

MATH96007 - MATH97019 - MATH97097
Methods for Data Science

Prof Mauricio Barahona

WHAT IS DATA SCIENCE?

Data science, also known as data-driven science, is an interdisciplinary field about scientific methods, processes, and systems to extract knowledge or insights from data in various forms, either structured or unstructured.

Data science is a “concept to unify statistics, data analysis and their related methods” in order to “understand and analyze actual phenomena” with data. It employs techniques and theories drawn from many fields within the broad areas of mathematics, statistics, information science, and computer science, in particular from the subdomains of machine learning, classification, cluster analysis, data mining, databases, and visualization.

“The ability to take data to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it's going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids. Because now we really do have essentially free and ubiquitous data.” Hal Varian

“The ability to take data to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it's going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids. Because now we really do have essentially free and ubiquitous data.” Hal Varian

Google's Chief Economist

THERE IS A LOT OF HYPE



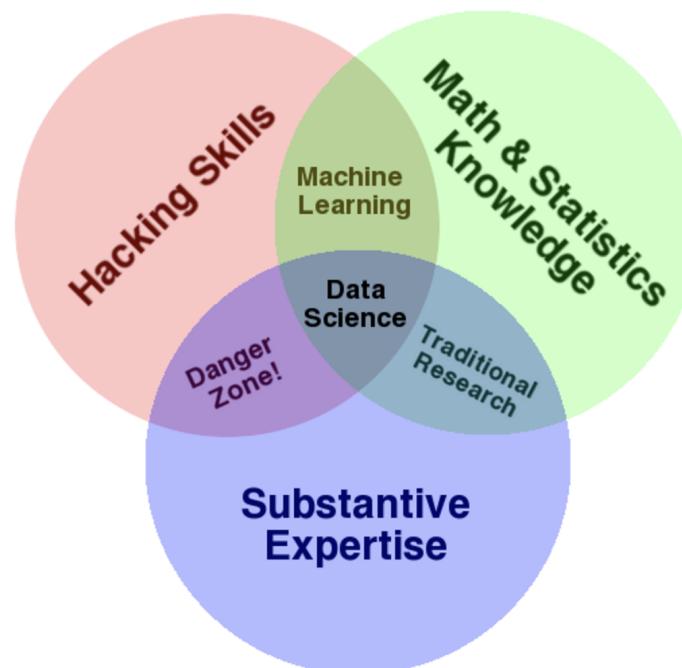
TRUTH DESPITE THE HYPE

I think there *is* something new in “data science”. It’s not a science. It’s more of a process; a merging of (1) skills in applied mathematics, computer science, statistics, visualisation and communication, with (2) rich datasets and domain knowledge.

In practice, it’s usually done in teams. No one has deep knowledge of statistics, mathematics, computer science, visualisation together with detailed knowledge of the data source, industry, interesting questions.

New aspects include: massive amounts of data (even “big data”), combined with cheap computing power.

VENN DIAGRAM: WHERE DOES DATA SCIENCE FIT



<http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram>

THE TITLE “DATA SCIENTIST”

There's no academic field called “data science”. After all, scientists have always used data. Statisticians theorise about data, and analyse it too. Very few academics are “professors of data science” .

Cathy O'Neil's book “Doing Data Science” says that an academic data scientist might be defined as: “a scientist, trained in anything from social science to biology, who works with large amounts of data, and must grapple with computational problems posed by the structure, size, messiness, and the complexity and nature of the data, while simultaneously solving a real-world problem.”

O'Neil, Cathy; Schutt, Rachel. Doing Data Science: Straight Talk from the Frontline (Kindle Locations 480-482). O'Reilly Media.

DATA SCIENCE JOBS

There are a lot of jobs as data scientists. What do these people do?

Fundamentally, a data scientist is someone who can extract meaning from data and communicate the results clearly.

Data science a company typically includes:

- ▶ data collection, storage and management
- ▶ who has access to what data
- ▶ how will the data be used, how does it add value
- ▶ privacy considerations
- ▶ * analysis of data
- ▶ * communicating the results

CASE STUDIES: SUCCESSES OF DATA SCIENCE

Moneyball! Data in baseball outperforms standard predictors of who will be successful. There's a book, and a movie with Brad Pitt.

Other big success areas:

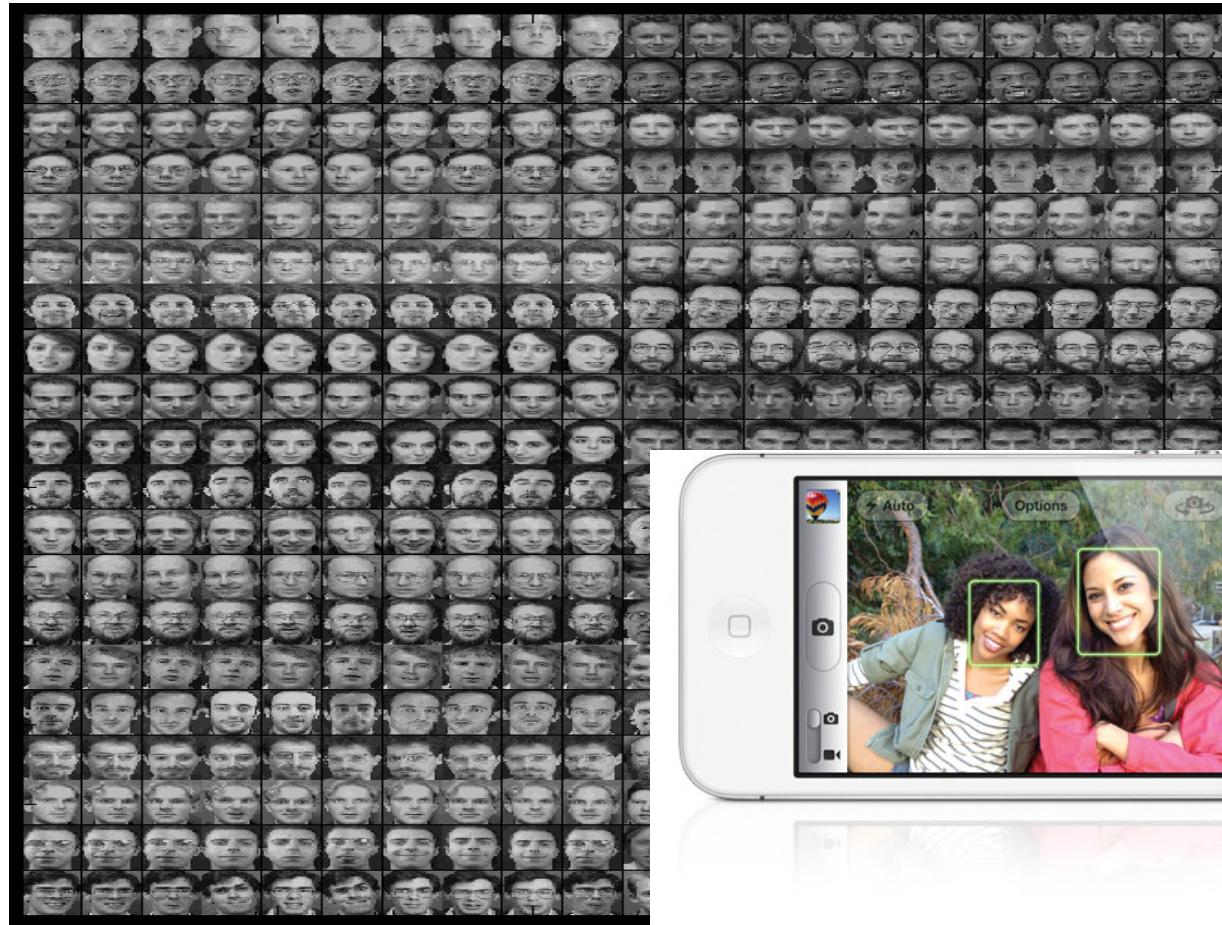
- ▶ Spam filters
- ▶ Fraud detection
- ▶ Election predictions (Nate Silver,
<https://fivethirtyeight.com>)
- ▶ Predictions of crime hotspots
- ▶ Text analysis: twitter, blogs
- ▶ Targeted advertising
- ▶ Song, movie and product recommendations

CASE STUDIES: SUCCESSES OF DATA SCIENCE

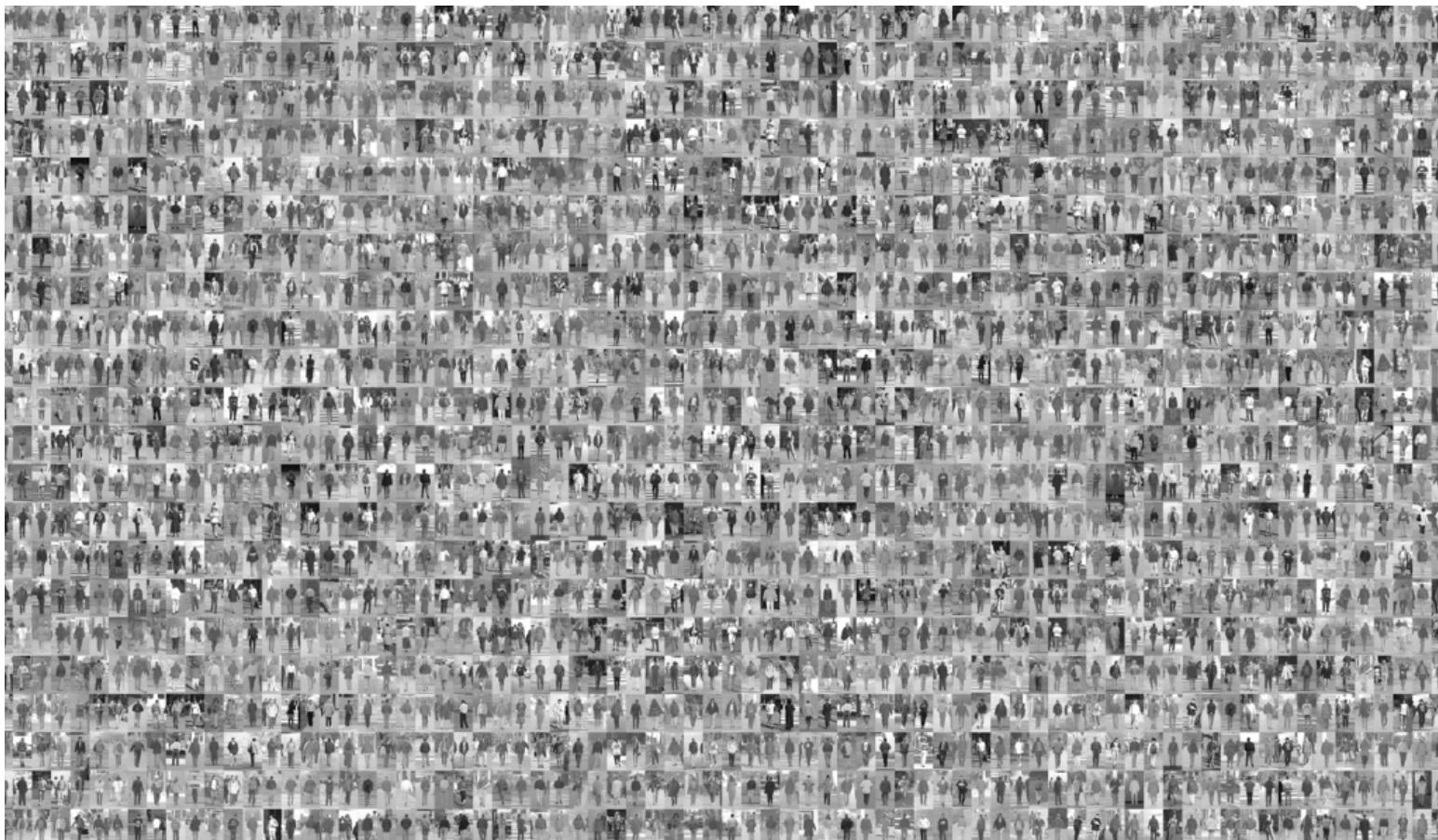
- ❑ Forecasting (*e.g. Energy demand prediction, finance*)
- ❑ Imputing missing data (*e.g. Netflix recommendations*)
- ❑ Detecting anomalies (*e.g. Security, fraud, virus mutations*)
- ❑ Classifying (*e.g. Credit risk assessment, cancer diagnosis*)
- ❑ Ranking (*e.g. Google search, personalization*)
- ❑ Summarizing (*e.g. News zeitgeist, social media sentiment*)
- ❑ Decision making (*e.g. AI, robotics, compiler tuning, trading*)



CASE STUDIES: SUCCESSES OF DATA SCIENCE

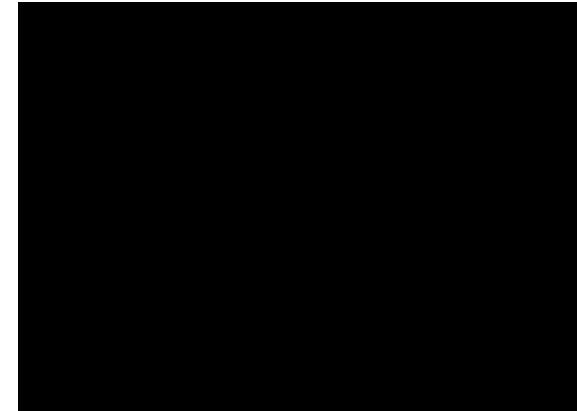


CASE STUDIES: SUCCESSES OF DATA SCIENCE



Millions of labeled examples are used to build real-world applications, such as pedestrian detection

CASE STUDIES: SUCCESSES OF DATA SCIENCE



CASE STUDIES: SUCCESSES OF DATA SCIENCE



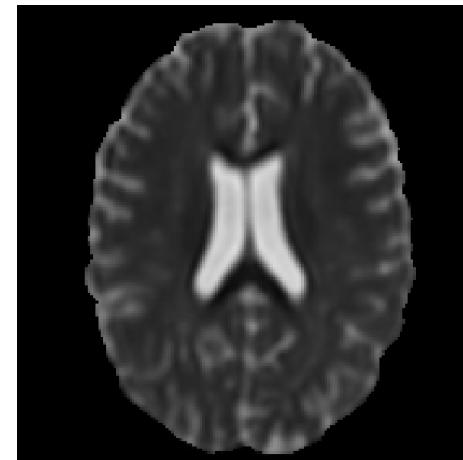
Real nMTR image



Synthetic nMTR image using L1 loss



Real RD image



Synthetic RD image

CASE STUDIES: SUCCESSES OF DATA SCIENCE



my reading was similar to everyones. she told me she was going to take her time and not rush me out of there. i was there not even 8 minutes she told me i was pregnant then she changed her mind and said i had a miscarriage. im 17 years old i told her she was wrong she then went on and said "i see you and your brother fight alot just know he loves you" i dont even have a brother.

she then told my friend she was going to get stabbed

Was this review helpful? Yes 2
Ask taydube about Fatima's Psychic Studio

[Problem with this review?](#)

Paul Bettany did a great role as the tortured father whose favorite little girl dies tragically of disease.

Natural Language Processing

CASE STUDIES: SUCCESSES OF DATA SCIENCE

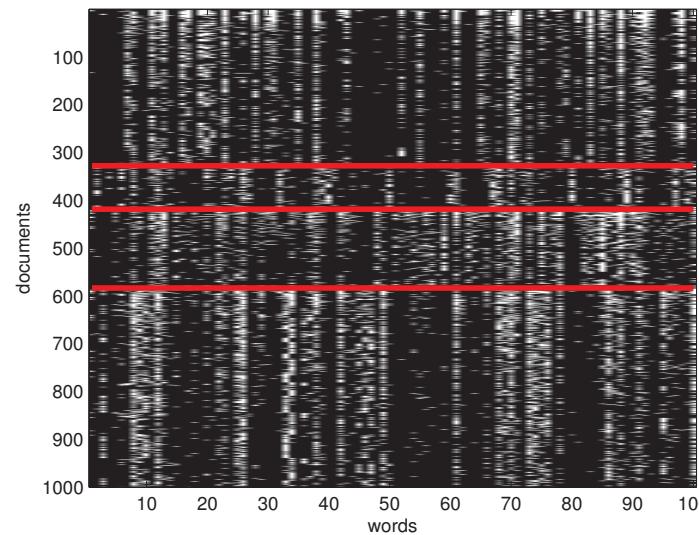


Figure 1.2 Subset of size 16242×100 of the 20-newsgroups data. We only show 1000 rows, for clarity. Each row is a document (represented as a bag-of-words bit vector), each column is a word. The red lines separate the 4 classes, which are (in descending order) comp, rec, sci, talk (these are the titles of USENET groups). We can see that there are subsets of words whose presence or absence is indicative of the class. The data is available from <http://cs.nyu.edu/~roweis/data.html>. Figure generated by newsgroupsVisualize.

Natural Language Processing

CASE STUDIES: SUCCESSES OF DATA SCIENCE



Speech recognition - Translation

In all these examples, one of the key steps was the mathematical conceptualisation/formulation

SPECIFIC QUESTIONS THAT YOU COULD ASK

- ▶ Predict whether a heart attack patient will have a second heart attack, using demographic, diet and clinical data
- ▶ Predict the price of a stock in 6 months, using company performance measures and economic data. (Don't invest your money by your answers...)
- ▶ Identify the numbers in a handwritten ZIP code, from a digital image.
- ▶ Estimate the amount of glucose in the blood of a diabetic person from the infrared absorption spectrum of that persons blood.
- ▶ Identify the risk factors for prostate cancer, based on clinical and demographic data
- ▶ Predict which flu strain will be successful next year based on its DNA sequence and relationships to other flu strains

THE PROCESS OF DATA SCIENCE

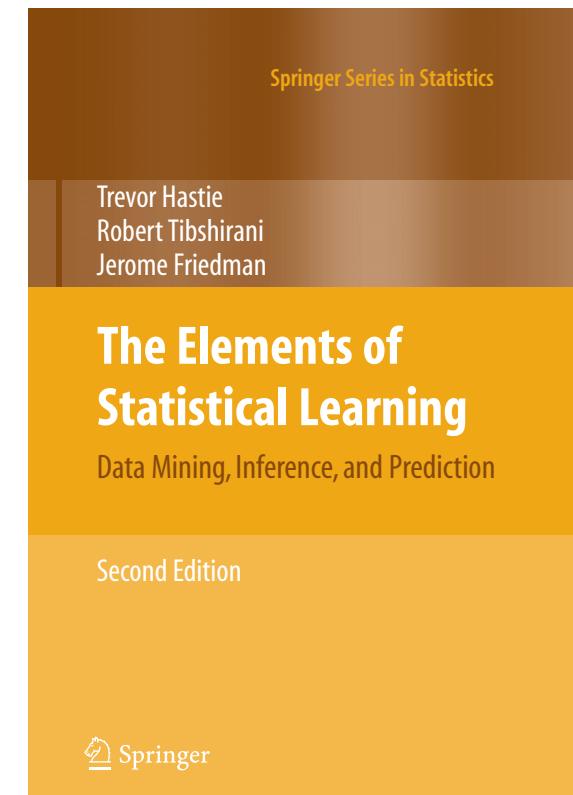
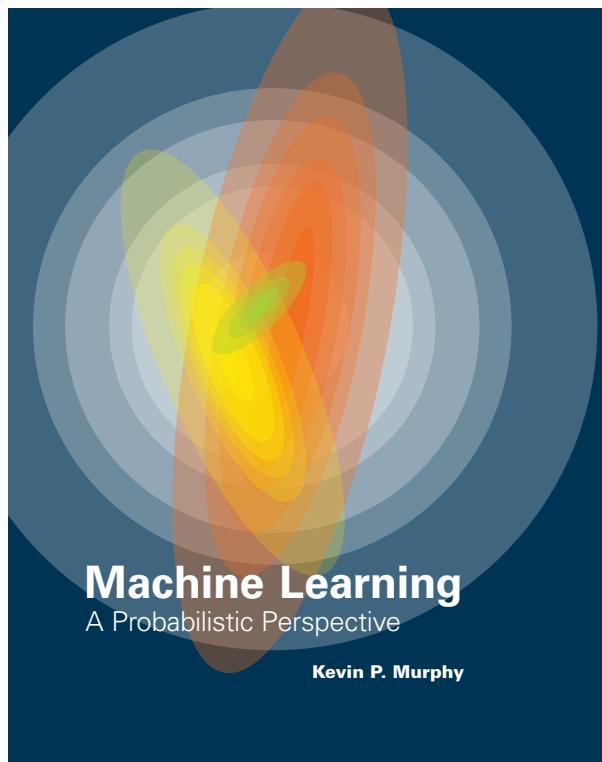
- ▶ The science: what's a good question and what's the dataset you will use to answer it?
- ▶ What is the input and what are you trying to determine
- ▶ Data handling, cleaning and exploration: computer programming and knowing things
- ▶ THE MATH and the main analysis:
 - ▶ statistics, computing: what do you compute, why, and what does it mean?
 - ▶ machine learning: supervised, unsupervised..
 - ▶ networks: characteristics, analysis, comparisons
- ▶ Communication: words, visualisations, summaries to tell the story

Aims and outcomes

- Learn mathematical concepts underpinning methods in learning from data
- The process of conceptualisation of the analysis
- The process of explanation of the analysis
- The mathematical justification of the analysis
- Getting a good exposure to current methods and their use in practice
- Challenges in dealing with (realistic) data

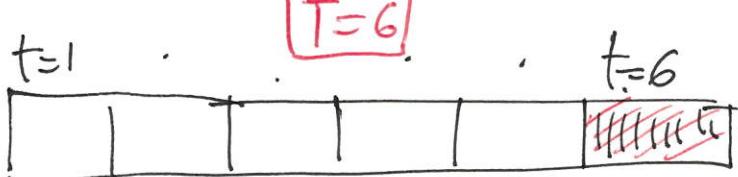
What the tools are, how they work mathematically, and how to analyse a dataset and clearly communicate the results

Lots of resources online.
Use them **but** make things your own
and understand the principles



k -NN : k -nearest neighbours
as a predictor.

T -fold cross validation



Split into T -folds

All samples $S = \{\vec{x}^{(i)}, y^{(i)}\} \quad i=1, \dots, N$

$$S = \bigcup_{t=1}^T S_t \quad |S_t| = \frac{|S|}{T}$$

$$\bar{S}_t = S - S_t$$

Equal size, properly sampled

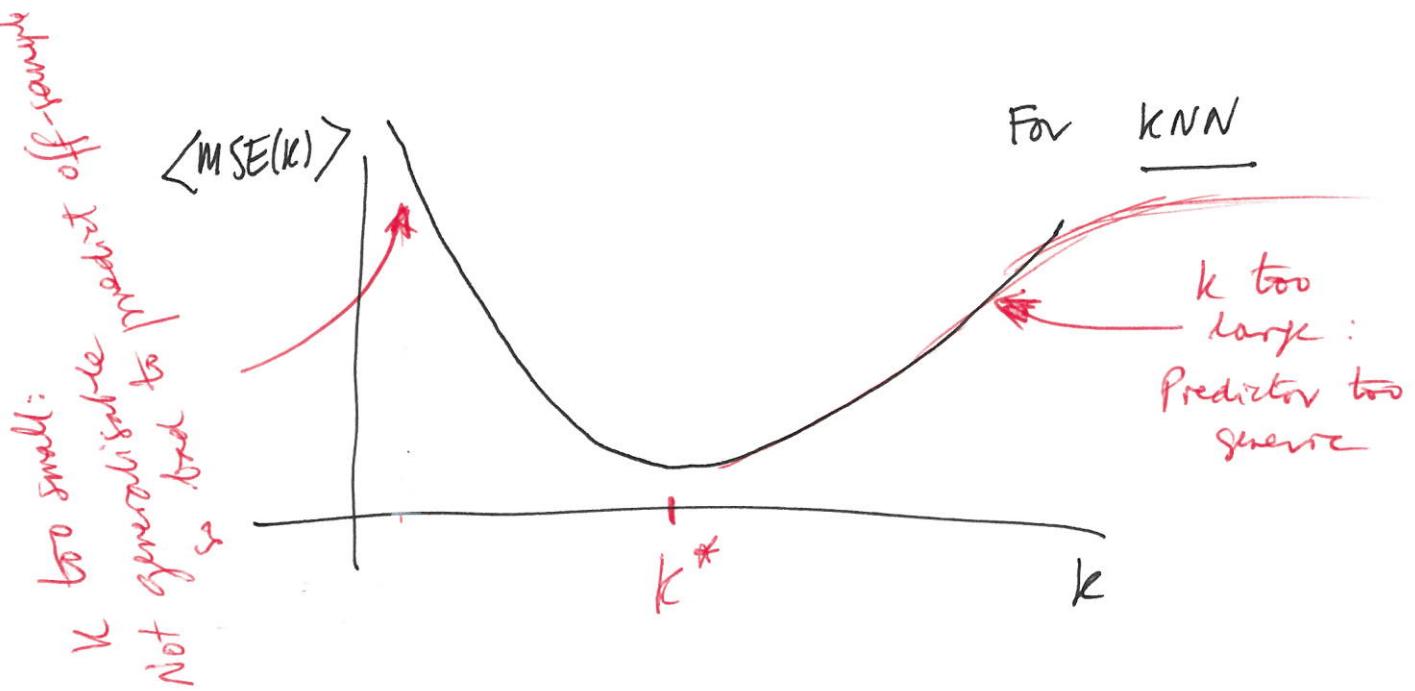
Different model for each \bar{S}_t $\rightarrow \hat{f}_{\bar{S}_t; k} \quad t=1, \dots, T$

Predict \hat{f}_t $\Rightarrow \sum_{i \in \bar{S}_t} \left[\hat{f}_{\bar{S}_t}(\vec{x}^{(i)}) - y^{(i)} \right]^2 = MSE_t$

$$\langle MSE(k) \rangle = \frac{1}{T} \sum_{t=1}^T MSE_t$$

with the model obtained from \bar{S}_t

This is a characteristic of how well the model predicts out-of-sample



Typical: $T \sim 5, 10$ machine learning

If $T = N$,

L00
leave-one-out

- Good estimate
- computationally expensive
- overfit.

