by the **mean** of the cluster, minimise the reconstruction error.
- Spectral clustering - cluster via **cutting graph representation** of the data.
- Normalised cut - **connectivity** between groups relative to their **volume**.
- The **second lowest eigenvalue** is the Fiedler value - used for splitting the graph into to **two groups**.

Generative Models

Session outline

Unsupervised learning

- Recap PCA and Auto-encoders
- Clustering: K-Means
- Clustering: Spectral Clustering

Generative models

- Review of Bayes' theorem and Bayesian inference

Recap: Probabilistic Classifiers

Determine a class from some features $\rightarrow P(y \mid \mathbf{x})$

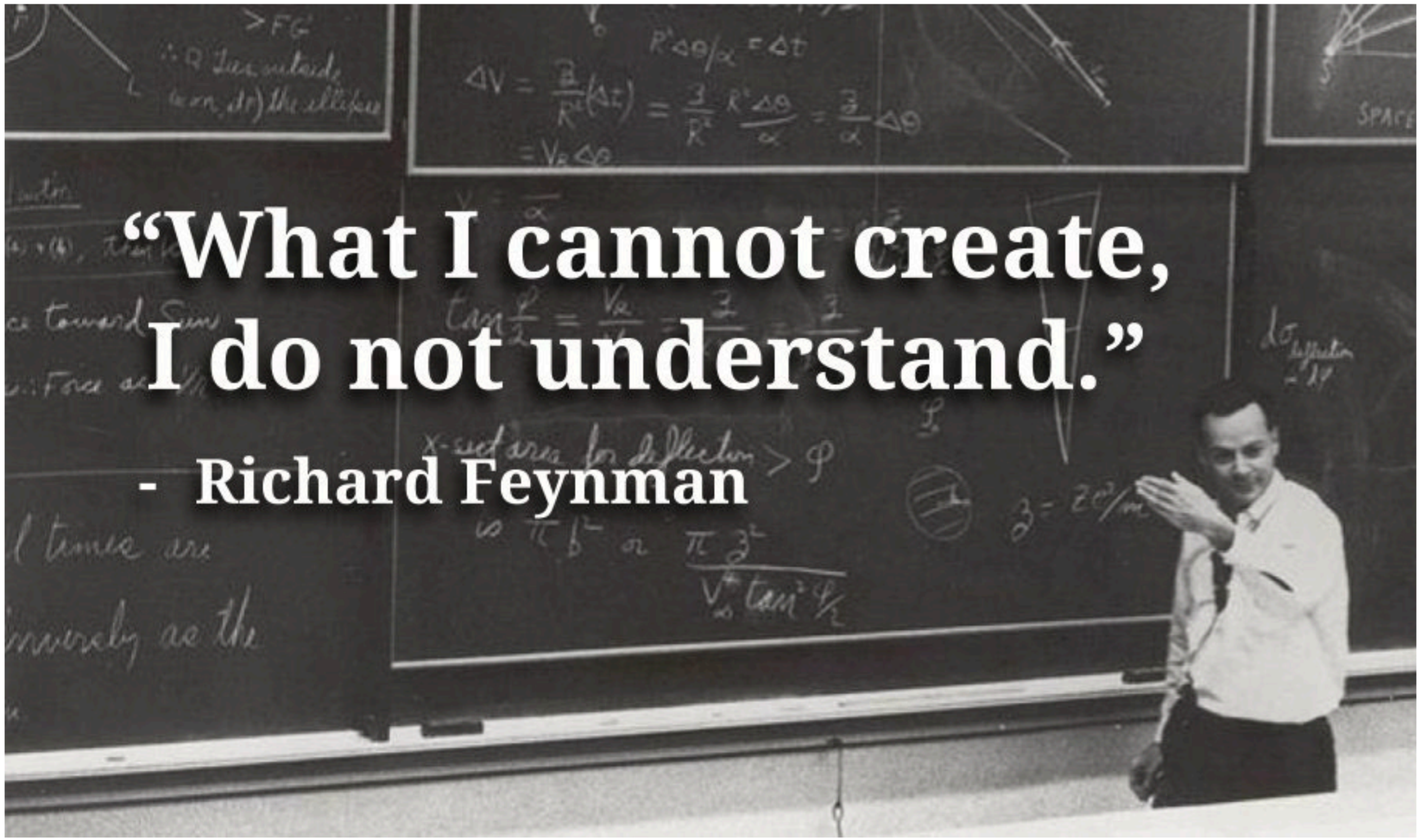
Discriminative

- Optimise an **explicitly defined** form of the decision boundary.
- Assume some functional form for $P(y \mid \mathbf{x})$.
- Estimate parameters for this based on training data.