\Rightarrow If

$$\vec{x}^{(i)} = \{0, 1\}^P, y^{(i)} \in \mathbb{R}$$

K-NN can also be applied to discrete descriptors

e.g.

$\vec{x}^{(i)}$	
$\vec{x}^{(j)}$	

Hamming distance: $d_H(\vec{x}^{(i)}, \vec{x}^{(j)})$

$$\text{distance} = 1 - \left| \sum_{m=1}^P I(x_m^{(i)} \neq x_m^{(j)}) \right|$$

Distance is the key:

\hookrightarrow Other methods use

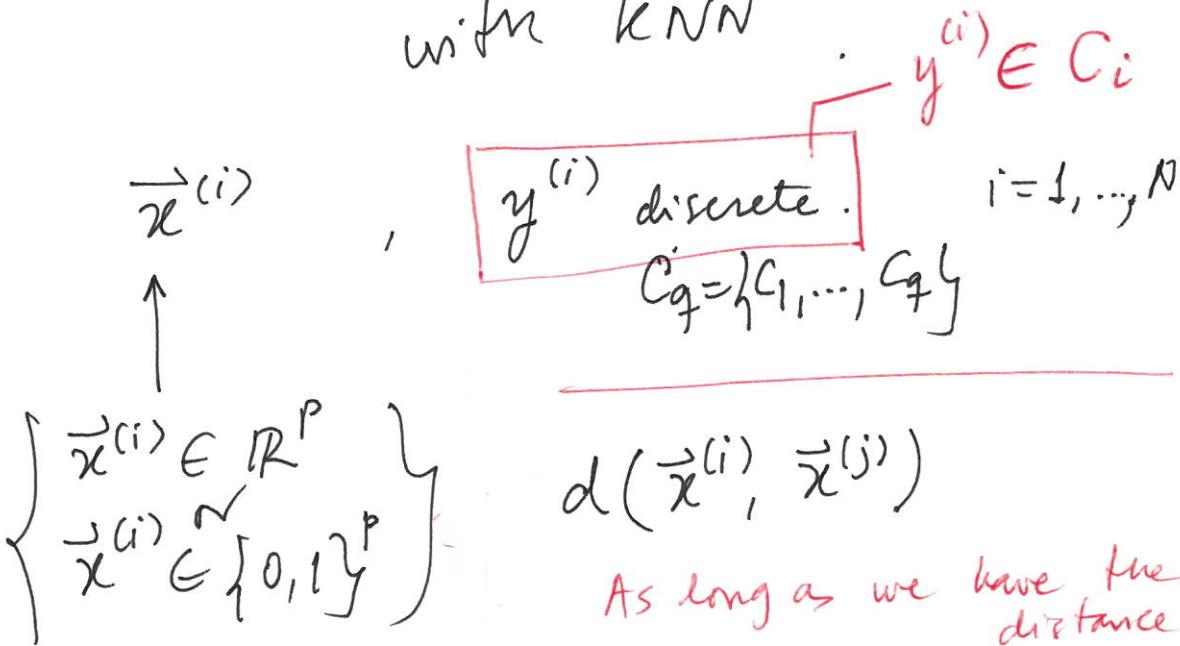
distances \Rightarrow kernels

to have local models

The rest is the same

$$N_k(\vec{x}^{in}) \text{ and } \hat{y} = \frac{1}{K} \sum_{i \in N_k(\vec{x}^{in})} y^{(i)}$$

Classification problem with KNN



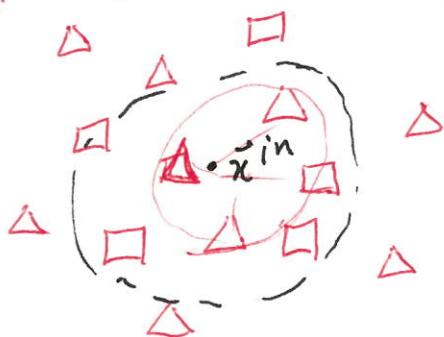
$$d(\vec{x}^{(i)}, \vec{x}^{(j)})$$

As long as we have the distance:

same ✓ ① Choose k (number of neighbors)

same ✓ ② For \vec{x}^{in} find $N_k(\vec{x}^{in})$

Predictor is different for classification: ^(new) ③ $\hat{y} \in C_g$ e.g. by majority rule.



$$k=4$$

$$\hat{y}(\vec{x}^{in}) \rightarrow \triangle$$

$$k=7$$

$$\hat{y}(\vec{x}^{in}) \rightarrow \square$$

Other predictors are used, including those that give a probabilistic output:

$$\forall i \in N_k(\vec{x}^{in}) \text{ compute } \vec{Q}(\vec{x}^{in}) = \sum_{i \in N_k(\vec{x}^{in})} I(y^{(i)} \in C_g)$$

$$I(\cdot) = \begin{cases} 1 & \text{if true} \\ 0 & \text{otherwise} \end{cases}$$

$$\vec{Q}(\vec{x}^{in}) = \begin{pmatrix} \# \text{ of points in } C_1 \\ \vdots \\ \# \text{ of points in } C_q \end{pmatrix} \frac{1}{K}$$

$q \times 1$

↓

Each component : Probability of \vec{x}^{in} belonging in
 Q_i class C_i

Standard KNN applies an argmax to this probability :

$$\hat{y} = \arg \max_q \vec{Q}(\vec{x}^{in})$$

↑
 Chooses the class
 with the maximum
 probability.

Global models for classification

$\vec{x}^{(i)} \in \mathbb{R}^P$; $y^{(i)} \in \{0, 1\}$ $i = 1, \dots, N$
 Binary.

Classic model: Logistic ~~"regression"~~
 ↳ classification.

Motivation: log odds are linear combination of descriptors

$$\beta_0 + \vec{x}^T \cdot \vec{\beta} = \log \frac{P(y=1)}{P(y=0)} = \log \frac{P(y=1)}{1-P(y=1)}$$

$$\vec{x}^T \cdot \vec{\beta} \stackrel{(p+1) \times 1}{=} \frac{P(y=1)}{1-P(y=1)} = e^{\vec{x}^T \cdot \vec{\beta}} \Rightarrow P(y=1) = \frac{1}{1+e^{-\vec{x}^T \cdot \vec{\beta}}} = h_{\vec{\beta}}(\vec{x})$$

Bernoulli variable
 with probability of success
 $P(y=1)$

$$\hookrightarrow P(Y=y | \vec{x}, \vec{\beta}) = h_{\vec{\beta}}(\vec{x})^y (1 - h_{\vec{\beta}}(\vec{x}))^{1-y}$$

If we assume independence in our samples; we can factorize probabilities to get:

$$P\left(\{y^{(i)}\} \mid \{\vec{x}^{(i)}\}, \vec{\beta}\right) = \prod_{i=1}^N h_{\vec{\beta}}(\vec{x}^{(i)})^{y^{(i)}} (1 - h_{\vec{\beta}}(\vec{x}^{(i)}))^{1-y^{(i)}}$$

Log-likelihood:

$$\mathcal{L} = \sum_{i=1}^N y^{(i)} \log h_{\vec{\beta}}(\vec{x}^{(i)}) + (1-y^{(i)}) \log (1 - h_{\vec{\beta}}(\vec{x}^{(i)}))$$

$$\nabla_{\vec{\beta}} \mathcal{L} \quad \text{to be maximized}$$

$$P(\{y^{(i)}\} \mid \{\vec{x}^{(i)}\}, \vec{\beta}) = \prod_{i=1}^N h_{\vec{\beta}}(\vec{x}^{(i)})^{y^{(i)}} (1 - h_{\vec{\beta}}(\vec{x}^{(i)}))^{1-y^{(i)}}$$

log-likelihood:

$$\mathcal{L} = \sum_{i=1}^N y^{(i)} \log h_{\vec{\beta}}(\vec{x}^{(i)}) + (1-y^{(i)}) \log (1 - h_{\vec{\beta}}(\vec{x}^{(i)}))$$

$$\Rightarrow \nabla_{\vec{\beta}} \mathcal{L} \quad \text{to be maximized}$$

Our model.

$$\left\{ \begin{array}{l} P(y=1) = h_{\vec{\beta}}(\vec{x}^{(i)}) = \frac{1}{1 + e^{-\vec{x}^{(i)\top} \vec{\beta}}} = h(\vec{x}^{(i)\top} \vec{\beta}) \\ P(y=0) = 1 - h_{\vec{\beta}}(\vec{x}^{(i)}) = \frac{e^{-\vec{x}^{(i)\top} \vec{\beta}}}{1 + e^{-\vec{x}^{(i)\top} \vec{\beta}}} = e^{-\vec{x}^{(i)\top} \vec{\beta}} h(\vec{x}^{(i)\top} \vec{\beta}) \end{array} \right.$$

Note that: $\log (1 - h_{\vec{\beta}}(\vec{x}^{(i)})) = -\vec{x}^{(i)\top} \vec{\beta} + \log(h(\vec{x}^{(i)\top} \vec{\beta}))$

$$\mathcal{L} = \sum_{i=1}^N \underbrace{\log h_{\vec{\beta}}(\vec{x}^{(i)})}_{\log h(\vec{x}^{(i)\top} \vec{\beta})} - (1-y^{(i)}) (\vec{x}^{(i)\top} \vec{\beta})$$

$$\nabla_{\vec{\beta}} \left[\log h_{\vec{\beta}}(\vec{x}^{(i)}) \right] = \frac{e^{-\vec{x}^{(i)\top} \vec{\beta}}}{1 + e^{-\vec{x}^{(i)\top} \vec{\beta}}} \vec{x}^{(i)}$$

$$= (1 - h_{\vec{\beta}}(\vec{x}^{(i)})) \vec{x}^{(i)}_{(p+1) \times 1}$$

$$\nabla_{\beta} L = \sum_{i=1}^N \left[1 - h_{\beta}(\vec{x}^{(i)}) \right] \vec{x}^{(i)} - (1 - y^{(i)}) \vec{x}^{(i)}$$

$$= \sum_{i=1}^N \left[y^{(i)} - h_{\beta}(\vec{x}^{(i)}) \right] \vec{x}^{(i)}$$

=====

Maximum :

$$\nabla_{\beta} L \Big|_{\beta^*} = 0$$

$\sum_{i=1}^N \left[y^{(i)} - h(\vec{x}^{(i)T} \cdot \beta^*) \right] \vec{x}^{(i)} = 0$

Equivalent of
normal equations
in LS.

$(p+1)$ equations

L is concave \Rightarrow global maximum
 β^* is β^*

which can be found by
standard optimisation.

$$P(\hat{y}=1) = h(\vec{x}^{inT} \cdot \beta^*) \in [0, 1]$$

classifier :

$$P(\hat{y}=1 | \vec{x}^{in}) > T$$

$$\vec{x}^{in} \in \{1\}$$

T is the threshold for the
classifier: usually

[But it can be
tuned a posteriori]

$$T = \frac{1}{2}$$

Summary of our solution for logistic regression:

$$X_{N \times (p+1)}$$

$$X\vec{\beta}^* = \vec{y}_{NK1}$$

Normal equations

$$X^T [\vec{y} - h(X\vec{\beta}^*)] = 0$$

$$\vec{h}(X\vec{\beta}^*) \rightarrow h_i = \left(\frac{1}{1 + e^{-x^{(i)\top} \vec{\beta}^*}} \right)$$

logistic regression

Compare

Remember that for LS:

Normal equations for
LS

$$X^T [\vec{y} - X\vec{\beta}_{LS}^*] = 0$$

log

Quality of the classifier

confusion matrix

or

contingency table

Two-classes

For a $\{0, 1\}$ classifier

		True	
		1	0
		True positives (TP)	False positives (FP)
Predicted	1	True positives (TP)	False positives (FP)
	0	False negatives (FN)	True negative (TN)

Type I error

Type II errors

We want most cases on the diagonal

{ True positive rate = $\frac{TP}{TP+FN} = TPR$

||
Sensitivity or recall

{ True negative rate : $\frac{TN}{TN+FP} = TNR$

||
Specificity

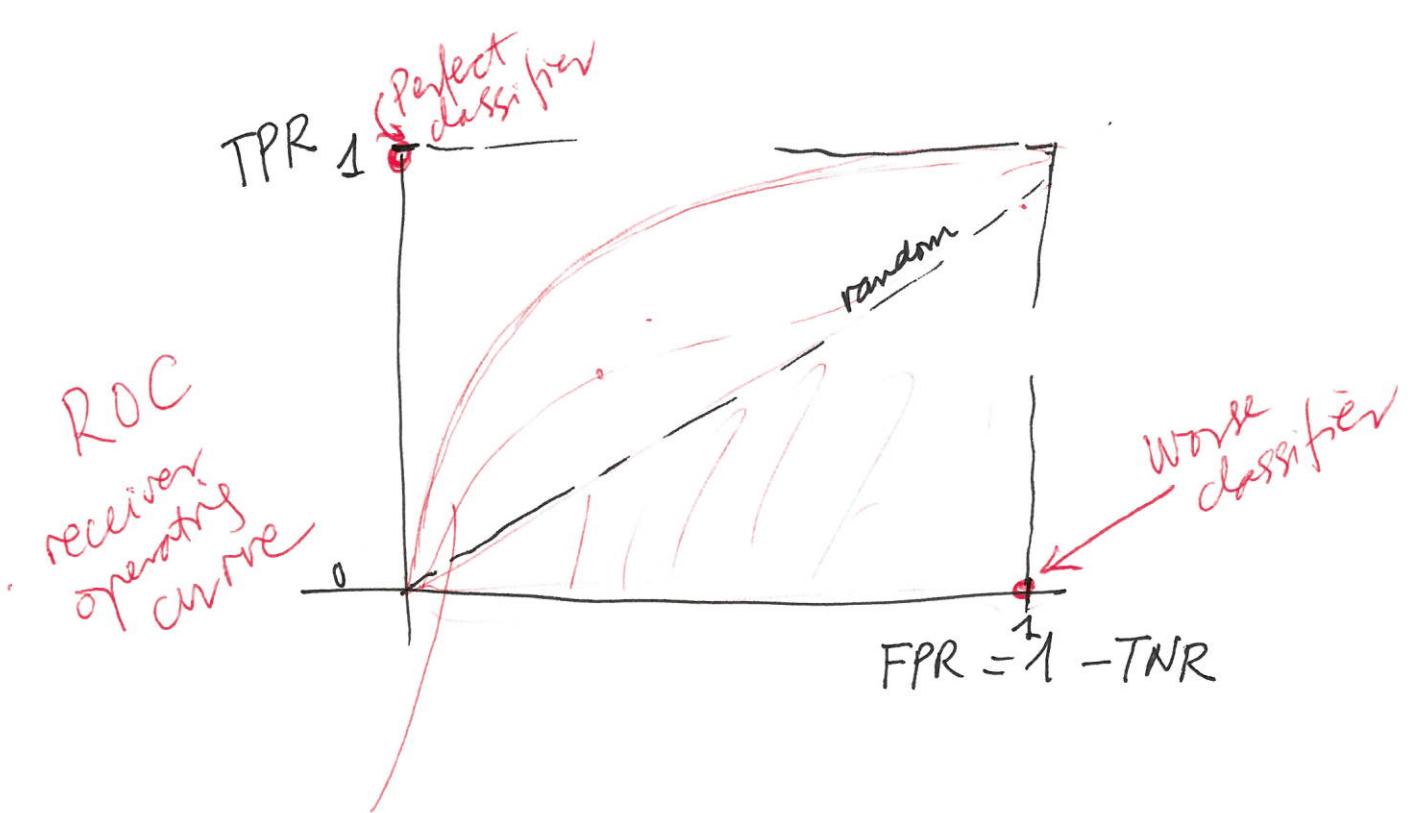
{ Accuracy : $\frac{TP+TN}{N_{\text{validation}}}$

{ Precision = $\frac{TP}{TP+FP}$

Recall = $\frac{TP}{TP+FN}$

Precision = $\frac{TP}{TP+FP}$

$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$



AUC : Area under the Curve
is an overall
measure of quality

$AUC > \frac{1}{2}$ Better than random
over a range of
parameters

Naive Bayes classifiers

Data: $\vec{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)})$ $i=1, \dots, N$
 $y^{(i)} \in C_q \subset \{C_1, \dots, C_Q\}$
 Q classes.

Bayes' theorem:

$$P(Y=y_q | X=\vec{x}) = \frac{P(X=\vec{x} | Y=y_q) P(Y=y_q)}{P(X=\vec{x})}$$

(A) Prior: (A.1) frequency of each class
 in the training set.

$$P(Y=y_q) = \frac{\sum_{i=1}^N I(y^{(i)}=y_q)}{N}$$

$$I(A) = \begin{cases} 1, & \text{if } A \text{ is true} \\ 0, & \text{otherwise} \end{cases}$$

or (A.2) use any additional information
 to inform your prior choice.

(B)

Naïve assumption:

$$P(X_1=x_1, X_2=x_2, \dots, X_p=x_p \mid Y=y_q) =$$

$$= \prod_{j=1}^p P(X_j=x_j \mid Y=y_q)$$

Wrong but it works!

B.1

$$P(X_j=x_j \mid Y=y_q) \approx \frac{\sum_{i=1}^N I(x_j^{(i)}=x_j \& y^{(i)}=y_q)}{\sum_{i=1}^N I(y^{(i)}=y_q)}$$

Laplace smoothing :

$$P(X_j=x_j \mid Y=y_q) = \frac{\sum_{i=1}^N [I(x_j^{(i)}=x_j \& y^{(i)}=y_q)] + 1}{\sum_{i=1}^N I(y^{(i)}=y_q) + N}$$

B.2 Assume a distribution and the construct estimators for the parameters of the distribution.

e.g. Gaussian:

$$P(X_j = x_j | Y = y_q) = \frac{1}{\sqrt{2\pi \sigma_{j,q}^2}} e^{-\frac{(x_j - \mu_{j,q})^2}{2\sigma_{j,q}^2}}$$

$$\mu_{j,q} = \frac{1}{N_q} \sum_{y^{(i)} \in C_q} x_j^{(i)}$$

Sample mean
for x_j .

$$\sigma_{j,q}^2 = \frac{1}{N_q - 1} \sum_{y^{(i)} \in C_q} (x_j^{(i)} - \mu_{j,q})^2$$

Sample variance

$$(C) P(X = \vec{x}) = \sum_{q=1}^Q P(X = \vec{x} | Y = y_q) P(Y = y_q)$$

(B) (A)

$$= \sum_{q=1}^Q [A \times B]_q$$

$$P(Y = y_q | X = \vec{x}) = \frac{[A \times B]_q}{\sum_{q=1}^Q [A \times B]_q}$$

$$(A)_q = P(Y = y_q) = \frac{1}{N} \sum_{i=1}^N I(y^{(i)} = y_q)$$

$$(B)_q = \prod_{j=1}^P P(X_j = x_j | Y = y_q) = \prod_{j=1}^P \left[\frac{\sum_{i=1}^N I(x_j^{(i)} = x_j \wedge y^{(i)} = y_q)}{\sum_{i=1}^N I(y^{(i)} = y_q)} \right]$$

outcome : Given \vec{x}^{in}

our NB classifier gives us a probability vector of \vec{x}^{in} belonging to each of the Q classes

$$\vec{\pi}^{(NB)}_{Q \times 1} = \begin{bmatrix} \pi_1^{(NB)} \\ \vdots \\ \pi_Q^{(NB)} \end{bmatrix}$$

$$\pi_q^{(NB)} = \frac{[\mathcal{A} \times \mathcal{B}]_q}{\sum_{q=1}^Q [\mathcal{A} \times \mathcal{B}]_q}$$

$$\vec{1}^T \cdot \vec{\pi}^{(NB)} = 1 \quad \checkmark \quad (\text{by construction!})$$

Classifier \Rightarrow $\hat{y}^{(NB)} = \arg \max_q \vec{\pi}^{(NB)}$

choose the maximum:

MAP (maximum a posteriori)

Leading to Random Forests

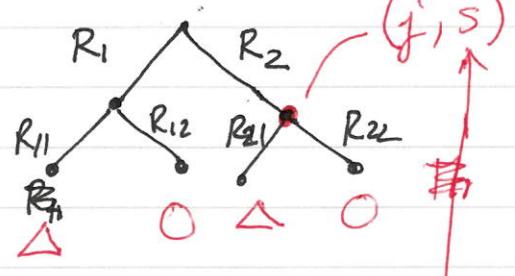
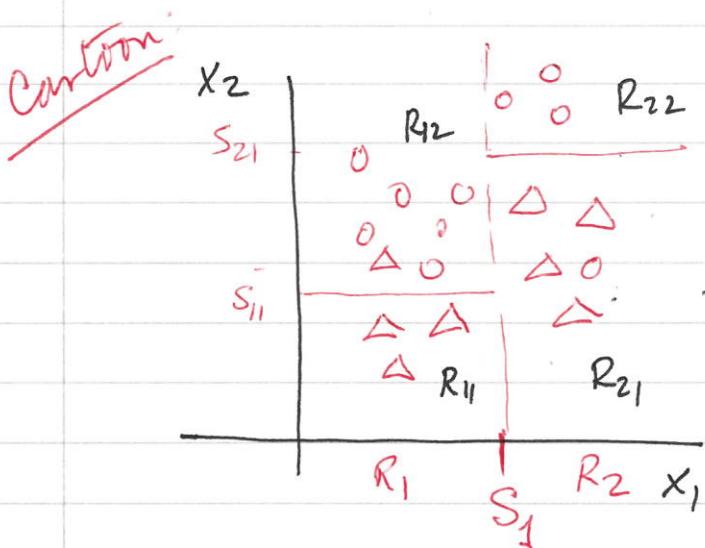
we start with decision trees.

$$\vec{x} = (x_1, \dots, x_p) \rightarrow y$$

↑ Discrete
or continuous

↑ Discrete or
continuous

Descriptor
that is split



In this case: $(j=2, s=S_{21})$

Each split $\left\{ \begin{array}{l} \{x_j : x_j < s \in R_1(j, s)\} \\ \{x_j : x_j \geq s \in R_2(j, s)\} \end{array} \right.$
~~(j, s)~~ corresponds to:

Regression:

choose (j, s) that minimizes a loss:

$$\min_{j, s} \left[\sum_{\vec{x}^{(i)} \in R_1(j, s)} (y^{(i)} - \bar{y}_{R_1})^2 + \sum_{\vec{x}^{(i)} \in R_2(j, s)} (y^{(i)} - \bar{y}_{R_2})^2 \right]$$

$$\bar{y}_{R_1} = \frac{\sum_{i=1}^N I(x_j^{(i)} < s) \cdot y^{(i)}}{\sum_{i=1}^N I(x_j^{(i)} < s)}$$

~~We represent each of the points in the region by the mean~~

- Optimise cost at every split:

greedily.

- Continue until a stopping criterion is met:

- e.g.
- Plateau in optimisation
 - small # of points in regions
 - using all predictors

Given \vec{x}^{in}

Find $R_{\vec{x}} / \vec{x}^{in} \in R_{\vec{x}}$

$$\hat{y}_{DT} = \bar{y}_{R_{\vec{x}}}$$

Mean of
the region
where \vec{x}^{in} falls.

- Optimise cost at every split.

greedily.