Generative

- Define a **process** that could have generated the observations.
- Assume some functional form for $P(\mathbf{x} \mid y)$ and $P(\mathbf{x})$.
- Estimate parameters for these from training data.
- Use Bayes' rule to find $P(y \mid \mathbf{x})$.

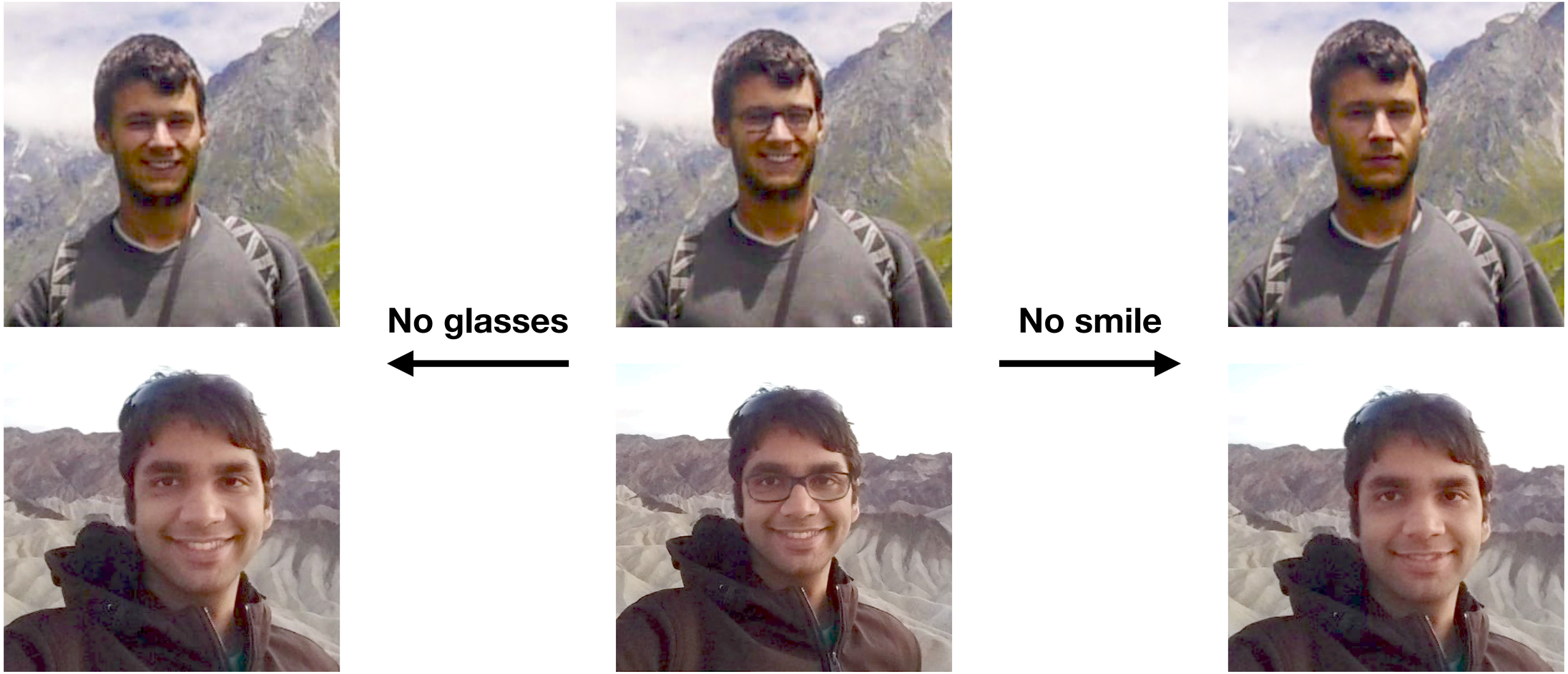
Generative models



Holy grail of ML: understand data → create data

Which image is real?

Which image is real? - Neither!



Generative models

Variational Auto-Encoders (VAEs)

Generating faces

https://www.youtube.com/watch?v=XNZIN7Jh3Sg

Digital Generative Art

https://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/

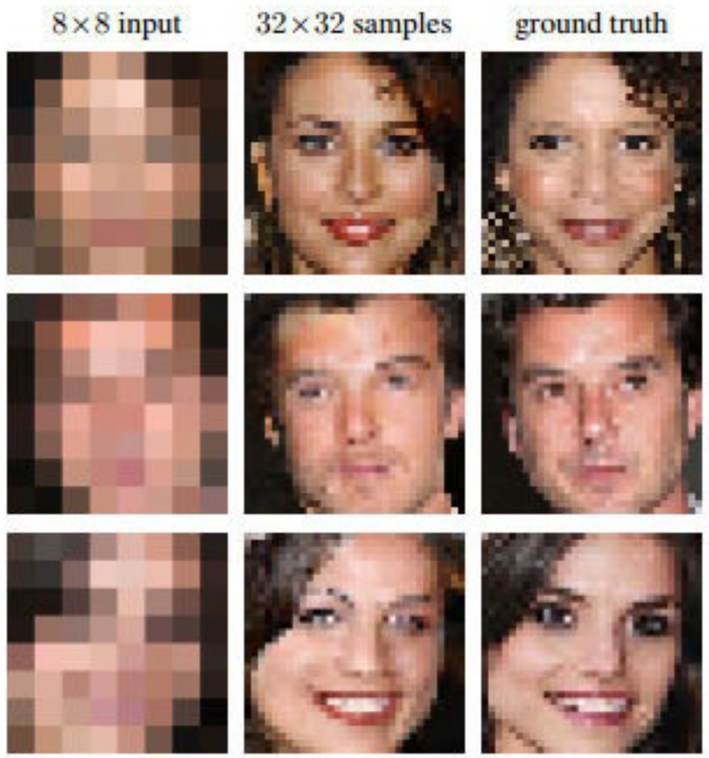
Generative Adversarial Networks

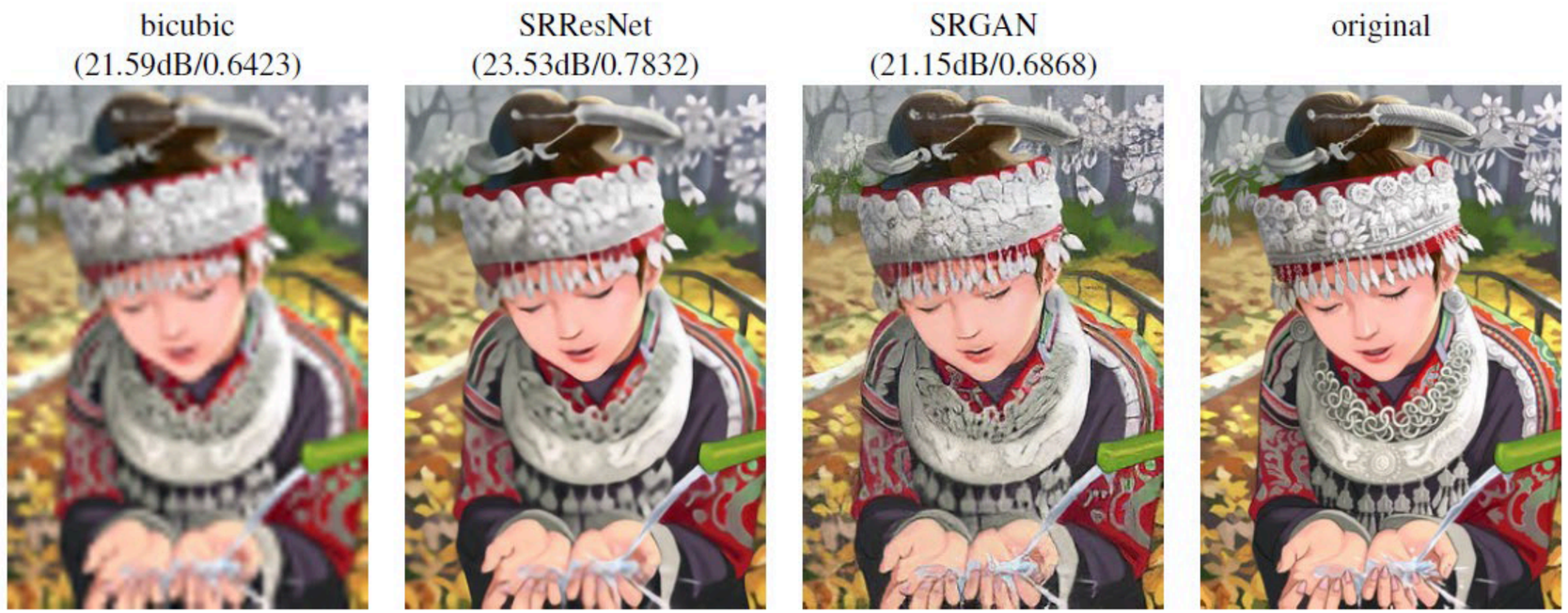
Generating images

https://www.youtube.com/watch?v=XOxxPcy5Gr4

Image upscaling

Conditional generative model: $P(\text{high res image} \mid \text{low res image})$





Ledig et al. 2017

Style transfer or colorisation

Conditional generative model: $P(\text{zebra image} \mid \text{horse image})$

Zhu et al. (2017)

Session outline

Unsupervised learning

- Recap PCA and Auto-encoders
- Clustering: K-Means
- Clustering: Spectral Clustering

Generative models

- **Review of Bayes' theorem and Bayesian inference**

Bayes' rule

Given data $\mathbf{x}$ and parameters $\boldsymbol{\theta}$:

Likelihood Prior

Posterior $$P(\boldsymbol{\theta} \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \boldsymbol{\theta}) P(\boldsymbol{\theta})}{P(\mathbf{x})}$$ Evidence

Prior: probability of the model **before** observing the data (evidence).

Posterior: probability of the model **after** observing the data (evidence).