- Continue until a stopping criterion is met:

- e.g.
- Plateau in optimisation
 - small # of points in regions
 - using all predictors

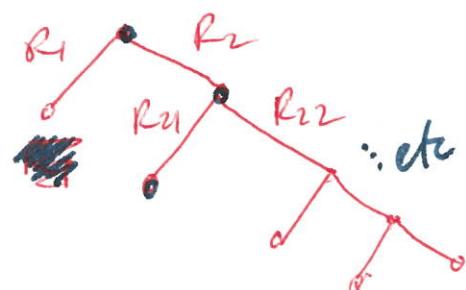
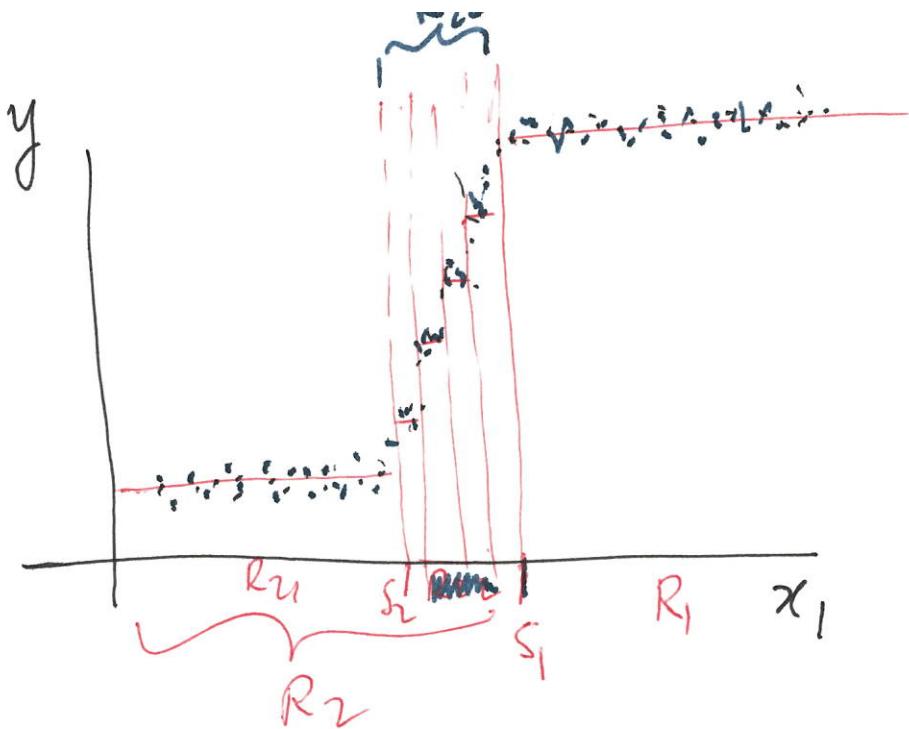
Given \vec{x}^{in}

Find $R_{\vec{x}} / \vec{x}^{in} \in R_{\vec{x}}$

$$\hat{y}^{DT} = \underline{\bar{y}}$$

Mean of
the region
where \vec{x}^{in} falls.

(TBC...)



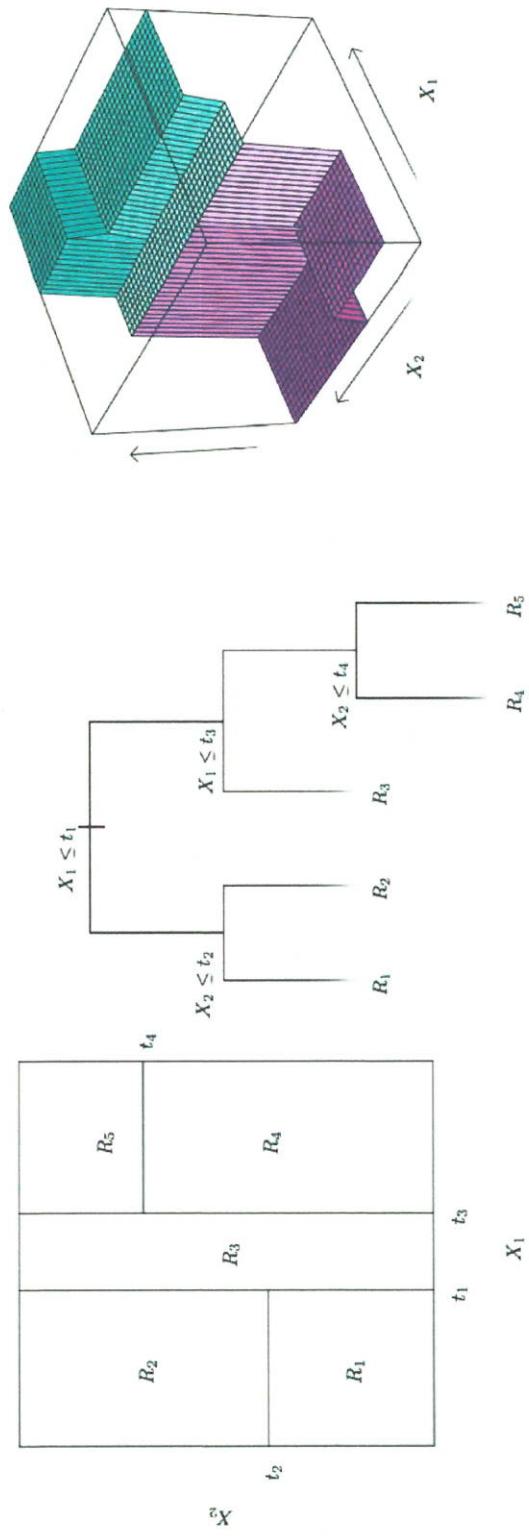
Example of how a DT would proceed with ~~a~~
the case of one descriptor

$$\left\{ \begin{array}{l} \vec{x} = x_1 \in \mathbb{R} \\ \text{and} \\ y \in \mathbb{R} \end{array} \right\}$$

Expressiveness of Decision Trees

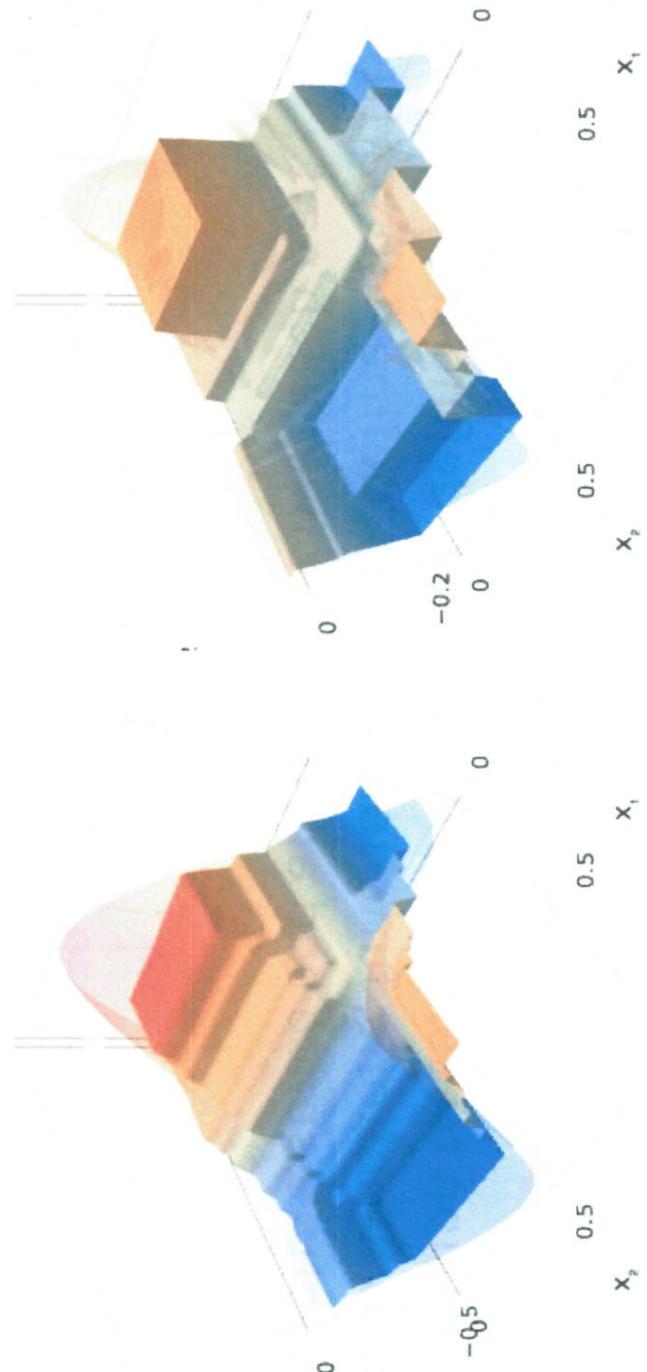
We've seen that classification trees approximate boundaries in the feature space that separate classes.

Regression trees, on the other hand, define **simple functions** or step functions, functions that are defined on partitions of the feature space and are constant over each part.

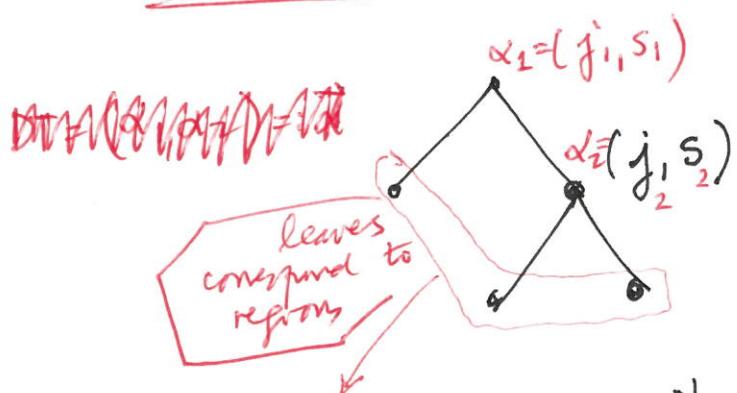


Expressiveness of Decision Trees

For a fine enough partition of the feature space, these functions can approximate complex non-linear functions.



Decision trees for classification:



Same as above:
Each split denoted by $(j_i, s_i) = \alpha_i$

For region $\vec{R}_{\vec{\alpha}}$:

$$[\vec{\Pi}(\vec{R}_{\vec{\alpha}})]_q = \frac{\sum_{i=1}^N I(\vec{x}^{(i)} \in \vec{R}_{\vec{\alpha}} \wedge y^{(i)} \in C_q)}{\sum_{i=1}^N I(\vec{x}^{(i)} \in \vec{R}_{\vec{\alpha}})}$$

$$q = 1, \dots, Q$$

$$\vec{\Pi}(\vec{R}_{\vec{\alpha}}) = \begin{bmatrix} \vdots \\ \vec{\Pi}(\vec{R}_{\vec{\alpha}})_q \\ \vdots \end{bmatrix}$$

Probability vector of belonging to each class q in $\vec{R}_{\vec{\alpha}}$

Given \vec{x}^{in} :

(1) Find $\vec{R}_{\vec{\alpha}} \ni \vec{x}^{in} \in \vec{R}_{\vec{\alpha}}$

(2) Obtain $\vec{\Pi}(\vec{R}_{\vec{\alpha}})$

(3) $\hat{y} = \hat{f}^{DT}(\vec{x}^{in}) = \arg \max_q \vec{\Pi}(\vec{R}_{\vec{\alpha}}(\vec{x}^{in}))$

Important tweak:

What is ~~the~~ Cost function for the splits
in the decision tree:

(1) Contingency table:

Minimize the error rate.

not sensitive

(2) [Gini index:
or
Cross-entropy]

Cost function

deals with
how informative
the split is:

Gini index of $\vec{\pi}(R\vec{x})$

$$GI[\vec{\pi}(R\vec{x})] = \sum_{q=1}^Q \pi(R\vec{x})_q (1 - \pi(R\vec{x})_q)$$

Cross entropy

$$CE[\vec{\pi}(R\vec{x})] = \sum_{q=1}^Q \pi(R\vec{x})_q \log(1 - \pi(R\vec{x})_q)$$

Two versions of information
(or entropy)

Properties of GI:

$$i \in \{1, \dots, J\}$$

$$\begin{bmatrix} p_1 \\ \vdots \\ p_J \end{bmatrix} = \vec{p}$$

$$\text{GI}(\vec{p}) = \sum_{i=1}^J p_i (1 - p_i) = 1 - \sum_{i=1}^J p_i^2$$

Lagrange multipliers

$$L = \text{GI}(\vec{p}) - \lambda \left(\sum_{i=1}^J p_i - 1 \right)$$

$$\left\{ \begin{array}{l} \frac{\partial L}{\partial p} = -2 \vec{p} + \vec{1} \\ \frac{\partial L}{\partial \lambda} = \sum_{i=1}^J p_i - 1 \end{array} \right\}$$

$$\vec{p}^* = \frac{-\lambda}{2} \vec{1} \quad p_i^* = \frac{1}{J}$$

- The maximum GI is when all elements of a probability vector are equal

cost function is GI:
In every split we minimize GI:
Find (j, s) such that
 $\min_{GIs} \text{GI}(\vec{p}(r_s))$

$$\vec{p} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \Rightarrow \text{GI}(\vec{p}) = 0$$

It's the minimum.

Decision trees tend to overfit

i.e., New training data usually lead to very different trees.

Solution:

Introduce randomised algorithms.

1. Bagging = Bootstrap aggregation

$$S = \{ \vec{x}^{(i)}, y^{(i)} \} \quad i = 1, \dots, N$$

Our sample

(1) Bootstrapping: Produce B samples from S all of size N by random sampling with replacement: S_b , $b=1, \dots, B$

(2) Models: From each of the B samples obtain

$$\left\{ \hat{f}_b^{\text{DT}} \right\}_{b=1}^B$$

In each S_b there will be repeated samples and absent samples from the original S

(3) Aggregate the model:

$$(i) \hat{f}(\vec{x}^{in}) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b^{DT}(\vec{x}^{in})$$

Regression

(ii) Classification:

$$\vec{x}^{in} \mapsto [\vec{\pi}_1^{DT}, \dots, \vec{\pi}_B^{DT}]$$

$$\vec{\pi}_b = \begin{pmatrix} \pi_{b,1} \\ \vdots \\ \pi_{b,Q} \end{pmatrix}$$

α classes

$$\vec{\pi} = \frac{1}{B} \sum_{b=1}^B \vec{\pi}_b^{DT} \quad (\text{aggregated probability})$$

$$\hat{y} = \arg \max_g \vec{\pi}$$

These outcomes ~~are~~ are still correlated usually because of dominance of some descriptors.

If the $\{\hat{f}_b^{DT}(\vec{x}^{in})\}$ for a given \vec{x}^{in} are all correlated, then averaging them does not reduce the variance beyond a limit.

Hence we need to un correlate more radically.

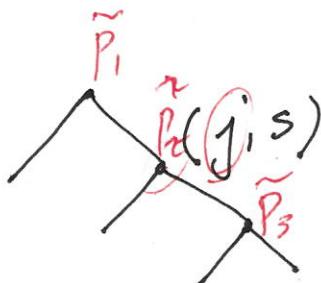
Random forests add an additional level of randomisation.

Steps:

(1) Generate a bootstrap sample

S_b

(2)



In DT

$$j \in [1, \dots, P]$$

maximize over all predictors

In RF:

At each split choose x_j from a random subset of descriptors with \tilde{p} descriptors.

$x_j \in \{x_i\}_{i \in RF}$ random subset of descriptors

$$|RF| = \tilde{p} < p$$

Chosen at random among the $[1, \dots, P]$

\hat{f}^{RF}

$[\tilde{P}_1, \tilde{P}_2, \tilde{P}_3, \dots]_S$

(3) Repeat S times

$$S = 1, \dots, S^d$$

u

The result is an ensemble of trees:

$$\left\{ \hat{f}_{[\tilde{p}_k]}^{\text{RF}} \right\}_{s=1}^{S^1} = \underline{\text{random forest}}$$

k is defined on each of the splits.

The predictor is the aggregation of

$$\left\{ \hat{f}_{[\tilde{p}_k]}^{\text{RF}} \right\}_{s=1}^{S^1}$$

- Regression: average of $\left\{ \hat{f}_{[\tilde{p}_k]}^{\text{RF}} \right\}_{s=1}^{S^1}$
- Classification: $\left\{ \vec{T}_{[\tilde{p}_k]}^{\text{RF}} \right\}_{s=1}^{S^1}$

Some comments:

(1) Loss of interpretability:

with aggregation we lose the direct connection with the descriptors.

* Parameters (hyper-parameters) :

- $S = \# \text{ of trees}$. Not very sensitive

- Minimum leaf size : $\begin{cases} \text{Classification} \rightarrow 1 \\ \text{Regression} \rightarrow 5 \end{cases}$

- $\tilde{p} < p$ $\begin{cases} \tilde{p} \sim \sqrt{p} & \text{prediction (regression)} \\ \tilde{p} \sim \frac{p}{3} & \text{classification} \end{cases}$
 $\# \text{ of descriptors allowed at each split}$

{ If $\tilde{p}=1$, trees are very uncorrelated }
 { If $\tilde{p} \approx p$, trees are correlated }

- * Bootstrap bags as some extra knowledge :

For every sample S_b , there are some 'out-of-bag' samples.

out-of-bag error = OOB error can be used for validation

- Example of ensemble methods

RF is an example of an ensemble method.

Others : $\begin{cases} (1) \text{ Mixing different classifiers} \\ (2) \text{ Generating ensemble of trees by } \underline{\text{boosting}} \end{cases}$

Support vector machines (SVM's) - Vapnik

Classification or geometric separation.

$$\vec{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)})$$

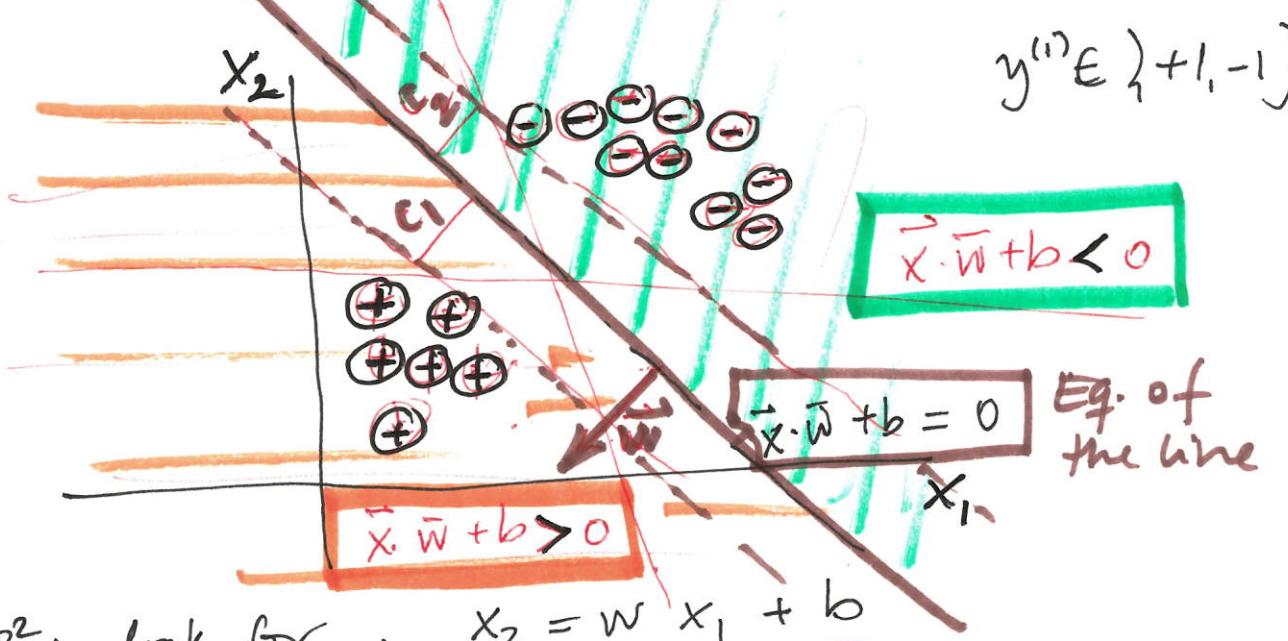
$$y^{(i)} \in \{-1, +1\}$$

Assume

$$\vec{x}^{(i)} \in \mathbb{R}^2$$

$$\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})$$

$$y^{(i)} \in \{+1, -1\}$$



In \mathbb{R}^2 : look for a line : $x_2 = w_1 x_1 + b$

$$\vec{x} \cdot \vec{w} + b = 0$$

$$\vec{w} = (w_1, -1)$$

$$(w_1, -1) \cdot (x_1, x_2) + b = 0 \quad \checkmark$$

For $\vec{x} \in \mathbb{R}^P$

$$\left[\vec{x}^T \vec{\beta} = \right] \vec{x} \cdot \vec{w} + b = 0$$
$$\vec{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix}, \vec{\beta} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_p \end{pmatrix}, \vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix}$$

Equation of
a hyperplane

Find the hyperplane

$$\vec{x} \cdot \vec{w} + b = 0$$

(\vec{w}, b) to be found.

$$c_1, c_2 \in \mathbb{R}^+ \quad \left\{ \begin{array}{l} \vec{x}^{(i)} \cdot \vec{w} + b > c_1 \text{ if } y^{(i)} = +1 \\ \vec{x}^{(i)} \cdot \vec{w} + b < -c_2 \text{ if } y^{(i)} = -1 \end{array} \right\}$$

Support vector machines

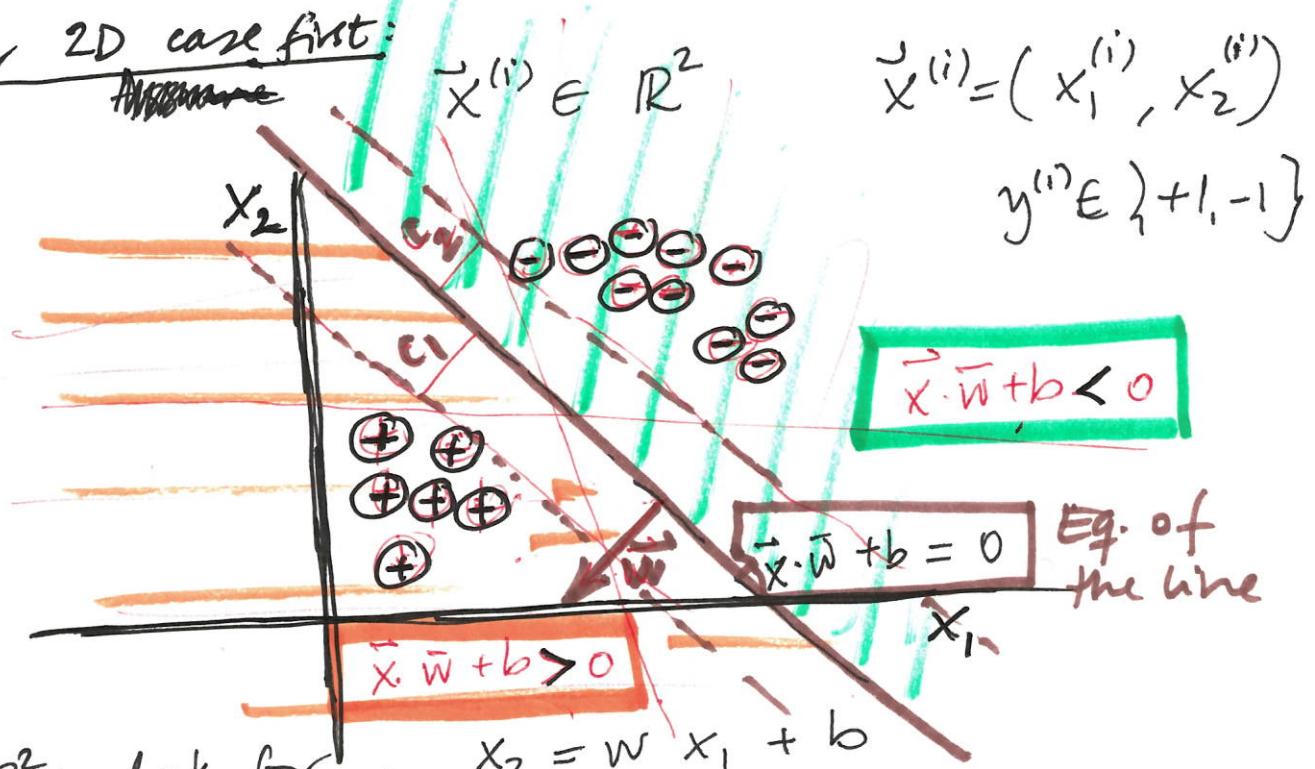
(SVM's)

- Vapnik

Classification as geometric separation.

$$\left[\vec{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)}) \text{ with } y^{(i)} \in \{-1, +1\} \right]$$

Consider 2D case first:



In \mathbb{R}^2 : look for a line

$$x_2 = w_1 x_1 + b$$

$$\vec{x} \cdot \vec{w} + b = 0$$

$$\vec{w} = (w_1, -1)$$

$$(w_1, -1) \cdot (x_1, x_2) + b = 0 \quad \checkmark$$

In general,

for $\vec{x} \in \mathbb{R}^P$

Note that:

$$\left[\vec{x}^T \cdot \vec{\beta} \right] = \vec{x} \cdot \vec{w} + b = 0$$

Equation of
a hyperplane

$$(1) \quad \vec{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix}, \vec{\beta} = \begin{pmatrix} b \\ w_1 \\ \vdots \\ w_p \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix}, \vec{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix}$$

Find the hyperplane

$$\left\{ \begin{array}{l} \vec{x} \cdot \vec{w} + b > 0 \text{ below} \\ \vec{x} \cdot \vec{w} + b < 0 \text{ above} \end{array} \right\} \quad \vec{x} \cdot \vec{w} + b = 0 \quad (\vec{w}, b) \text{ to be found.}$$

$$c_1, c_2 \in \mathbb{R}^+ \quad \left\{ \begin{array}{l} \vec{x}^{(i)} \cdot \vec{w} + b > c_1 \text{ if } y^{(i)} = +1 \\ \vec{x}^{(i)} \cdot \vec{w} + b < -c_2 \text{ if } y^{(i)} = -1 \end{array} \right\}$$

Once we have $(\vec{w}, b) = \vec{\beta}^T$, we have our model to classify according to the above

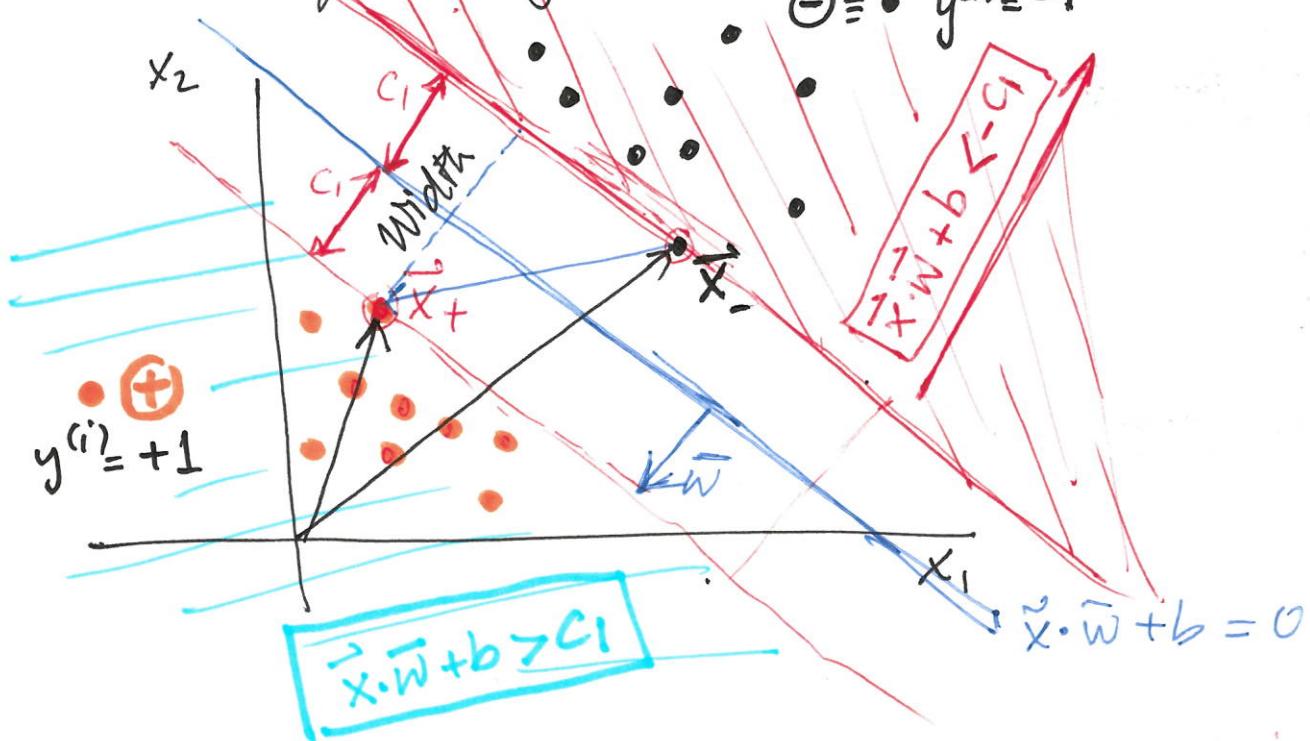
$$\left\{ \begin{array}{l} \vec{x}_{in} \cdot \vec{w} + b > 0 \text{ then } \hat{y} = +1 \\ \vec{x}_{in} \cdot \vec{w} + b < 0 \text{ then } \hat{y} = -1 \end{array} \right.$$

What is the criterion for the hyperplane?