Likelihood: probability of **observing** a (random) data point **given** the model (**compatibility** of the data with the given model).

Evidence or marginal likelihood: probability of **observing** a (random) data point across **all model variations.**

Principles of Bayesian inference

Formulation of a generative model

Likelihood $P(\mathbf{x} \mid \boldsymbol{\theta})$

Prior $P(\boldsymbol{\theta})$

Observation of the data

$\mathbf{x}_n$

Update of beliefs based on observations, given a prior state of knowledge

$$P(\boldsymbol{\theta} \mid \mathbf{x}_n) \propto P(\mathbf{x}_n \mid \boldsymbol{\theta}) P(\boldsymbol{\theta})$$

Bayesian Linear Regression

Recall that in linear regression we assumed:

$$y_n = \mathbf{w}^T \mathbf{x}_n + e_n \qquad \rightarrow P(y_n \,|\, \mathbf{x}_n, \mathbf{w}, \sigma) = \text{Gaussian}(\mathbf{w}^{\text{T}}\mathbf{x}_{\text{n}}, \sigma)$$

We can also assume some prior distribution of the parameters ($\mathbf{w}$) representing our current belief:

$$P(\mathbf{w}) = \text{Gaussian}(\bar{\mathbf{w}}, \sigma_{\text{w}})$$

Initial parameters:

$\bar{\mathbf{w}}$ = mean weight vector,

σ_w = uncertainty in the weights

Using this we can apply Bayes' rule to find the **posterior** (new distribution for the weights). Other distributions can be used, some form conjugate pairs.

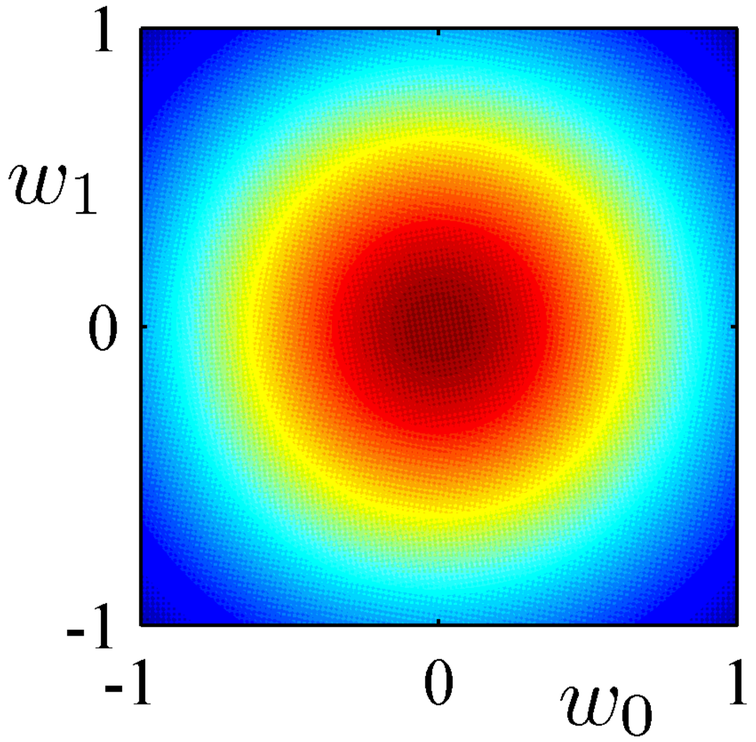
Maximum a posteriori (MAP) estimate: mode of posterior distribution instead of the likelihood (MLE).

Bayesian linear regression: example

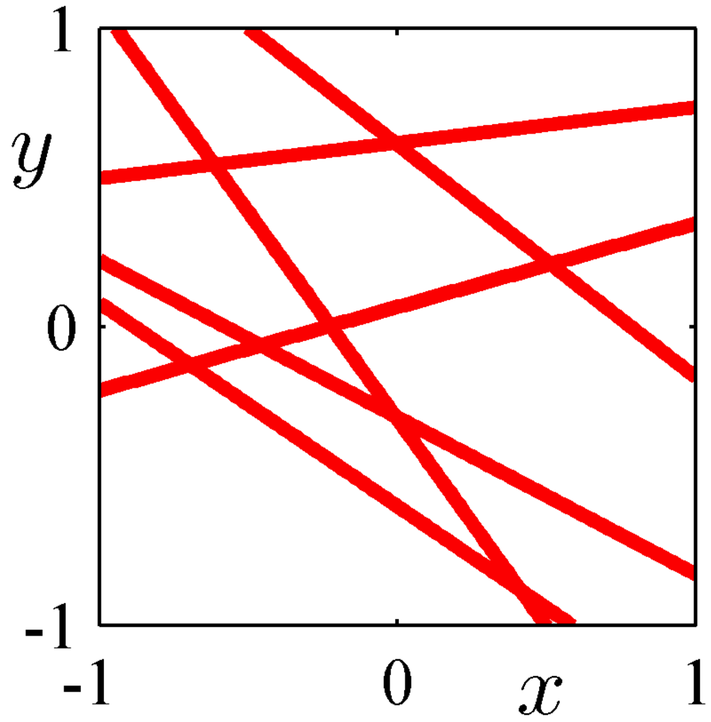
x, y are 1D so use $\mathbf{w} = (w_0, w_1)^T$

Estimate w_0, w_1 using Bayesian inference.

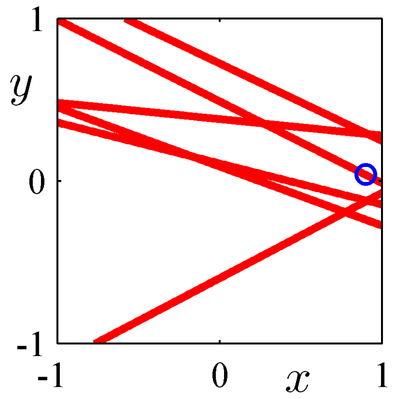
$$P(\mathbf{w}) = \text{Gaussian}(\bar{\mathbf{w}}, \sigma_{\text{w}})$$