Make the width of the street as large as possible

~~Geometry of the SVM criterion:~~

$$\Theta \equiv \bullet \quad y^{(i)} = -1$$



①

$$y^{(i)} (\vec{x}^{(i)} \cdot \vec{w} + b) \geq 0$$

Condition on
both sides of
hyperplane

②

The solution is half way:

$$\vec{x}_+ \cdot \vec{w} + b = C_1$$

$$\vec{x}_- \cdot \vec{w} + b = -C_1$$

$$\text{Width} = (\vec{x}_+ - \vec{x}_-) \cdot \frac{\vec{w}}{\|\vec{w}\|} = \frac{2C_1}{\|\vec{w}\|}$$

$$C_1 = 1$$

✓ Can always
be fixed

$$\text{Width} = \frac{2}{\|\vec{w}\|}$$

$$\max \text{ width} \Rightarrow \min_{\bar{w}} \frac{1}{2} \|\bar{w}\|^2$$

Optimization (SVM):

$$\min_{\bar{w}} \frac{1}{2} \|\bar{w}\|^2$$

subject to

$$y^{(i)} (\vec{x}^{(i)} \cdot \bar{w} + b) \geq 1$$

$$i=1, \dots, N$$

All points above/below
the two lines
at distance 1 from
 $\vec{x} \cdot \bar{w} + b = 0$

Reminder :

(+) Lagrange multipliers:

Optimize a function
subject to
equality constraints

$$\begin{aligned} & \min_{\vec{x}} f(\vec{x}) \\ & \text{subject to } g_i(\vec{x}) = 0 \quad i=1, \dots, m \end{aligned}$$

Lagrangian : $L = f(\vec{x}) + \sum_{i=1}^m \alpha_i g_i(\vec{x})$

$$\left\{ \begin{array}{l} \nabla_{\vec{x}} L = 0 \\ \nabla_{\alpha} L = 0 \end{array} \right\} \text{ at } (\vec{x}^*, \alpha^*)$$

(2) Linear programming :

Optimize a linear objective function under linear constraints (equalities and inequalities)

$$\min_{\vec{x}} \vec{w} \cdot \vec{x}$$

(linear cost)

subject to

$$g_i(\vec{x}) = 0 \quad i=1, \dots, m$$

g_i are linear

$$h_j(\vec{x}) \geq 0 \quad j=1, \dots, k$$

h_j are linear

The constraints
define a convex set

Simplex solves this problem.

(3) Quadratic program: (QP)

Solution given by KKT
(Karush, Kuhn, Tucker)
1939 1951

$$\min_{\vec{x}} f(\vec{x})$$

quadratic

subject to linear $g_i(\vec{x}) \leq 0$

$$i=1, \dots, m$$

Construct Lagrangian: Lagrange multipliers

$$L = f(\vec{x}) + \sum_{i=1}^m \alpha_i g_i(\vec{x})$$

At the minimum: \vec{z}^* , there exists $\vec{\alpha} = (\alpha_1, \dots, \alpha_m)$ such

that: ① $-\nabla f(\vec{z}) = \sum_{i=1}^m \alpha_i \nabla g_i(\vec{z}^*)$, $\nabla L|_{\vec{z}^*} = 0$

Primal

$$\textcircled{2}$$

$$g_i(\vec{z}^*) \leq 0 \quad \forall i$$

$$\nabla L|_{\vec{\alpha}} = 0$$

$$\textcircled{3}$$

$$\alpha_i \geq 0 \quad \forall i$$

Dual

$$\textcircled{4}$$

$$\alpha_i \cdot g_i(\vec{z}^*) = 0 \quad \forall i$$

complementary slackness.

Applied to SVM optimisation:

$$\min_{\vec{w}} \frac{1}{2} \|\vec{w}\|^2$$

subject to $1 - y^{(i)}(\vec{x}^{(i)} \cdot \vec{w} + b) \leq 0$

$g_i(\vec{w}, b) \quad i=1, \dots, N$

Construct Lagrangian:

$$L(\bar{w}, b; \vec{\alpha}) = \frac{1}{2} \|\bar{w}\|^2 + \sum_{i=1}^N \alpha_i (1 - y^{(i)} (\vec{x}^{(i)} \cdot \bar{w} + b))$$

$$\left. \begin{aligned} \frac{\partial L}{\partial b} &= - \sum_{i=1}^N y^{(i)} \alpha_i \\ \nabla_{\bar{w}} L &= \bar{w} - \sum_{i=1}^N \alpha_i y^{(i)} \vec{x}^{(i)} \end{aligned} \right\} \begin{array}{l} \text{At the minimum} \\ \text{they equal zero} \end{array}$$

$$\left. \begin{aligned} \sum_{i=1}^N y^{(i)} \alpha_i &= 0 \\ \bar{w}^* &= \sum_{i=1}^N \alpha_i y^{(i)} \vec{x}^{(i)} \end{aligned} \right\}$$

Rewrite in terms of $\vec{\alpha}$:

$$L(\vec{\alpha}) = \sum_{i=1}^N \alpha_i - \sum_{i=1}^N \alpha_i y^{(i)} \vec{x}^{(i)} \cdot \bar{w} -$$

$$- \cancel{\sum_{i=1}^N \alpha_i y^{(i)} b} + \frac{1}{2} \bar{w} \cdot \bar{w}$$

$$L(\vec{\alpha}) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \bar{w} \cdot \bar{w} = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} \vec{x}^{(i)} \cdot \vec{x}^{(j)}$$

Conclusion: $[\alpha_i = 0 \quad \text{if} \quad \vec{x}^{(i)} \neq \vec{x}_+ \text{ or } \vec{x}_-]$

From ④ $\alpha_i (1 - y^{(i)} (\vec{x}^{(i)} \cdot \bar{w} + b)) = 0 \quad \forall i$

$$\begin{aligned}\bar{w}^x &= \sum_{i=1}^N \alpha_i y^{(i)} \bar{x}^{(i)} \\ &= \alpha_+ \bar{x}_+ + \alpha_- \bar{x}_- \\ &= \uparrow \quad = \uparrow \\ &\text{"Anchor points."} \\ &\text{Support vectors}\end{aligned}$$

These are found by the process of convex optimization above

Hard-margin SVM

(Perfect separation)

How do we extend SVMs to more realistic scenarios when there is no hyperplane that can separate the classes perfectly?

- ① Soft-margin SVM = soft boundary
- ② Going beyond linear. \Rightarrow Kernels

Imperfect separation

Soft - margin SVM

Definition: Hinge loss: cost of a violation.

$$\xi^{(i)} = \max \left(0, 1 - (\vec{x}^{(i)} \cdot \vec{w} + b) y^{(i)} \right)$$

size of violation

Soft-margin SVM
optimisation:

$$\min_{\vec{w}} \frac{1}{2} \|\vec{w}\|^2 + \sum_{i=1}^N \xi^{(i)}$$

$$\text{subject to } 1 - y^{(i)} (\vec{w} \cdot \vec{x}^{(i)} + b) \leq \xi^{(i)}$$
$$\xi^{(i)} \geq 0 \quad i=1, \dots, N$$

Remember that $y^{(i)} (\vec{x}^{(i)} \cdot \vec{w} + b) \geq 1$
for all points when there is no violation

Beyond linearity:

Kernelised SVM

based on the 'Kernel trick'

Standard SVM (linear)

linear classifier (\vec{w}^*, b)

$$\vec{w}^* = \sum_{i=1}^N \alpha_i y^{(i)} \vec{x}^{(i)}$$

$\alpha_i = 0 \quad \text{if } i \text{ not a support vector}$

Assume

there are only two support vectors (generic case):

Then only α_+ and $\alpha_- \neq 0$: $\vec{w}^* = \alpha_+ \vec{x}_+ - \alpha_- \vec{x}_-$

$$\begin{aligned} b &= 1 - \vec{x}_+ \cdot [\alpha_+ \vec{x}_+ - \alpha_- \vec{x}_-] \\ &= 1 - \alpha_+ \boxed{\vec{x}_+ \cdot \vec{x}_+} + \alpha_- \boxed{\vec{x}_+ \cdot \vec{x}_-} \end{aligned}$$

These define our model

Obtained from
maximising of
the dual:

$$L(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y^{(i)} y^{(j)} (\vec{x}^{(i)} \cdot \vec{x}^{(j)})$$

Given \vec{x}^{in} ,

$$\left\{ \begin{array}{l} \vec{x}^{in} \cdot \vec{w}^* + b \geq 0 \Rightarrow \hat{y} = +1 \\ \vec{x}^{in} \cdot \vec{w}^* + b \leq 0 \Rightarrow \hat{y} = -1 \end{array} \right.$$

Kernel functions

Given $\vec{x}, \vec{y} \in \mathbb{R}^d$, consider $\vec{\phi}(\vec{x}) = \vec{z} \in \mathbb{R}^D$
 potentially in $D \gg d$

In some cases, for the right ϕ we have more properties:

$$\text{e.g. } \vec{x} = (x_1, x_2) \in \mathbb{R}^2$$

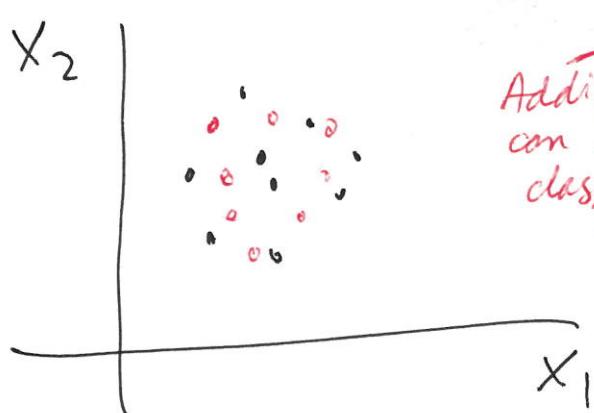
$$\vec{\phi}(\vec{x}) = \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \in \mathbb{R}^3$$

$$\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{y}) = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2$$

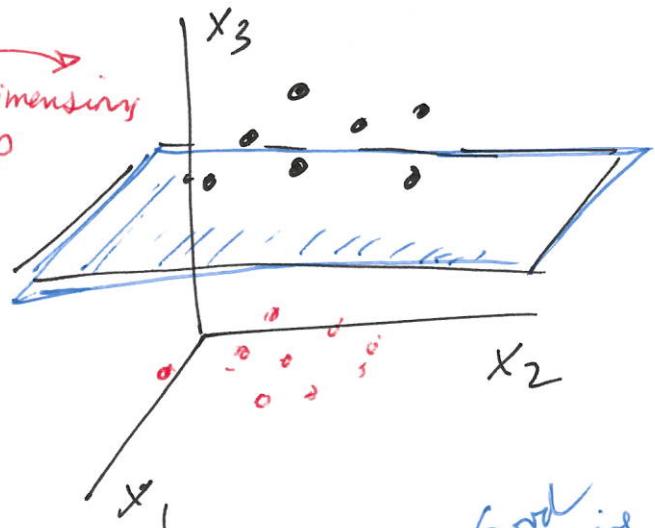
$$\vec{x} \cdot \vec{y} = x_1 y_1 + x_2 y_2$$

$$\underbrace{g(\vec{x} \cdot \vec{y})}_{\substack{\uparrow \\ 2D \cdot 2D}} = (\vec{x} \cdot \vec{y})^2 = x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 = \underbrace{\vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{y})}_{\substack{3D \\ 3D}}$$

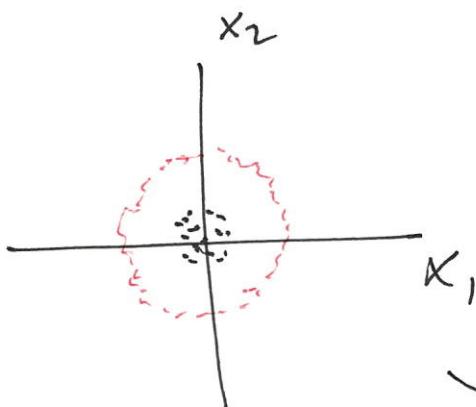
Idea : Nonlinearity and high-dimension can help with classification.



Adding dimension
can help classify



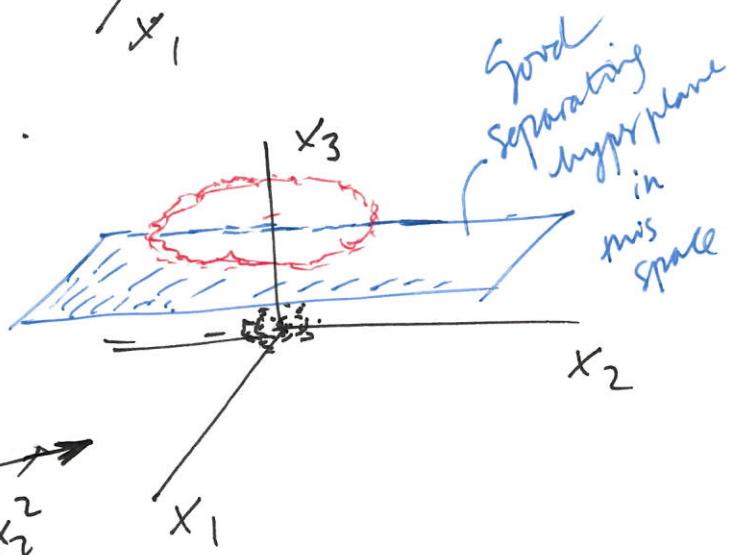
More interesting :



No separating hyperplane

Add nonlinear coordinate

$$x_3 = x_1^2 + x_2^2$$



There exists a good separating hyperplane in this space:

$$(x_1, x_2, x_1^2 + x_2^2)$$

In general, a Kernel function maps a pair of vectors in the lower dimensional space (\mathbb{R}^d) to give the dot product of transformed (nonlinear) vectors in the high-dimensional space (\mathbb{R}^D)

$$K(\vec{x}, \vec{y}) = \vec{\phi}(\vec{x}) \cdot \vec{\phi}(\vec{y}) \quad \text{where } \vec{x}, \vec{y} \in \mathbb{R}^d \text{ and } \vec{\phi}(\vec{x}), \vec{\phi}(\vec{y}) \in \mathbb{R}^D$$

K is a Kernel with associated $\vec{\phi}$ iff K is positive semi-definite

Several important Kernels :

$$(1) \quad K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + 1)^n \quad \text{Polynomial}$$

$- \|\vec{x} - \vec{y}\|^2 / \sigma^2$

$$(2) \quad K(\vec{x}, \vec{y}) = e^{-\|\vec{x} - \vec{y}\|^2 / \sigma^2} \quad \text{Gaussian Radial basis}$$

$$(3) \quad K(\vec{x}, \vec{y}) = \tanh(\beta(\vec{x} \cdot \vec{y}) + c) \quad \text{Sigmoid.}$$

choose a kernel and compute the corresponding Kernelised SVM.

More formally:

Choose
a kernel
 $k(\bar{x}, \bar{y})$

Kernelised SVM
optimization:

$\vec{z} = \phi(\bar{x})$
associated with
 $K(\bar{x}, \bar{y})$

$$\min_{\vec{w}_z} \frac{1}{2} \|\vec{w}_z\|^2$$

$$\text{subject to } 1 - y^{(i)} (\underbrace{\vec{w}_z \cdot \vec{z}^{(i)}}_{i=1, \dots, N} + b) \leq 0$$

Looking back at the SVM expression:

$$\vec{w}_z \cdot \vec{z}^{(i)} = \phi(\bar{w}) \cdot \phi(\bar{x}^{(i)}) = k(\bar{w}, \bar{x}^{(i)})$$

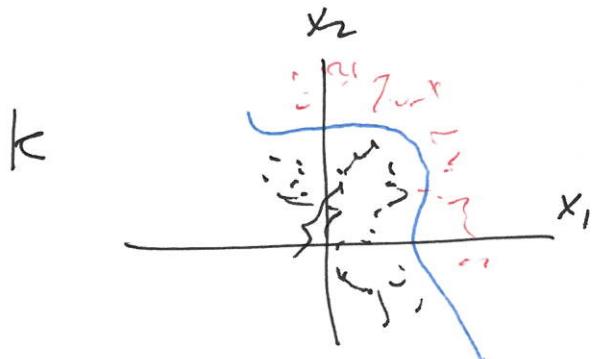
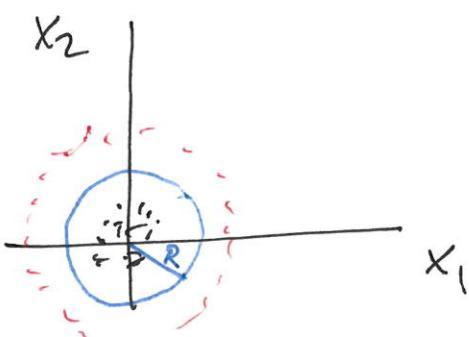
In the Lagrangian:

$$\vec{z}^{(i)} \cdot \vec{z}^{(j)} = k(\bar{x}^{(i)}, \bar{x}^{(j)})$$

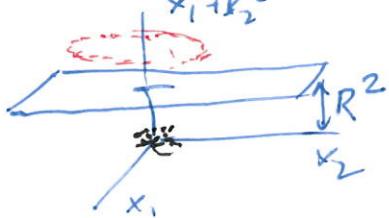
Decision:

Given \bar{x}^{in}

$$\begin{cases} k(\bar{x}^{in}, \bar{w}) + b > 0 \Rightarrow \hat{y} = +1 \\ k(\bar{x}^{in}, \bar{w}) + b < 0 \Rightarrow \hat{y} = -1 \end{cases}$$



this is equivalent
to



The decision boundary is
nonlinear in the \vec{x} space

Artificial neural networks

Robert L. Peach

- Why deep learning?
- **Multi-layer perceptrons (MLP)**
 - The Perceptron
 - Activation functions
 - Forward propagation
 - Loss functions
 - Back propagation
 - Regularisation in ANNs
- **Convolutional neural networks (CNNs)**
 - Image classification / object identification
 - Convolutions
 - Pooling
- **Recurrent neural networks (RNNs)**
 - Forecasting
 - Increasing memory – LSTMs, Gated RNNs

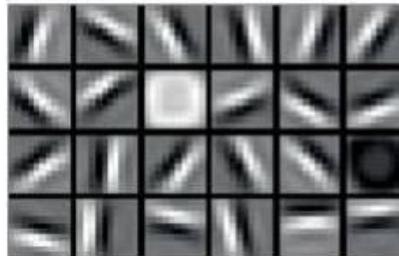
Deep Learning

Hand engineered features are time consuming, brittle and not scalable in practice
Can we learn the **underlying features** directly from the data?