6 random samples of the weights

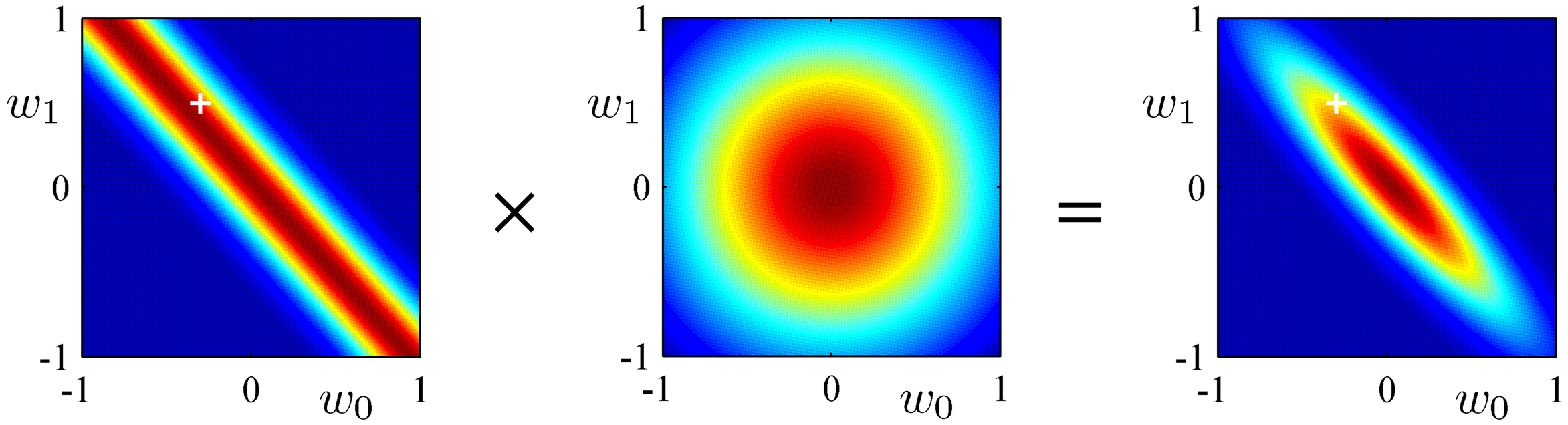


Bayesian linear regression: example



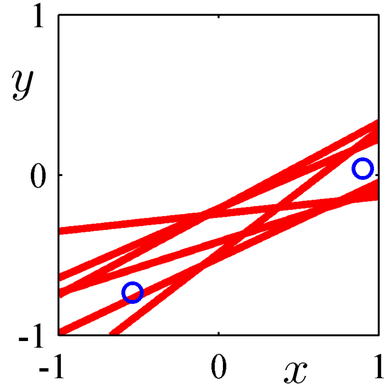
First data point $x_1 = 0.9$ and $y_1 = 0.1$

$$P(y_1 = 0.1 \,|\, x_1 = 0.9, \mathbf{w}, \sigma)$$



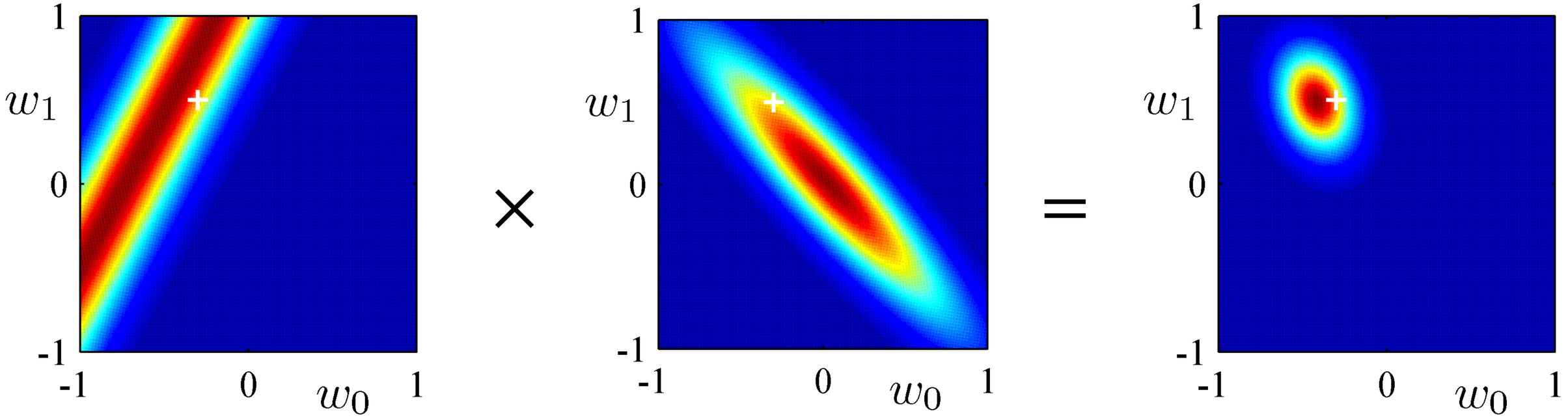
Likelihood **Prior** **Posterior**

Bayesian linear regression: example



Second data point $x_2 = -0.7$ and $y_2 = -0.8$

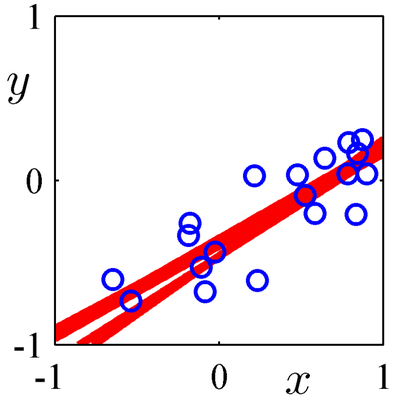
$$P(y_2 = -0.8 \,|\, x_2 = -0.7, \mathbf{w}, \sigma)$$