Millions of images



Low level features



Lines & Edges

Mid level features



Eyes & Nose & Ears

High level features

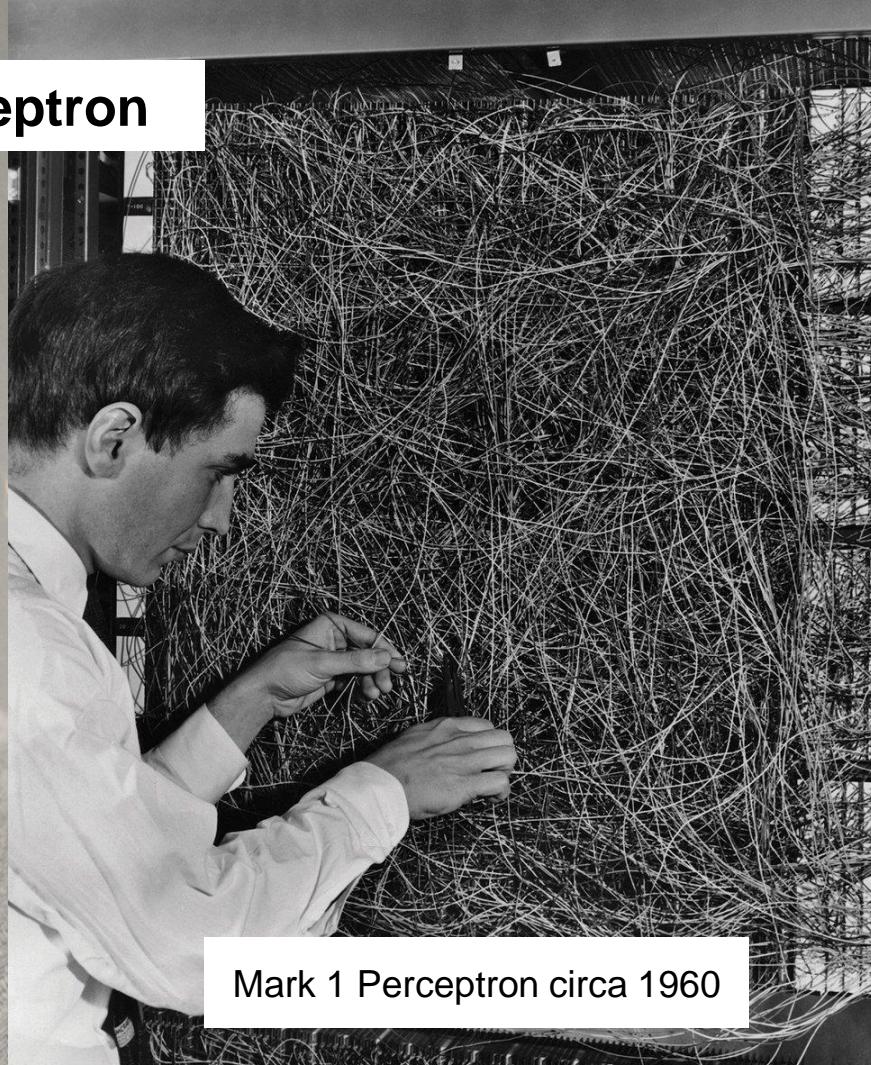


Facial Structure

Perceptron

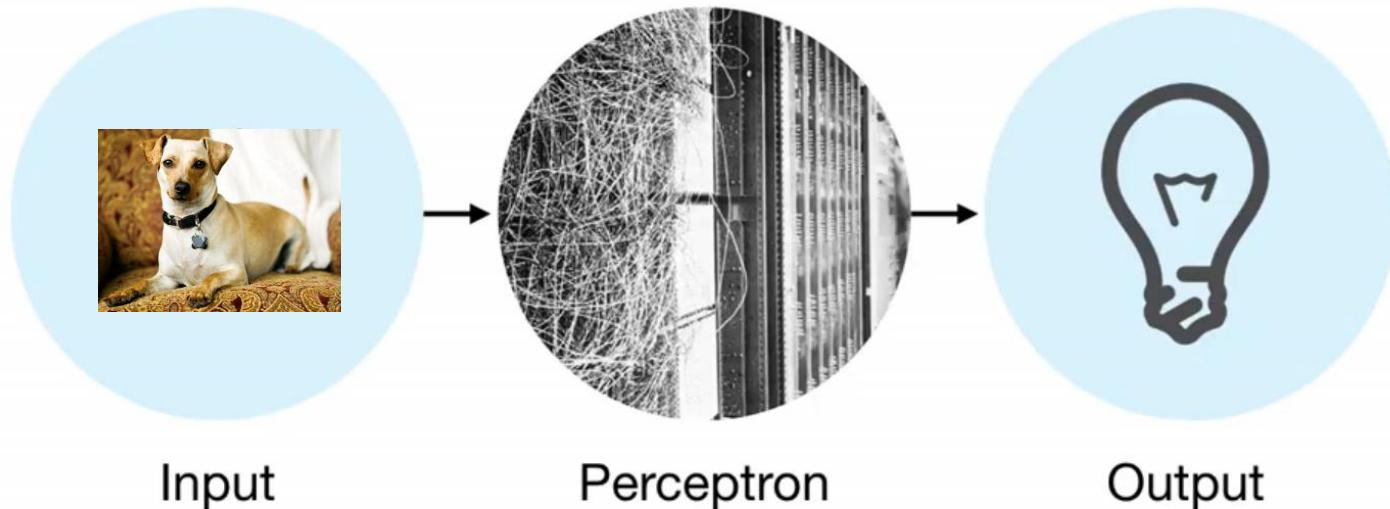


Frank Rosenblatt

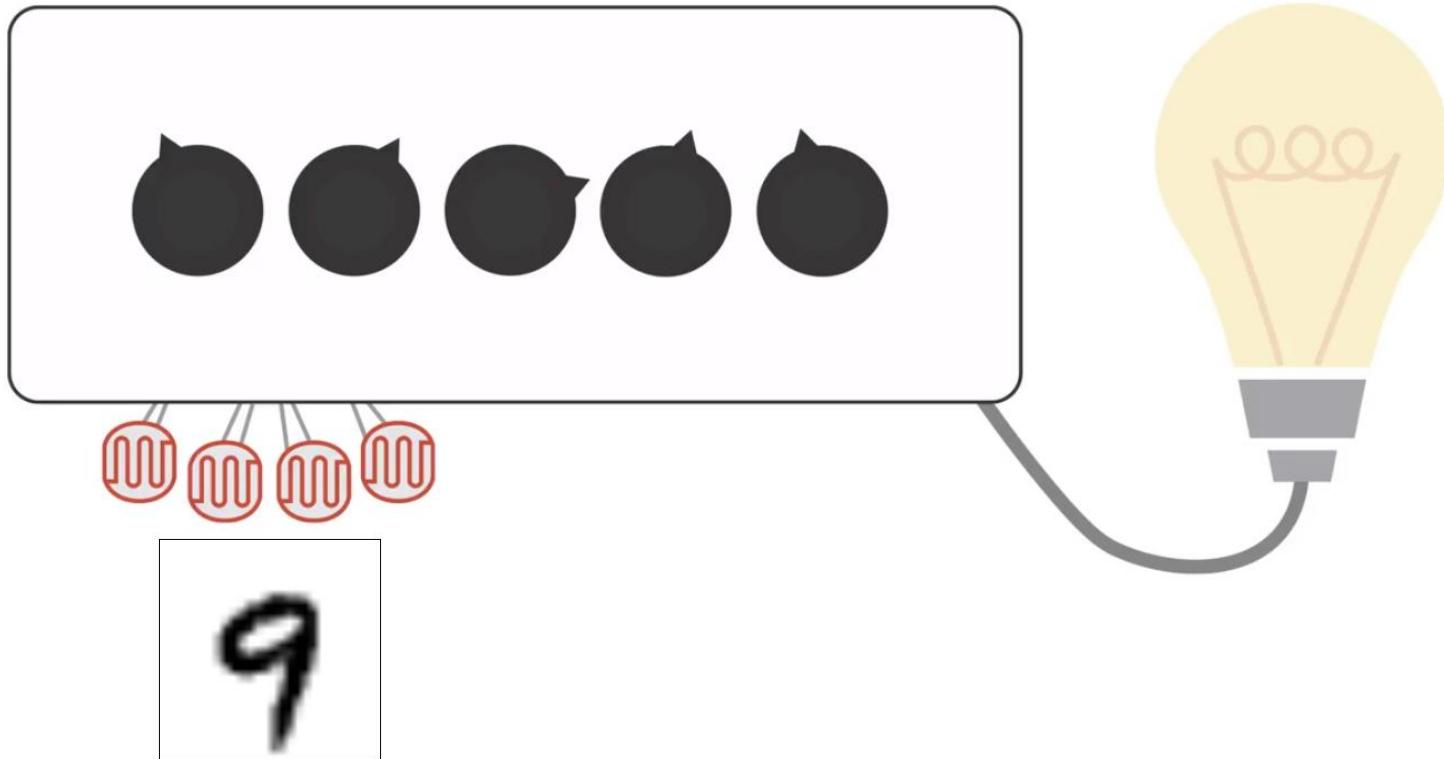


Mark 1 Perceptron circa 1960

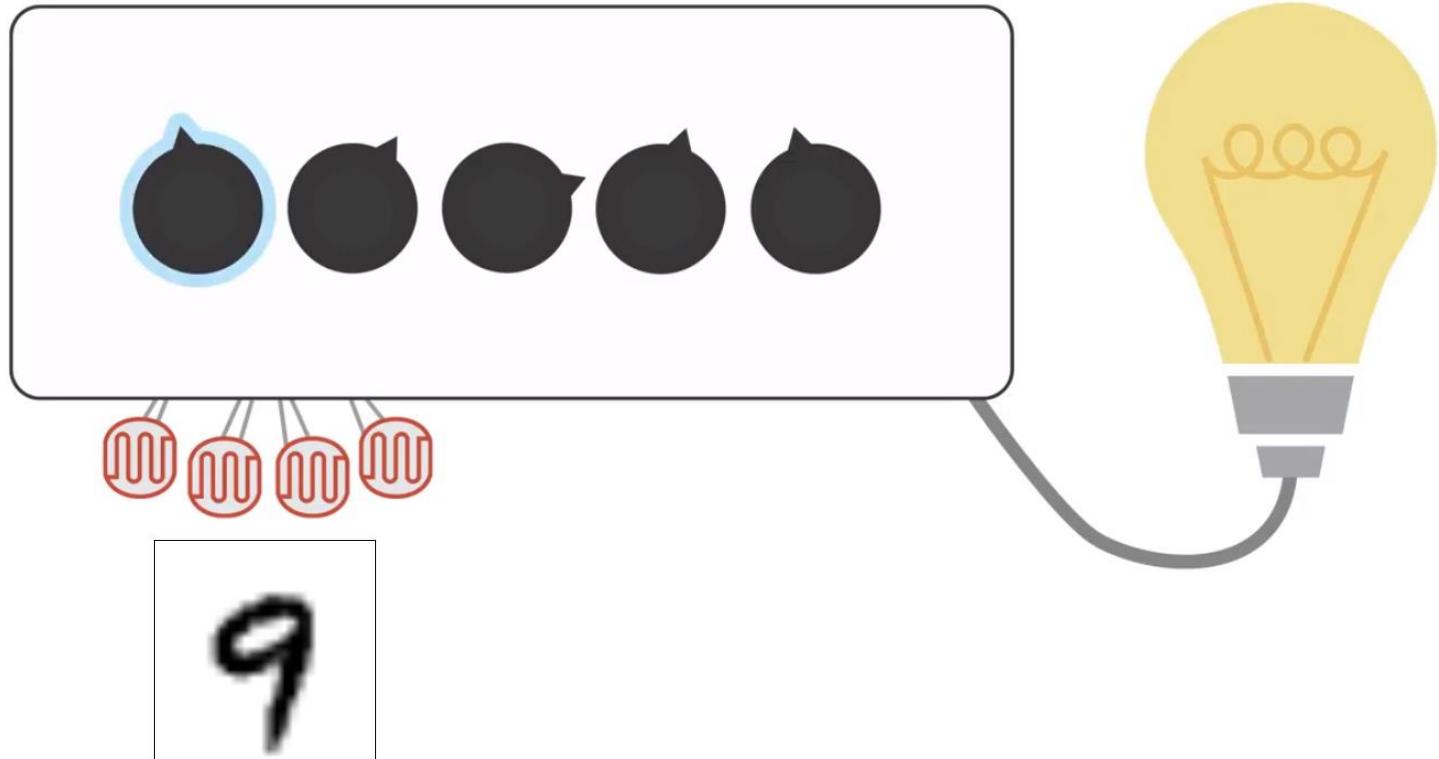
Rosenblatt's Machine



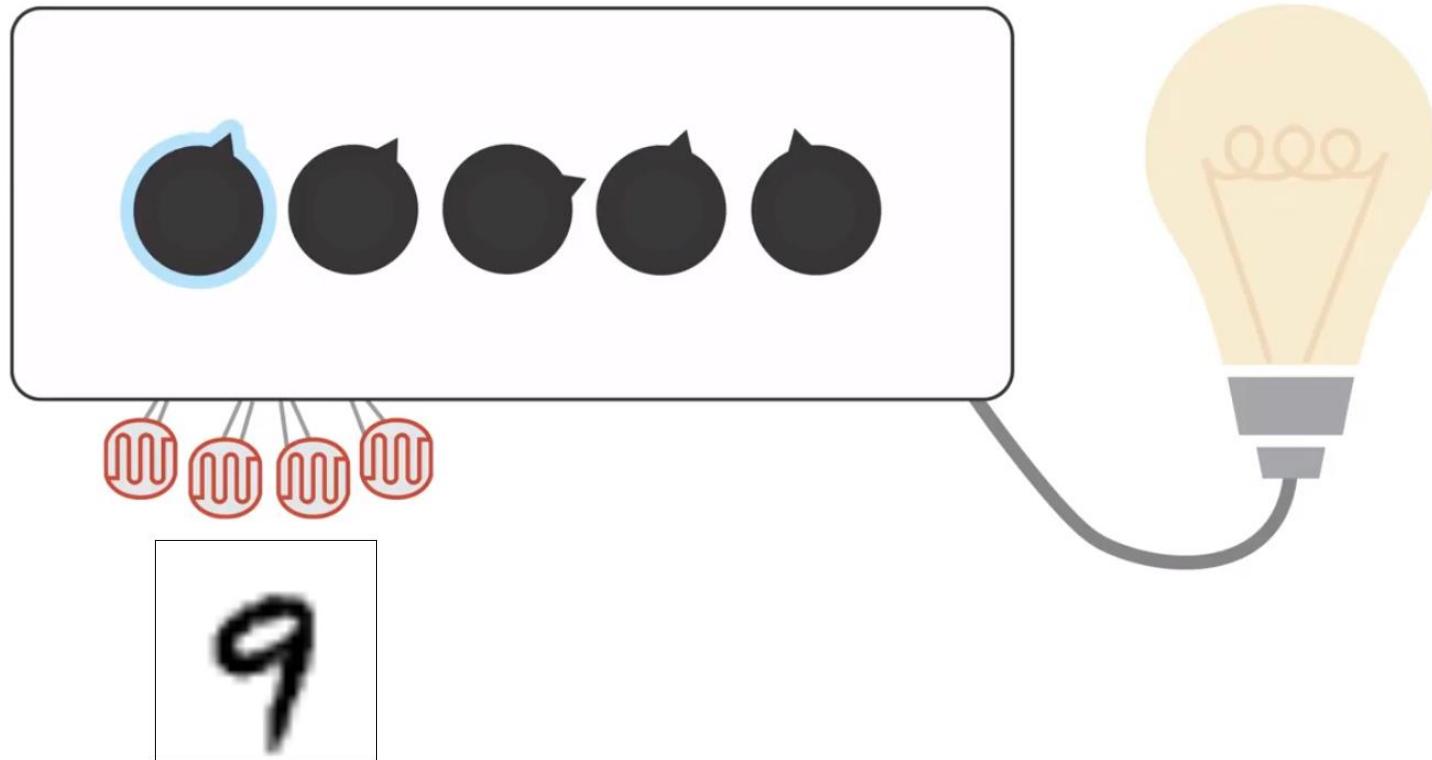
Rosenblatt's Machine



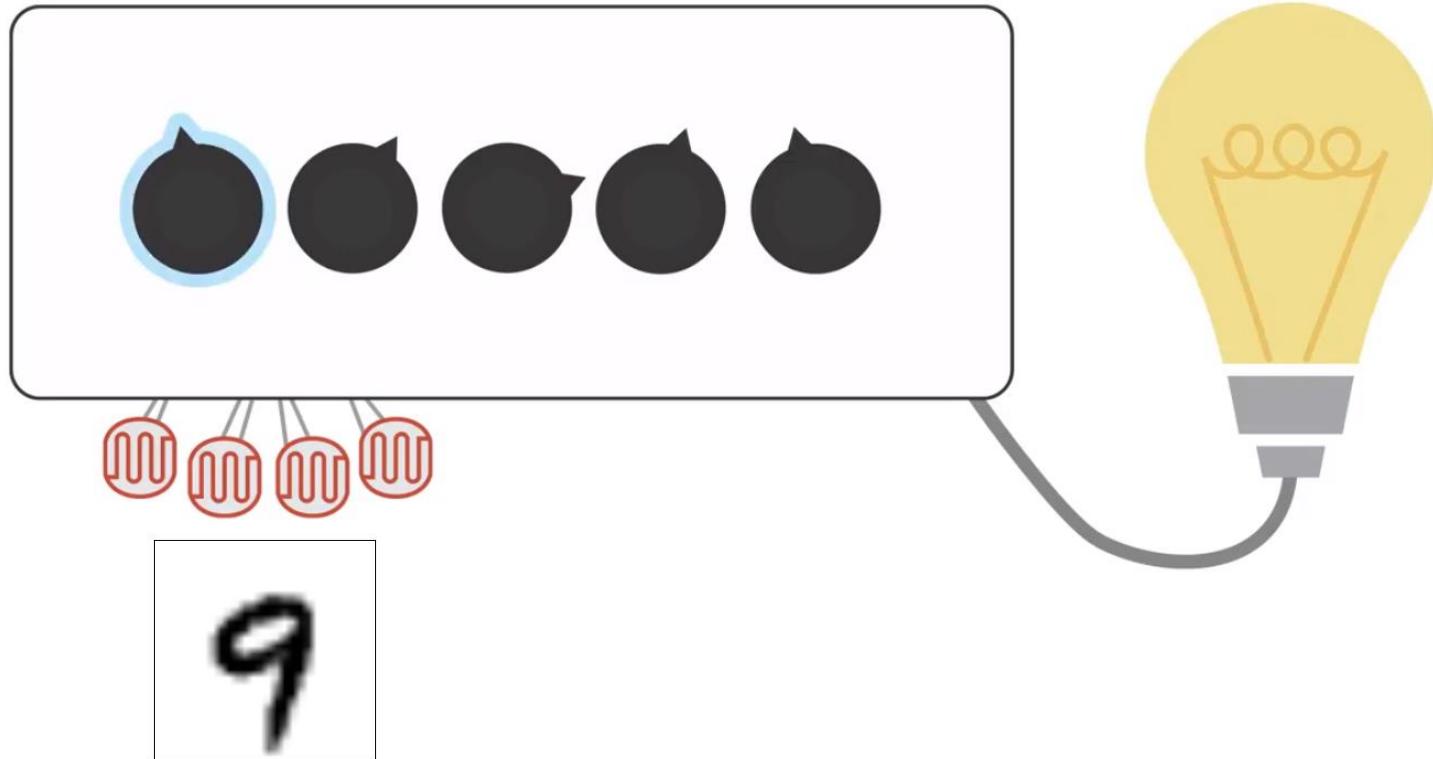
Rosenblatt's Machine



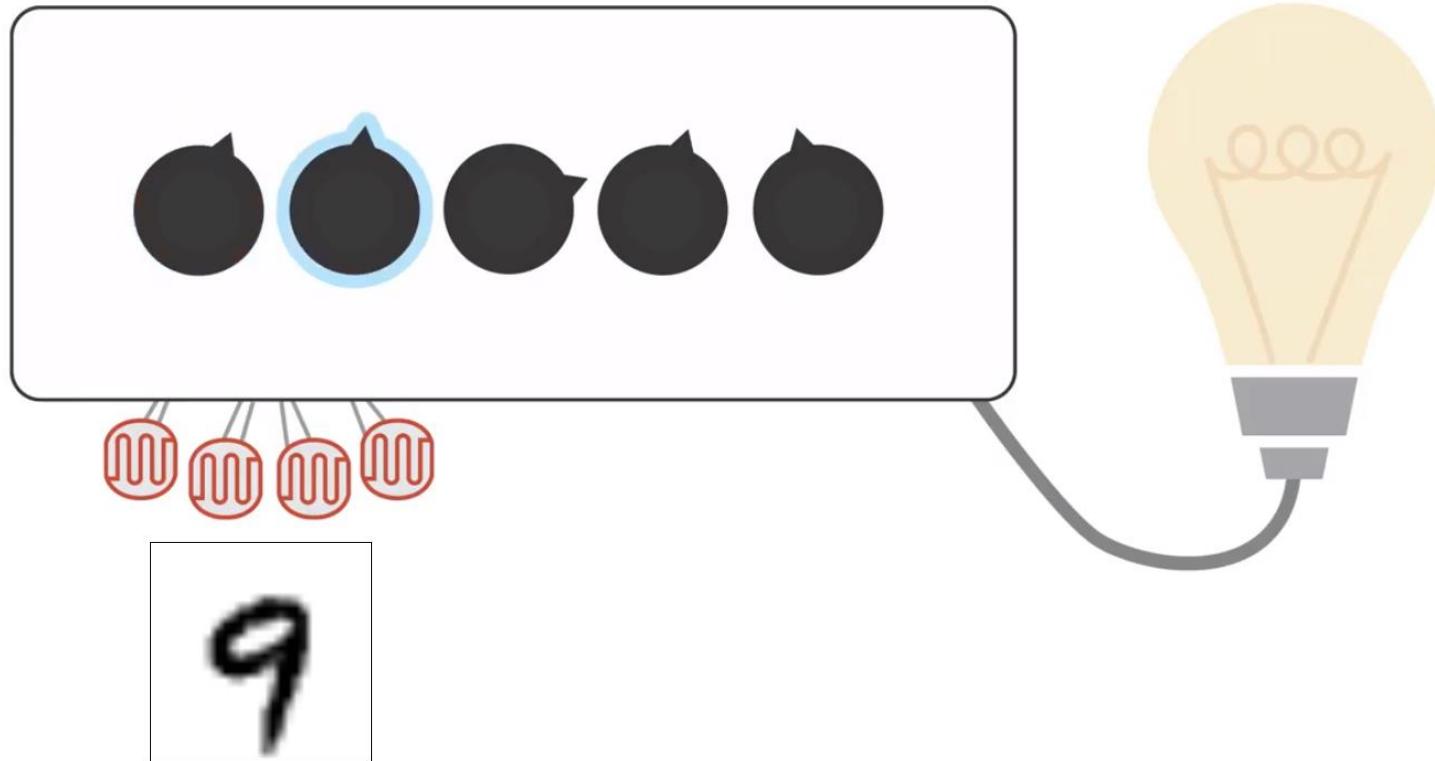
Rosenblatt's Machine



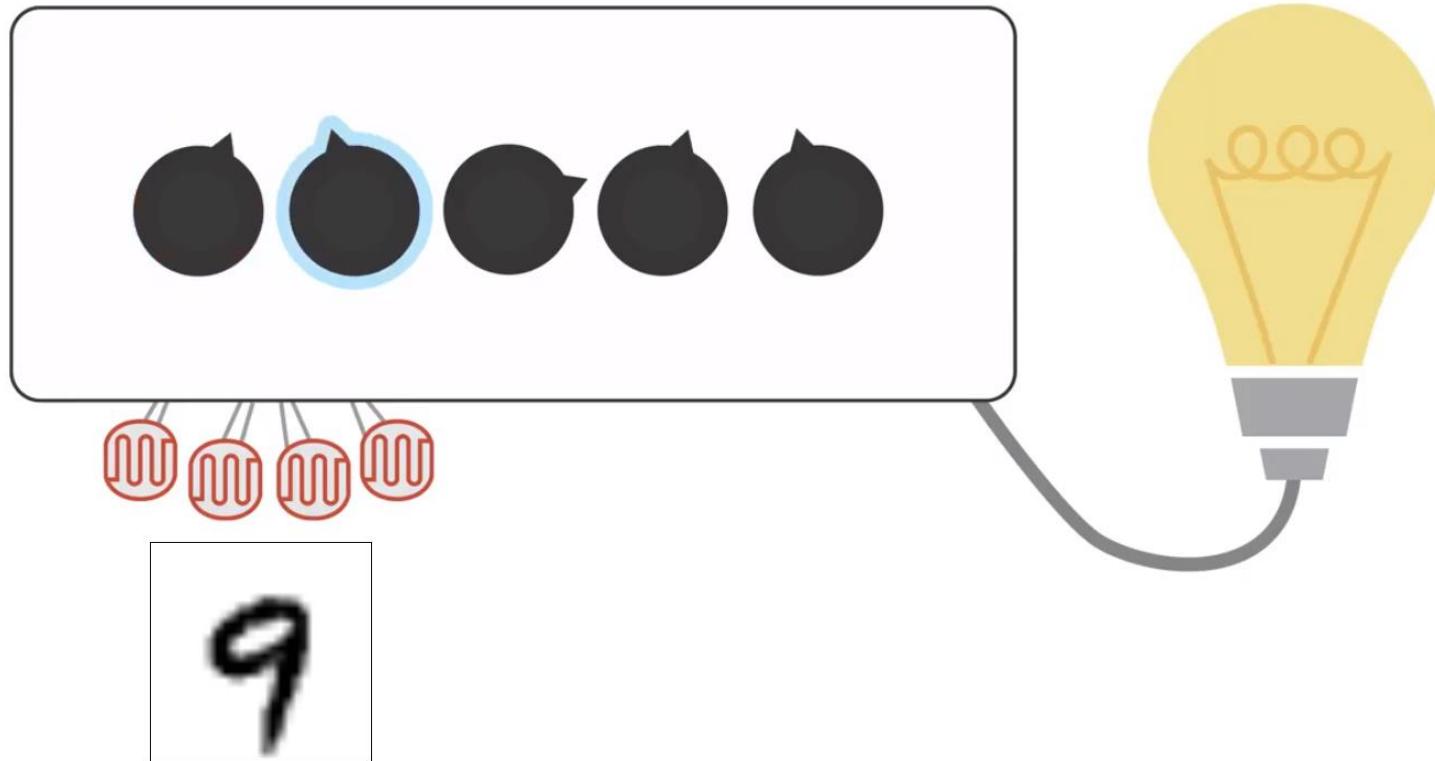
Rosenblatt's Machine



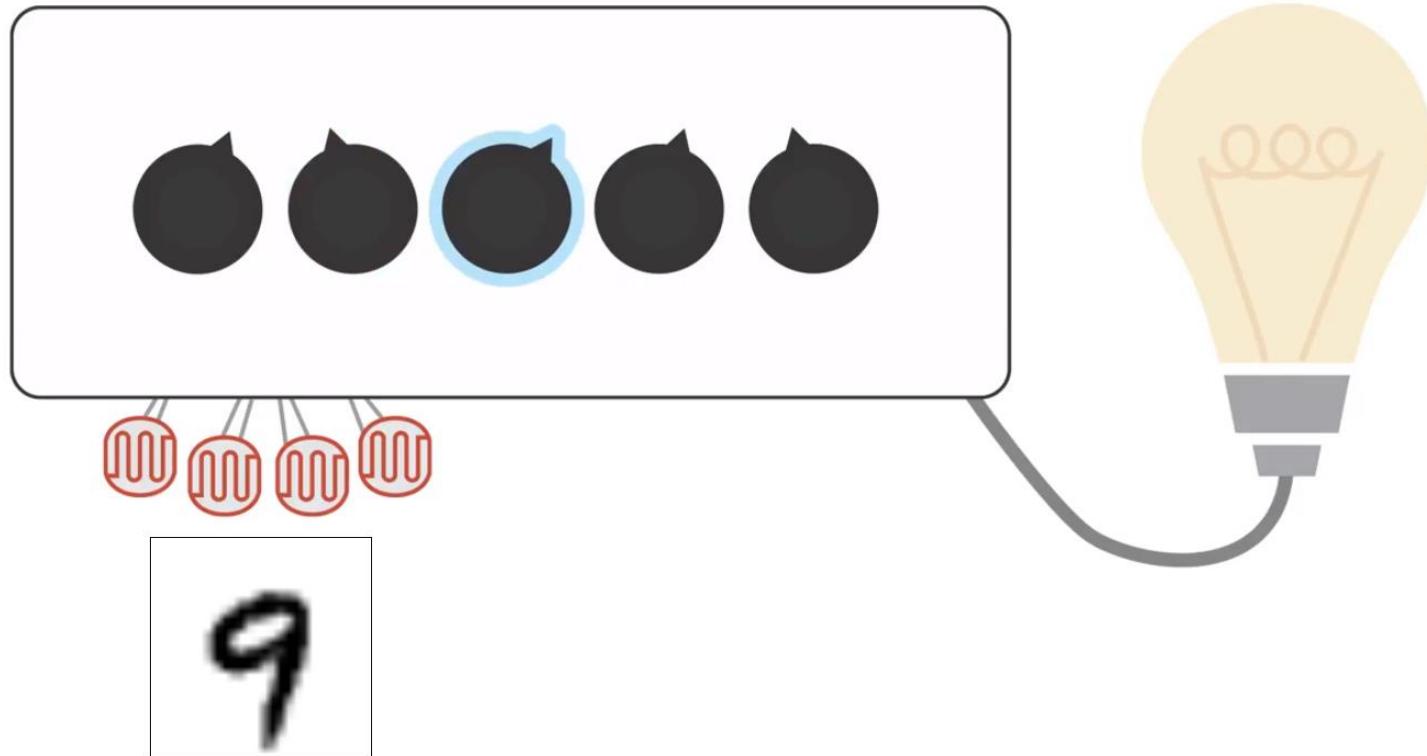
Rosenblatt's Machine



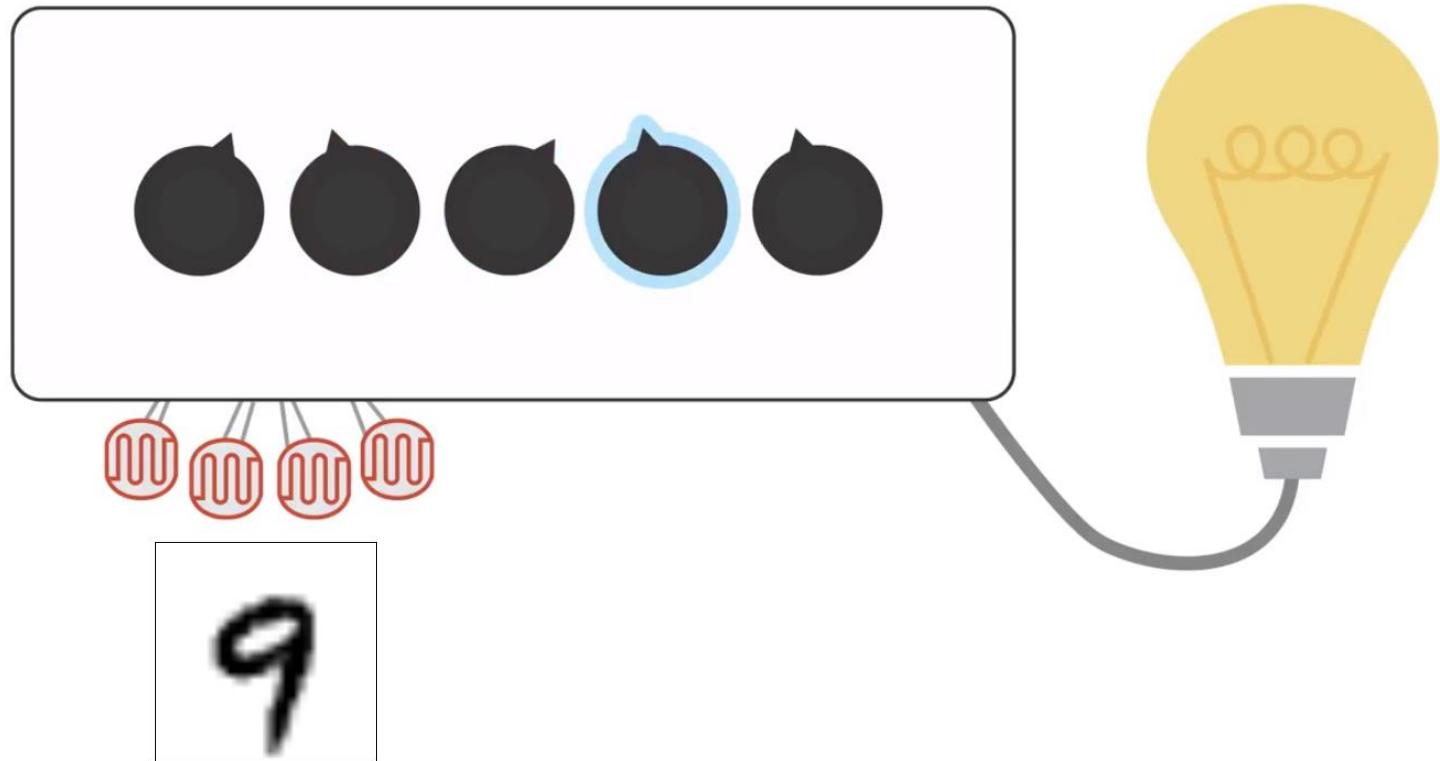
Rosenblatt's Machine



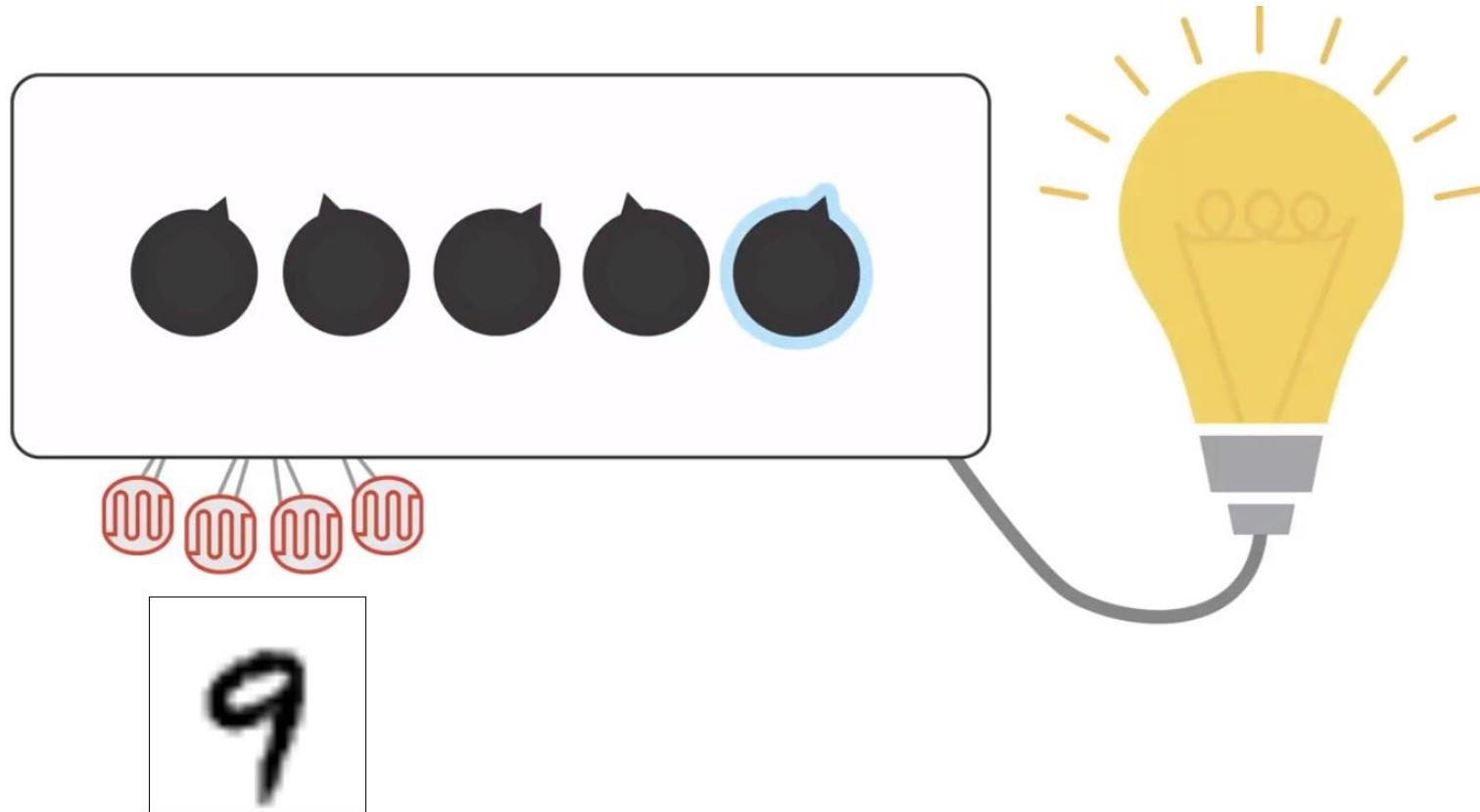
Rosenblatt's Machine



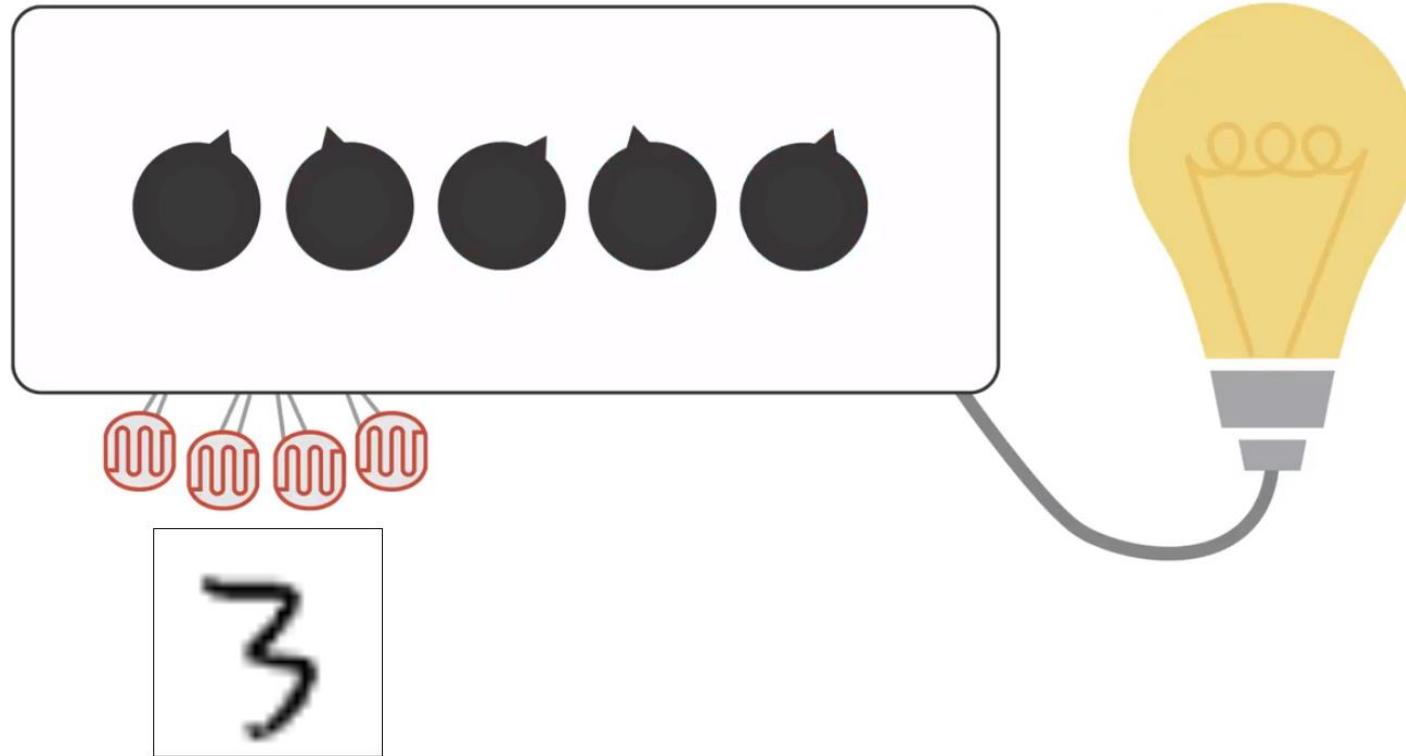
Rosenblatt's Machine



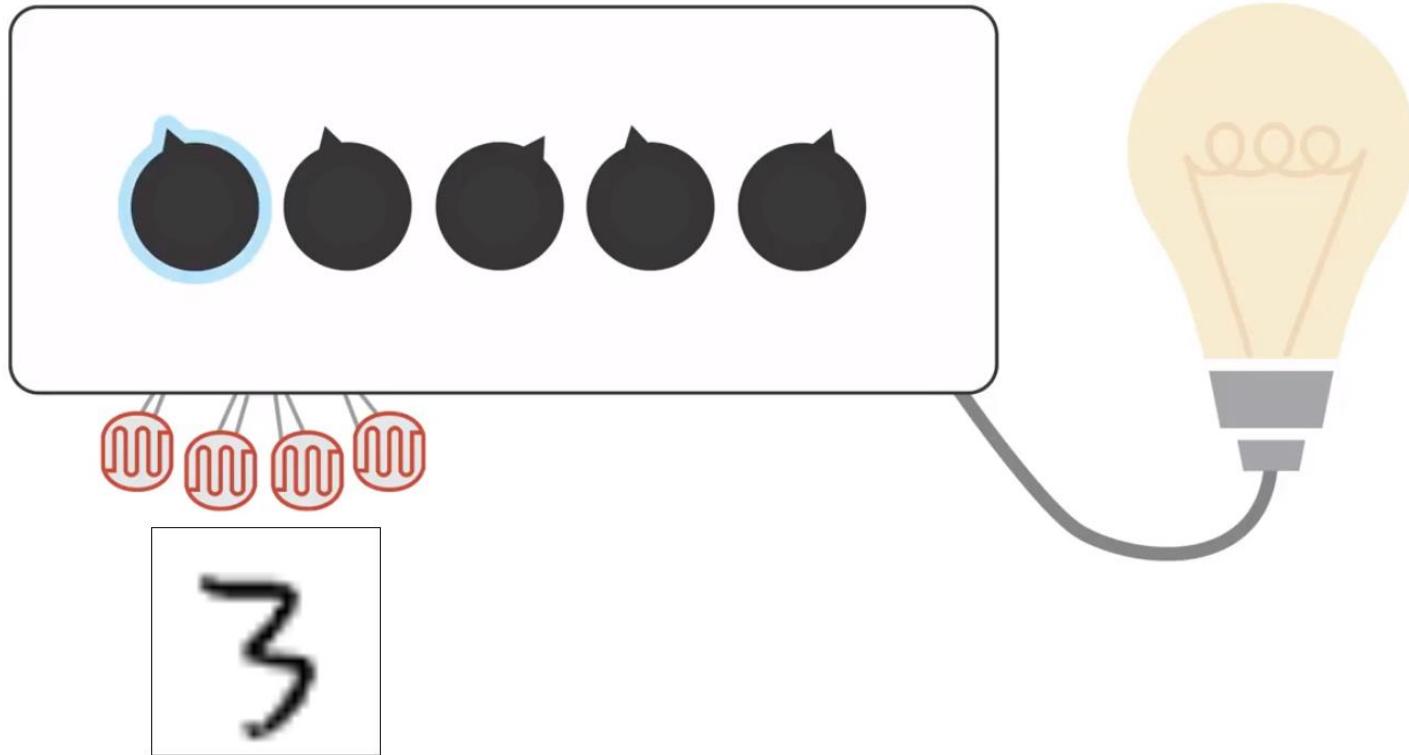
Rosenblatt's Machine



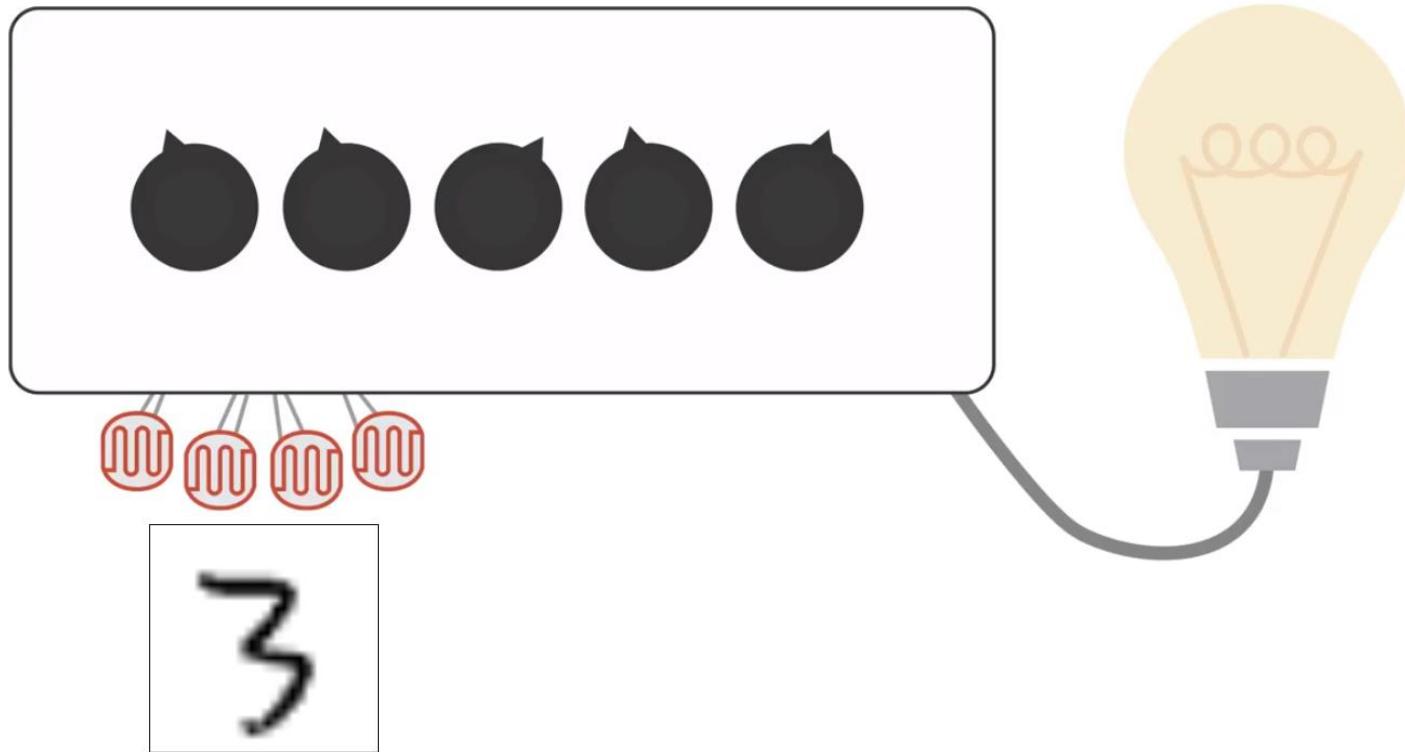
Rosenblatt's Machine



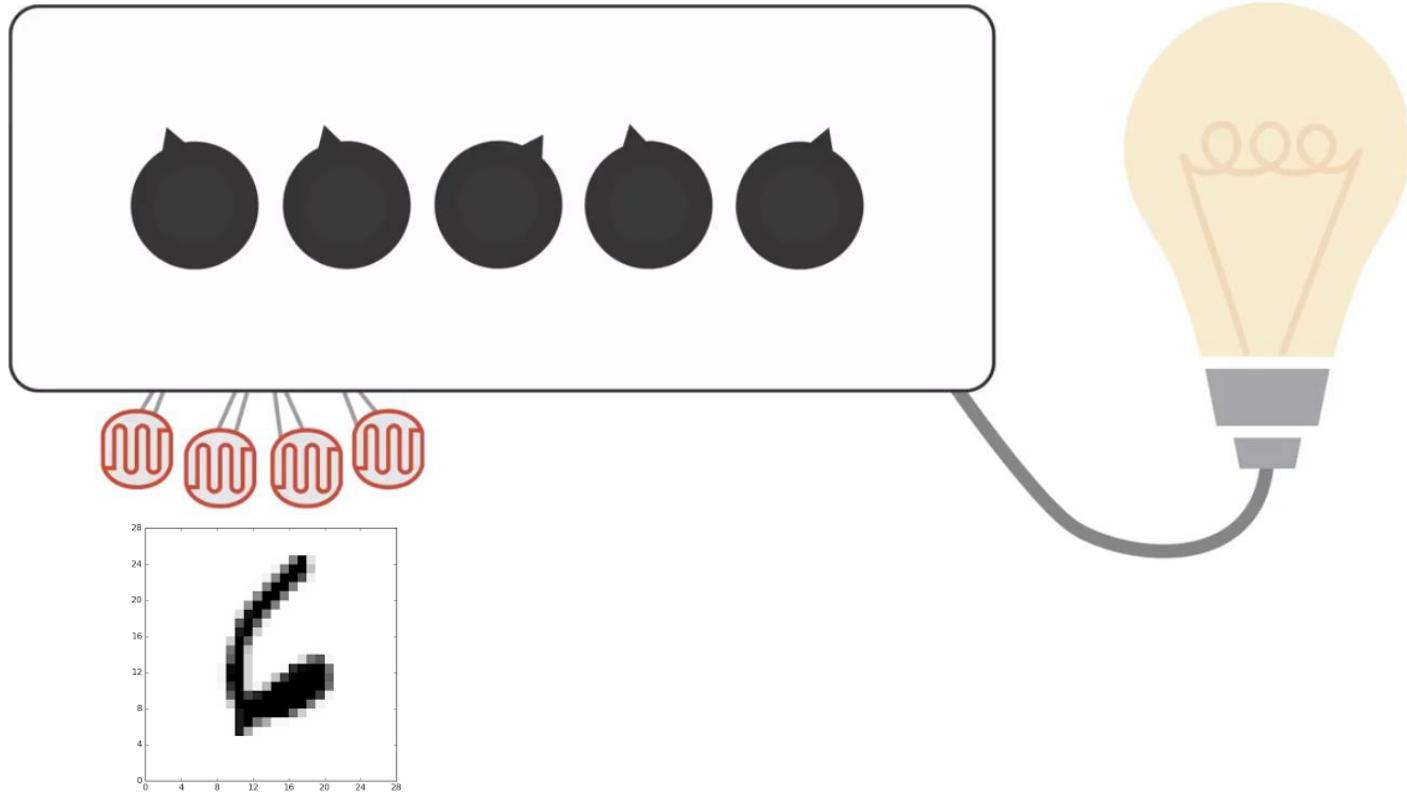
Rosenblatt's Machine



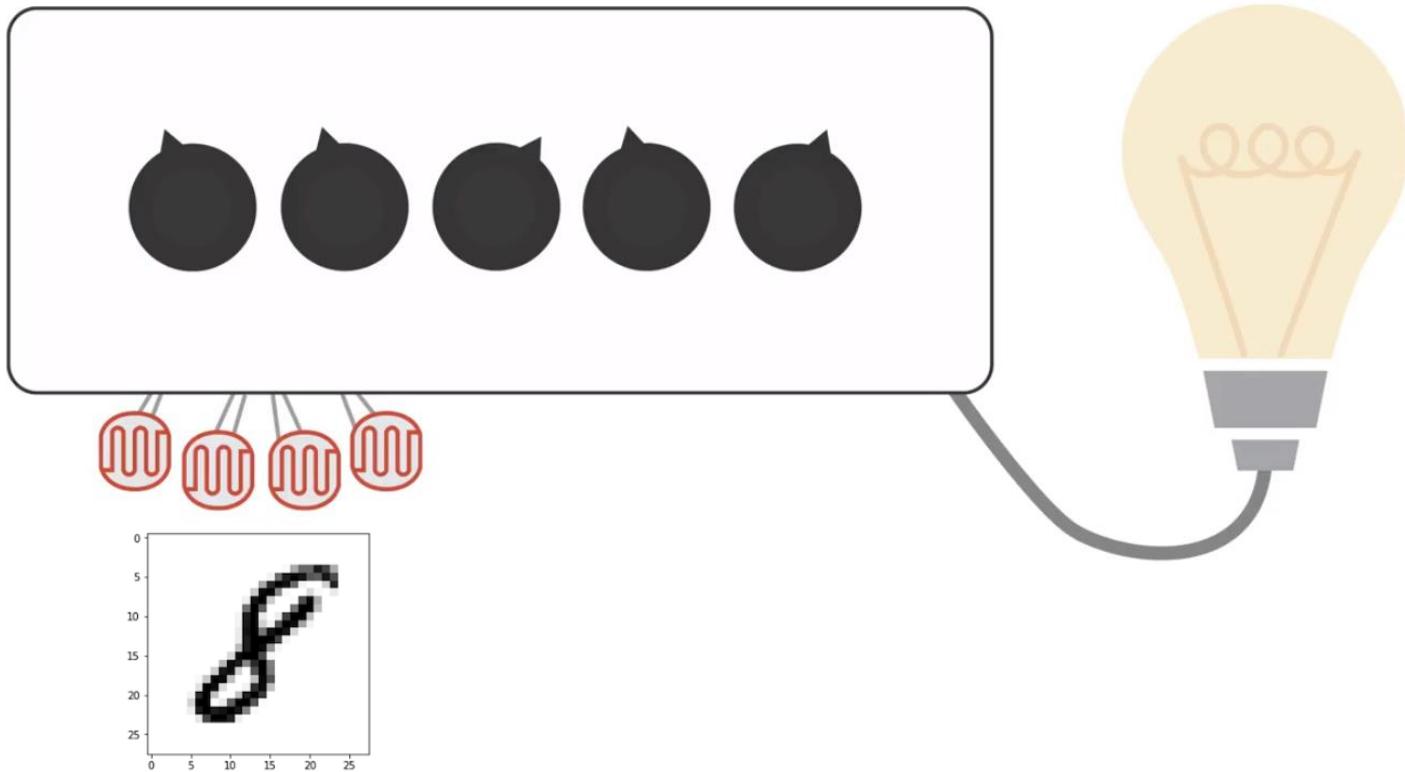
Rosenblatt's Machine



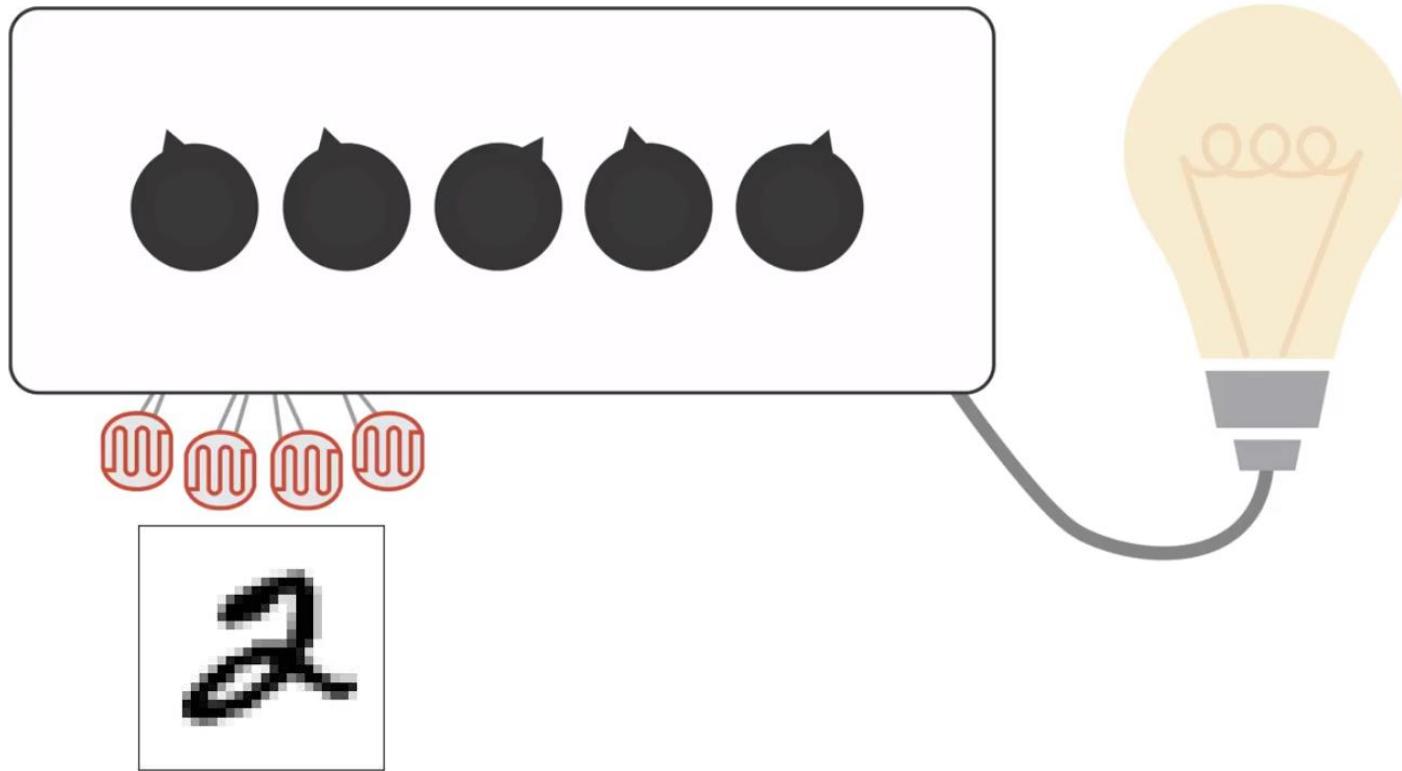
Rosenblatt's Machine



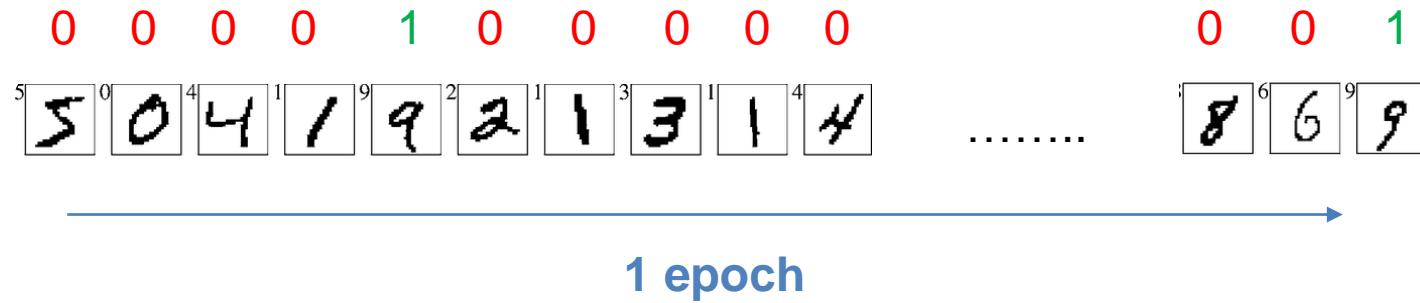
Rosenblatt's Machine



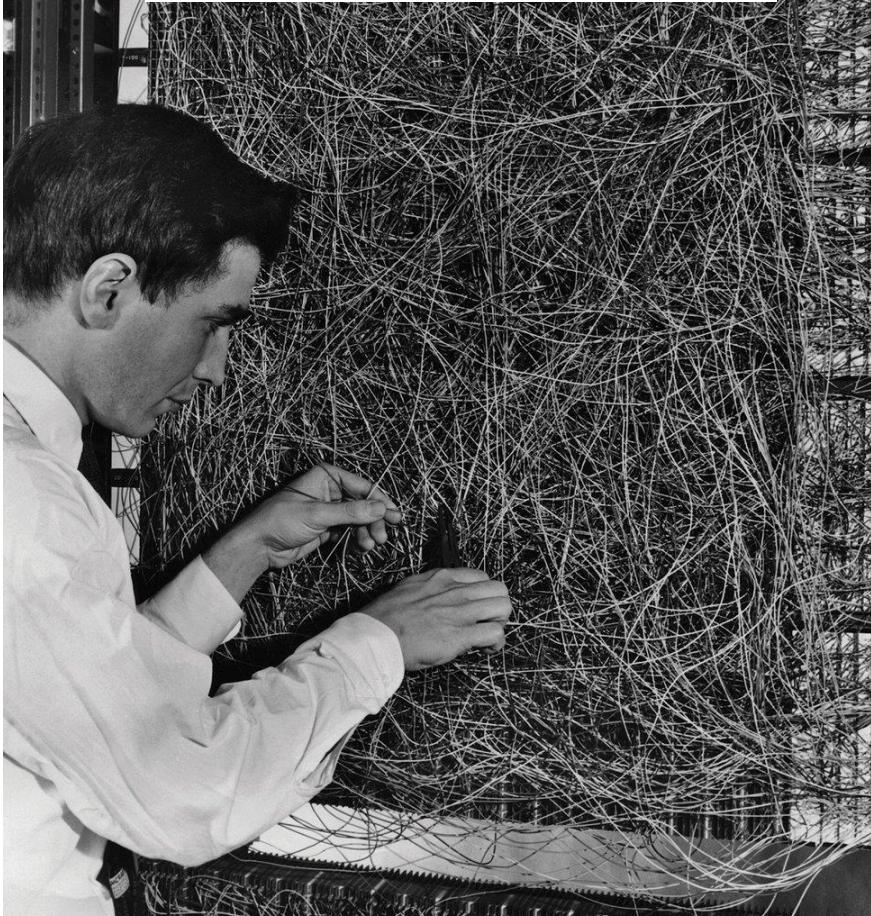
Rosenblatt's Machine



Training Data



Tuning the Perceptron



We are thinking less about machines and more about algorithms...

The **perceptron** is the fundamental building block of deep learning.

Also known as a **neuron** in deep learning.

Why now?

Neural networks have existed for decades. Why do we care now?

Big data

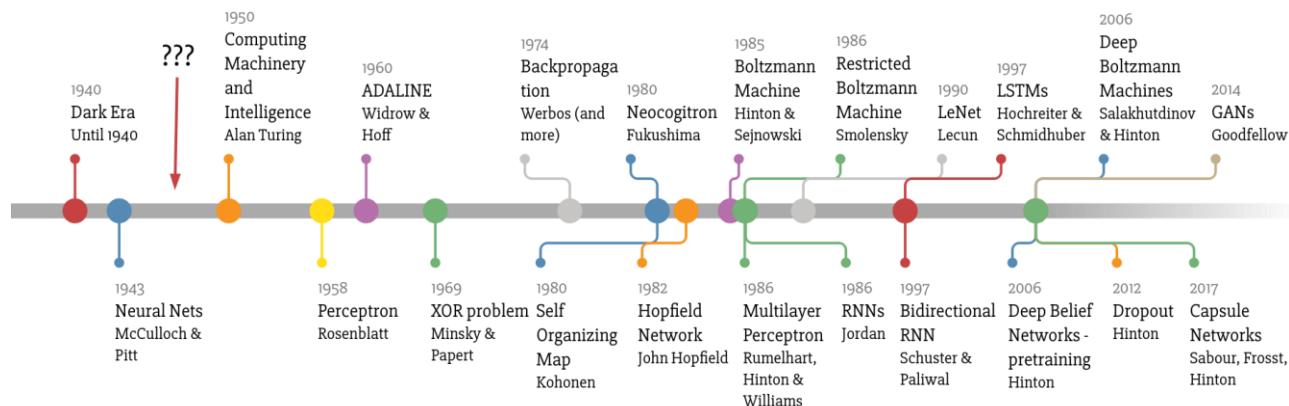
- Large datasets
- Easier collection and storage
- More platforms for collecting data

Hardware

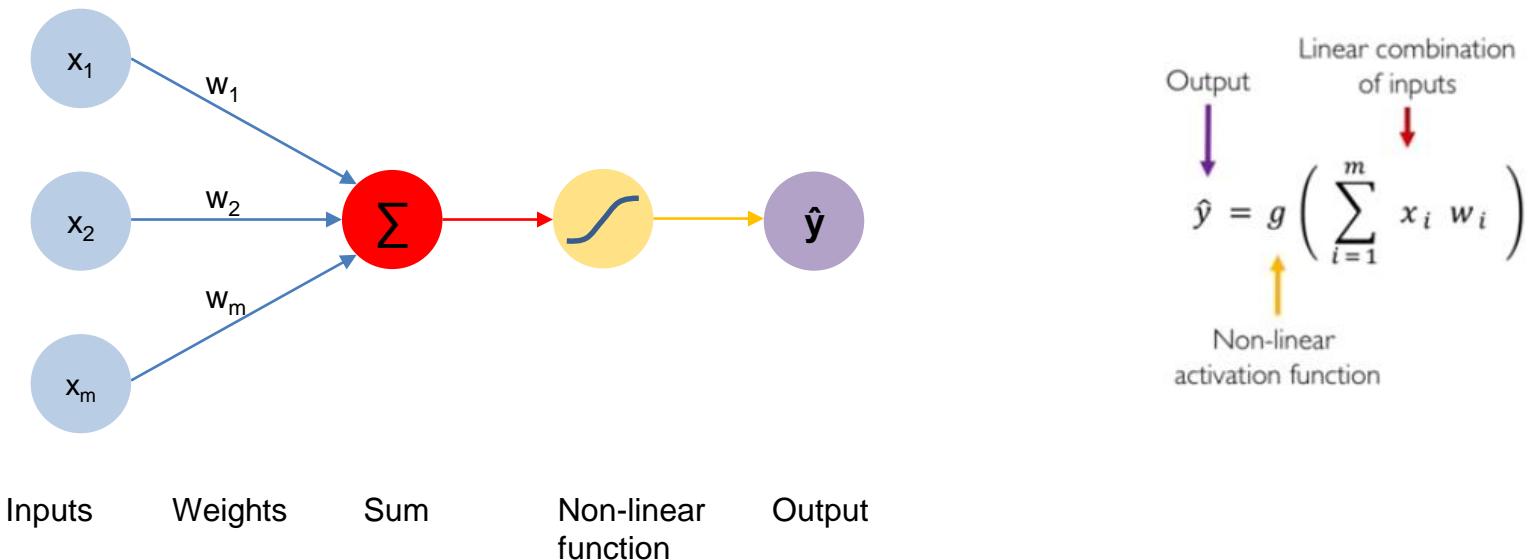
- Graphics processing units (GPUs)
- Massively parallelizable calculations

Software

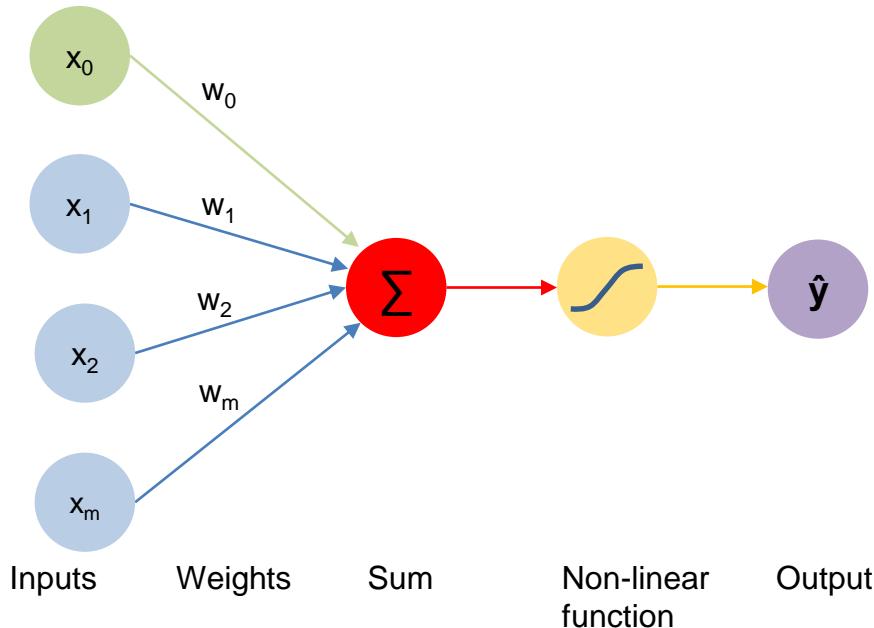
- Improved mathematical architectures
- Efficient and open source toolboxes



The perceptron: Forward propagation



The perceptron: Forward propagation



Linear combination of inputs

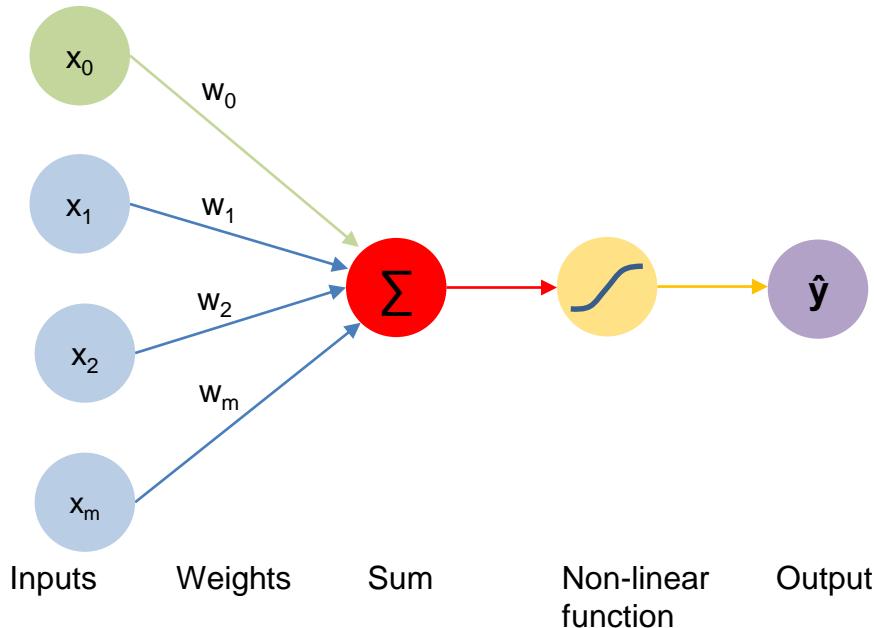
Output

Non-linear activation function

Bias

$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

The perceptron: Forward propagation

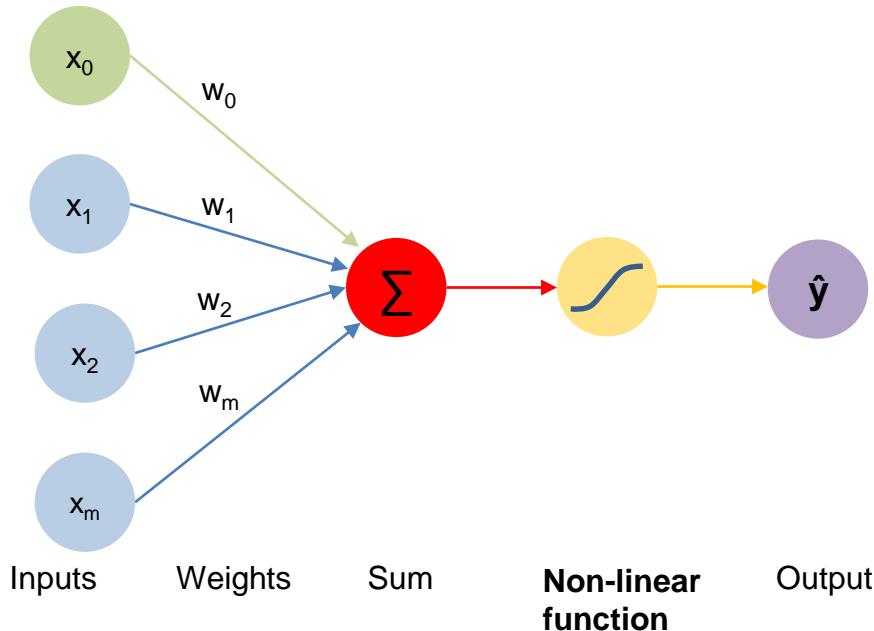


$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

$$\hat{y} = g (w_0 + \mathbf{X}^T \mathbf{W})$$

where: $\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

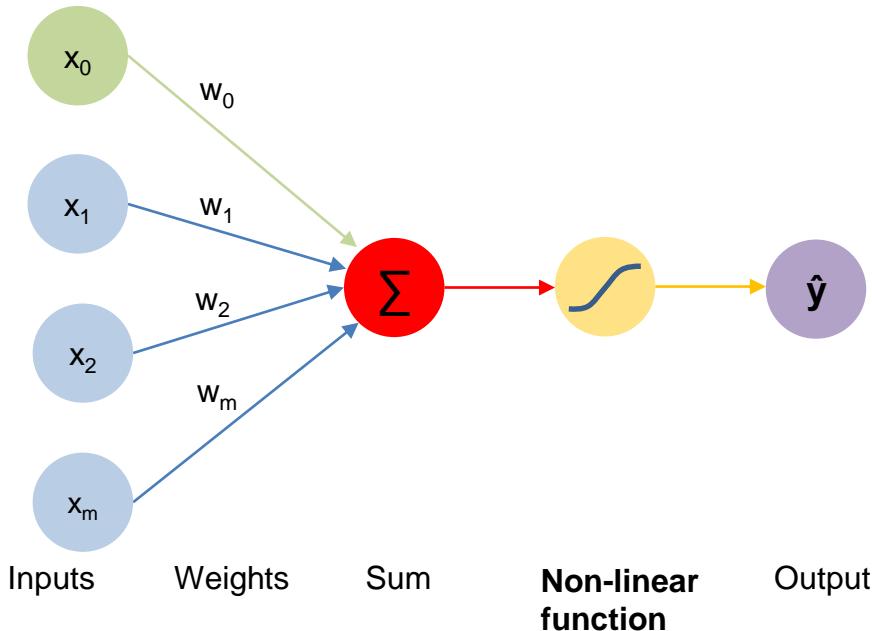
The perceptron: Forward propagation



Activation Functions

$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