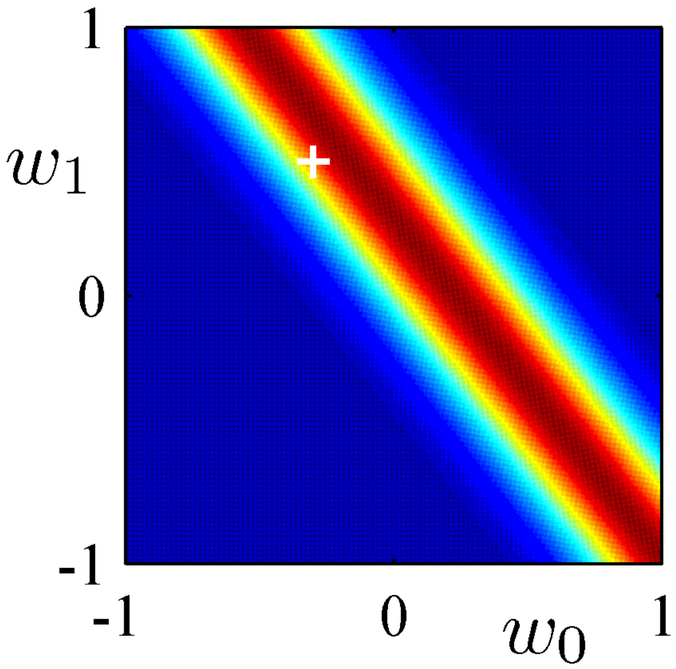
Likelihood **Prior** **Posterior**

Bayesian linear regression: example

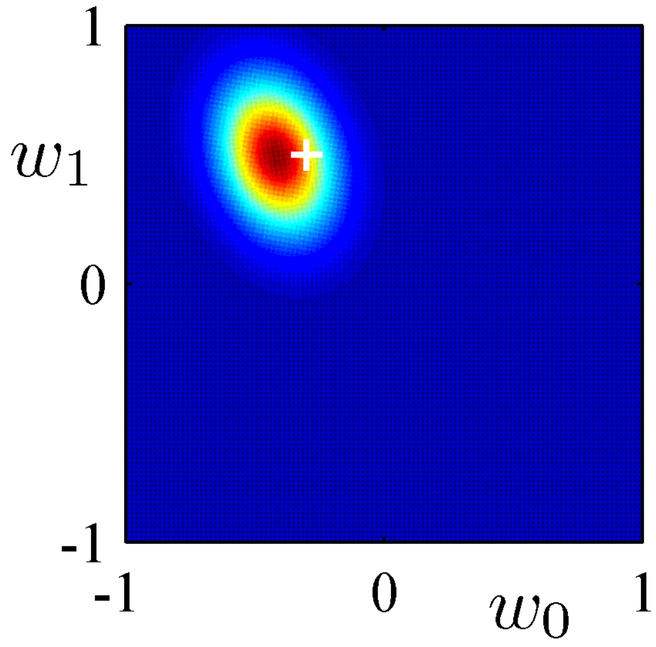
20 data points



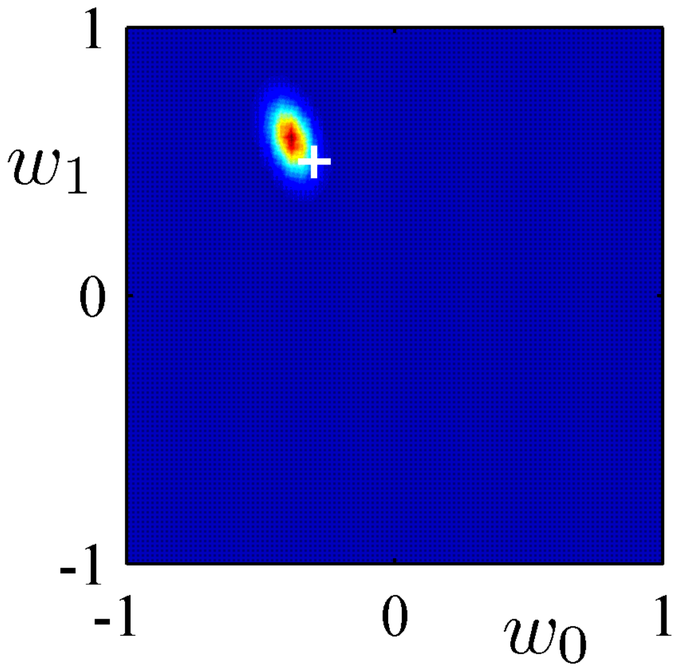
$P(y_{20} \mid x_{20}, \mathbf{w}, \sigma)$



$\times$



$=$



Likelihood **Prior** **Posterior**

What is the key assumption here?

Conditional independence of the data - **Naive Bayes**

The likelihood for the data points do **not** depend on the previous samples:

$$P(\mathbf{x}_n \,|\, \mathbf{x}_{n-1}, \ldots, \mathbf{x}_1, \mathbf{w}) = P(\mathbf{x}_n \,|\, \mathbf{w})$$

Naive Bayes

$$P(\mathbf{w} \,|\, \mathbf{x}_n, \ldots, \mathbf{x}_1,) = \left(\prod_{n=1}^{N} P(\mathbf{x}_n \,|\, \mathbf{w}) \right) \frac{P(\mathbf{w})}{P(\mathbf{x}_n, \ldots, \mathbf{x}_1)}$$

Bayesian Regression Ingredients

- **Data** + pre-processing
- **Model**:
  - **Structure/architecture**: likelihood and prior distributions, (basis functions)
  - **Parameters**: weights and biases
  - **Hyper-parameters**: distribution parameters (e.g variance) + (any for basis)
- **Evaluation metric**: mean squared error
- **Optimisation**: Closed form for certain distributions, gradient descent or similar otherwise.

Pros and Cons of Bayesian Methods

Pros:

- Provide **uncertainty estimation**, i.e predicting the output distribution with mean and (co)**variance.**
- Make use of more information (**prior**, if available)
- Less overfitting (in general)

Cons:

- Complexity
- Subjectivity: all inferences are based on beliefs (**assumptions**). Which prior do we choose? What if it is wrong?

Take home messages

- Generative models aim to create **generating** functions or distributions to model data.
- Bayes' theorem: Posterior $\propto$ Likelihood x Prior
- Bayesian regression: assume some **distribution** for the **parameters** and **data**, update this distribution using Bayes' theorem.
- Naive Bayes': data samples are **conditionally independent**

Further reading

A First Course in Machine Learning by Rogers and Girolami:

- K-means clustering: Chapter 6 (section 6.2)
- Bayesian regression: Chapter 3 (section 3.1, 3.2, 3.3)

Spectral clustering:

- Normalised Cuts and Image Segmentation by Shi and Malik (2000)
- (Paper uploaded on BB, you don’t need to read in detail but it should give some context to the topic).