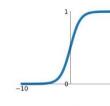
The perceptron: Forward propagation



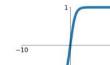
Activation Functions

$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

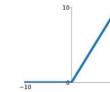
Sigmoid
 $\sigma(x) = \frac{1}{1+e^{-x}}$



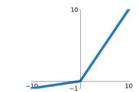
tanh
 $\tanh(x)$



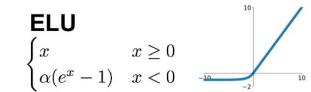
ReLU
 $\max(0, x)$



Leaky ReLU
 $\max(0.1x, x)$

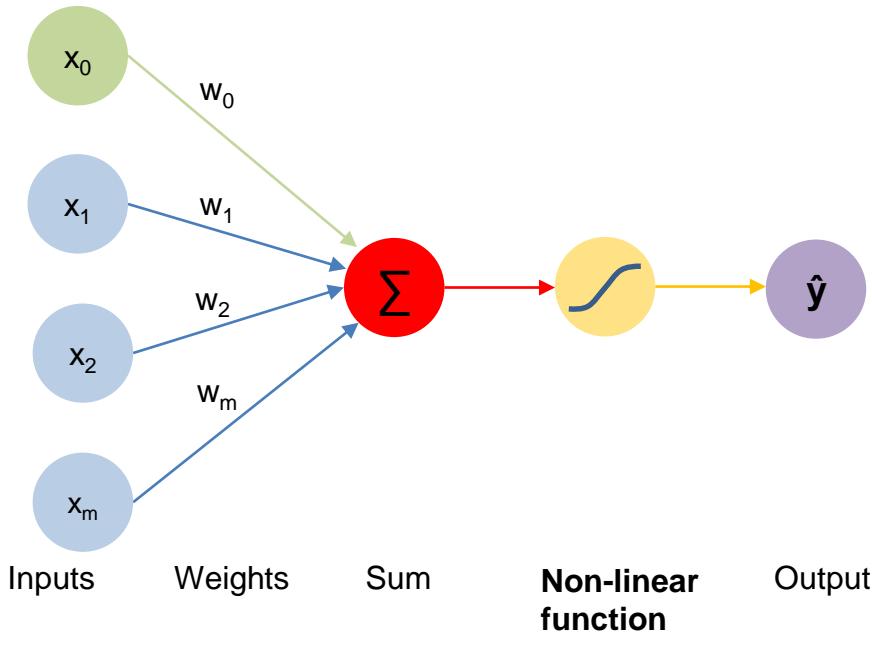


Maxout
 $\max(w_1^T x + b_1, w_2^T x + b_2)$



ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

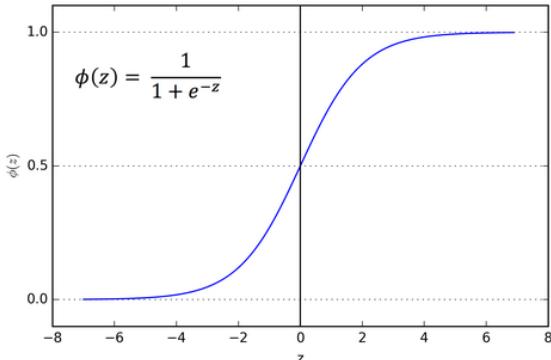
The perceptron: Forward propagation



Activation Functions

$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

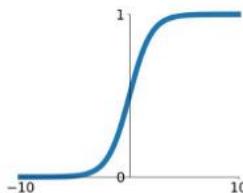
Example: Logistic function



The perceptron: Forward propagation

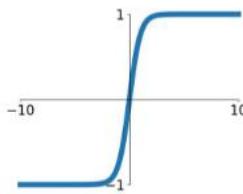
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



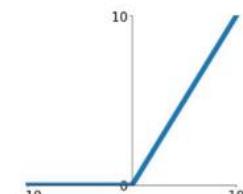
tanh

$$\tanh(x)$$



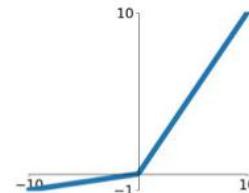
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

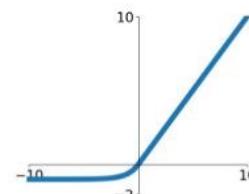


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

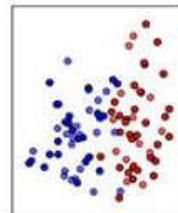
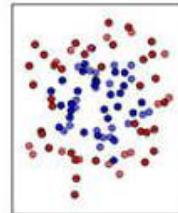
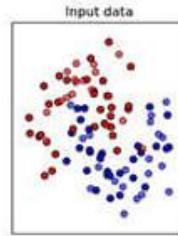


Why use activation functions?

Activation functions introduce *non-linearities* into the network

Linear activation functions
produce linear decisions.

Non-linearities allow us to
approximate arbitrarily
complex functions

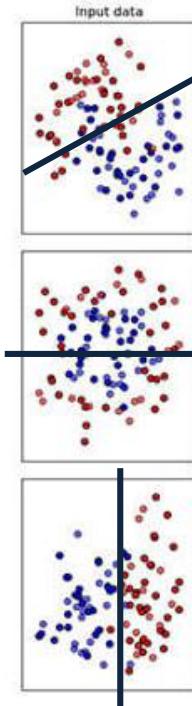


Why use activation functions?

Activation functions introduce *non-linearities* into the network

Linear activation functions produce linear decisions.

Non-linearities allow us to approximate arbitrarily complex functions

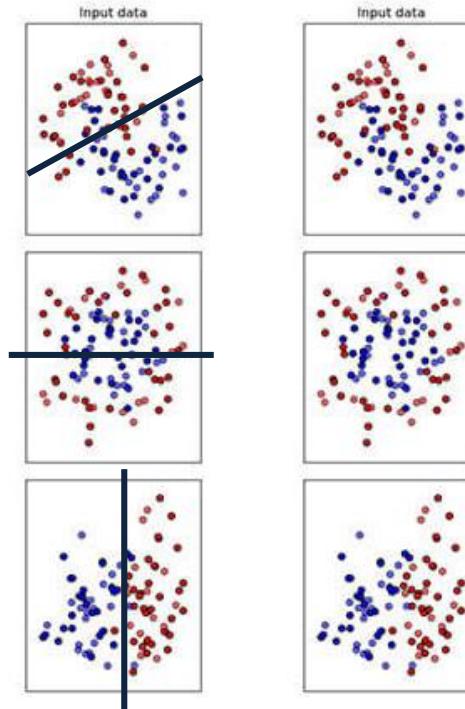


Why use activation functions?

Activation functions introduce *non-linearities* into the network

Linear activation functions produce linear decisions.

Non-linearities allow us to approximate arbitrarily complex functions

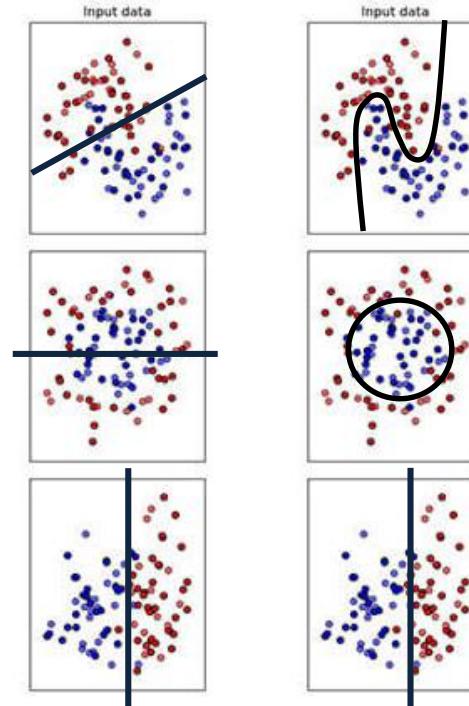


Why use activation functions?

Activation functions introduce *non-linearities* into the network

Linear activation functions produce linear decisions.

Non-linearities allow us to approximate arbitrarily complex functions

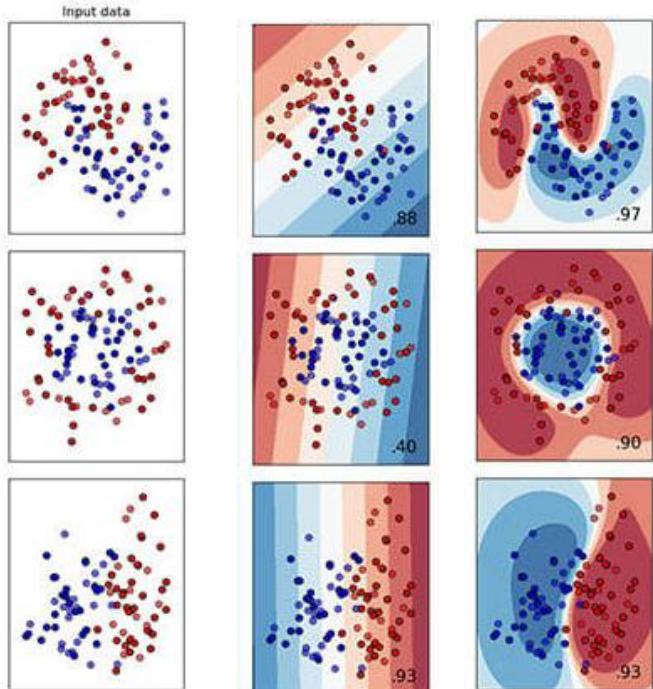


Why use activation functions?

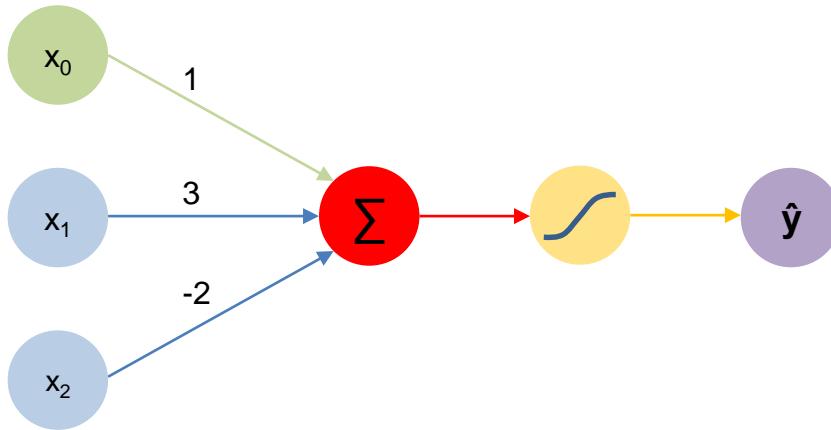
Activation functions introduce *non-linearities* into the network

Linear activation functions produce linear decisions.

Non-linearities allow us to approximate arbitrarily complex functions



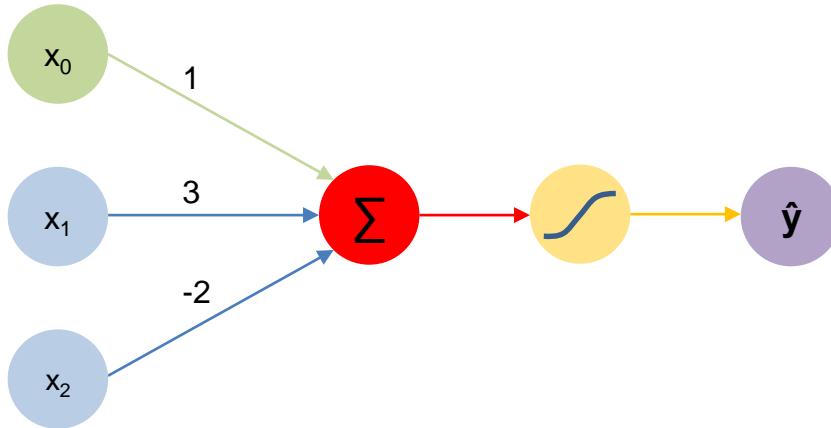
The perceptron: Forward propagation



We have: $w_0 = 1$ and $\mathbf{w} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(w_0 + \mathbf{X}^T \mathbf{w}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g(1 + 3x_1 - 2x_2)\end{aligned}$$

The perceptron: Forward propagation



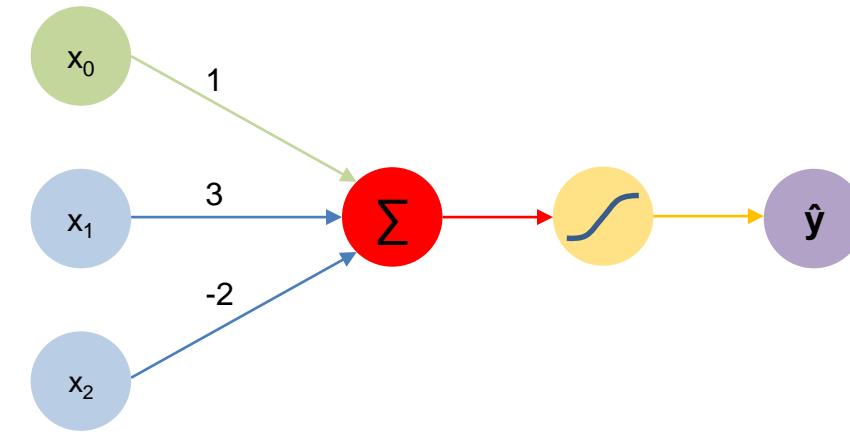
We have: $w_0 = 1$ and $\mathbf{w} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(w_0 + \mathbf{X}^T \mathbf{w}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g\left(1 + 3x_1 - 2x_2\right)\end{aligned}$$

This is just a line in 2D!

Inputs Weights Sum Non-linear
function Output

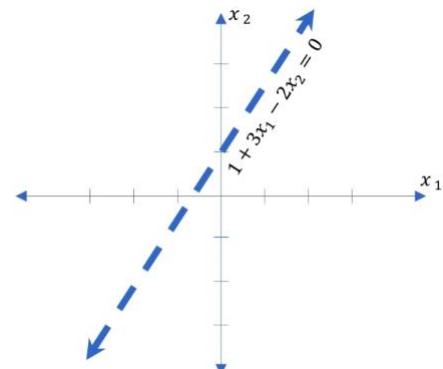
The perceptron: Forward propagation



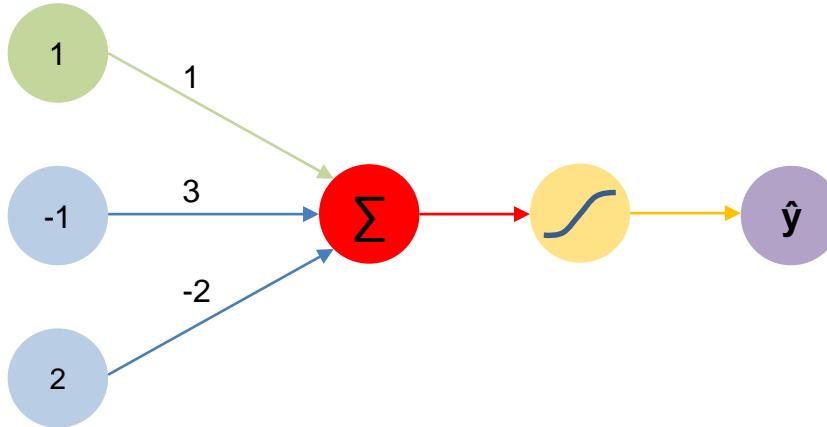
We have: $w_0 = 1$ and $\mathbf{w} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(w_0 + \mathbf{X}^T \mathbf{w}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g\left(1 + 3x_1 - 2x_2\right)\end{aligned}$$

This is just a line in 2D!



The perceptron: Forward propagation



Inputs	Weights	Sum	Non-linear function	Output
--------	---------	-----	---------------------	--------

$$X = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

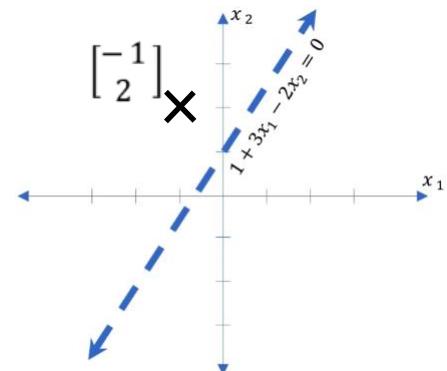
$$\begin{aligned}\hat{y} &= g(1 + (3 * -1) - (2 * 2)) \\ &= g(-6) \approx 0.002\end{aligned}$$

g = Logistic function

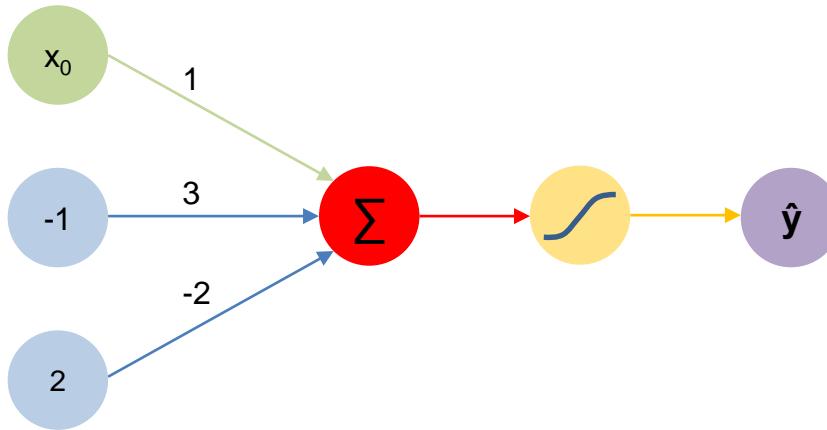
We have: $w_0 = 1$ and $W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(w_0 + X^T W) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g\left(1 + 3x_1 - 2x_2\right)\end{aligned}$$

This is just a line in 2D!



The perceptron: Forward propagation



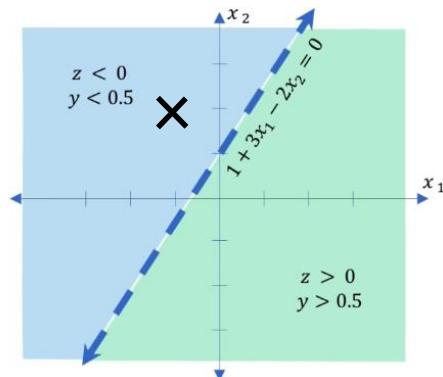
Inputs	Weights	Sum	Non-linear function	Output
$X = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$			$\hat{y} = g(1 + (3 * -1) - (2 * 2))$ $= g(-6) \approx 0.002$	

g = Logistic function

We have: $w_0 = 1$ and $W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(w_0 + X^T W) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g(1 + 3x_1 - 2x_2)\end{aligned}$$

This is just a line in 2D!



The perceptron: 3 steps

Take a dot product → Add a Bias → Take a non-linearity

Output

Linear combination
of inputs

$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

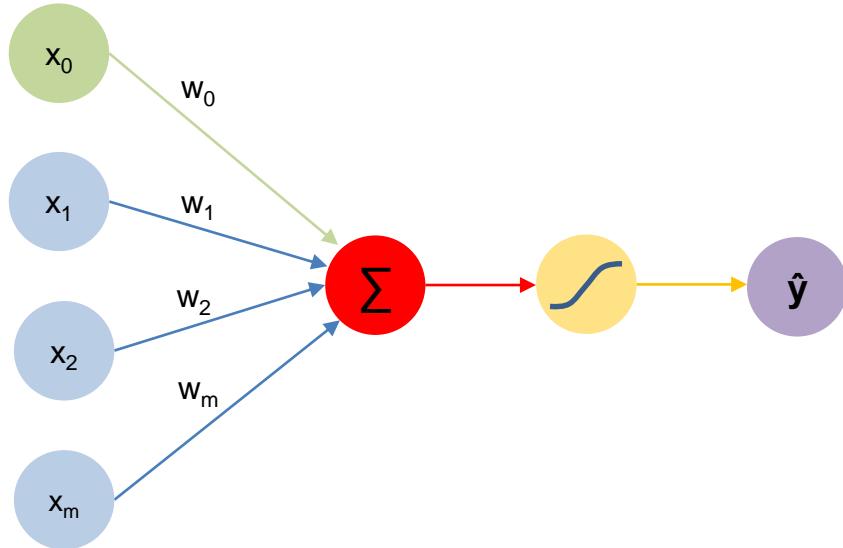
Non-linear activation function

Bias

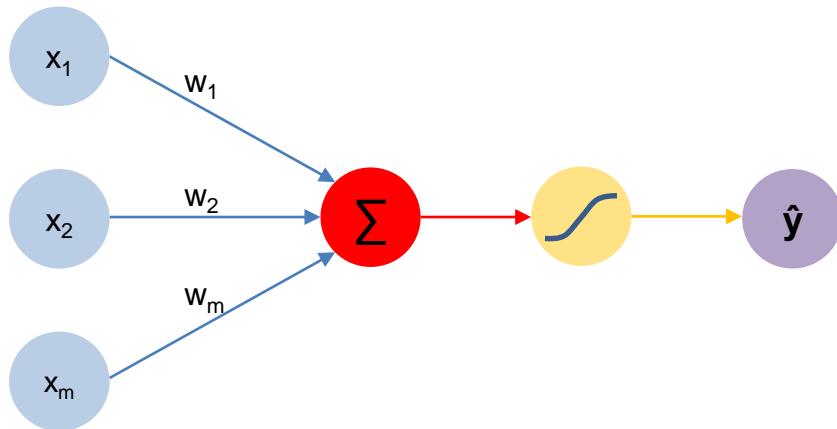
$\hat{y} = g (w_0 + X^T W)$

The diagram illustrates the three steps of a perceptron. It starts with a red arrow pointing from 'Take a dot product' to 'Add a Bias'. Another red arrow points from 'Add a Bias' to 'Take a non-linearity'. Below this, a purple arrow points down to the output equation. The equation itself is $\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$. A red arrow points from 'Linear combination of inputs' to the term $\sum_{i=1}^m x_i w_i$. A green arrow points from 'Bias' to the term w_0 . A yellow arrow points from 'Non-linear activation function' to the function g . To the right of the equation, the final output is given as $\hat{y} = g (w_0 + X^T W)$.

The perceptron

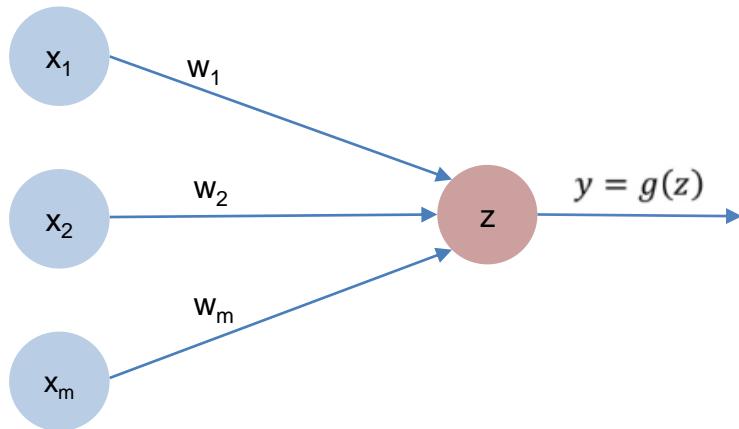


The perceptron



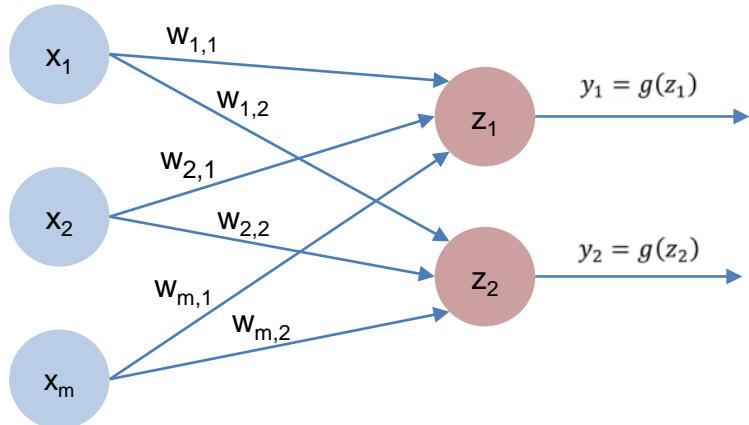
$$z = w_0 + \sum_{j=1}^m x_j w_j$$

The perceptron



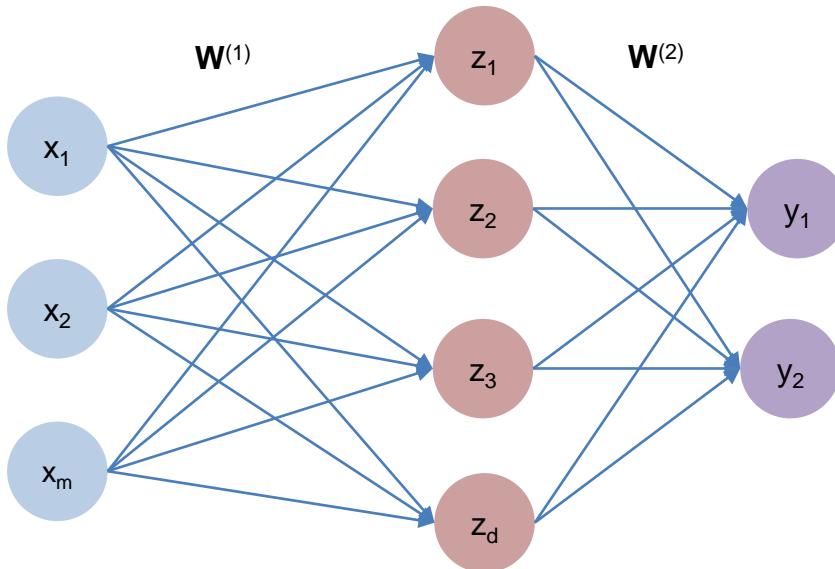
$$z = w_0 + \sum_{j=1}^m x_j w_j$$

Multi-output perceptron



$$z_i = w_{0,i} + \sum_{j=1}^m x_j w_{j,i}$$

Single layer Neural network



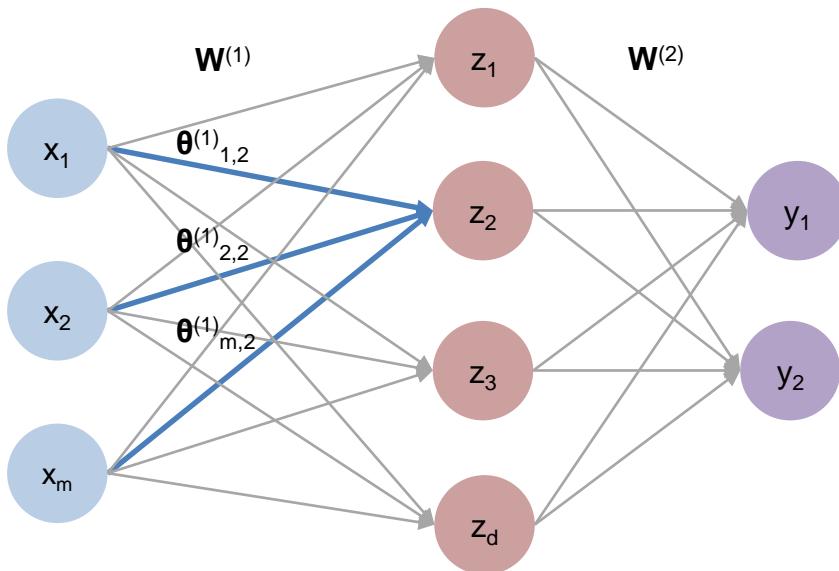
Inputs

Hidden
layer

Outputs

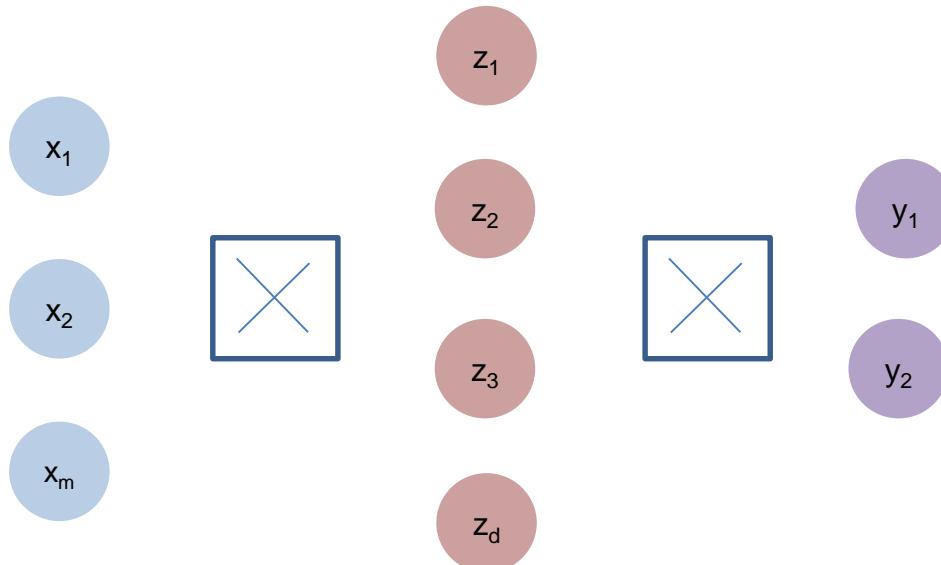
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

Multi-output perceptron

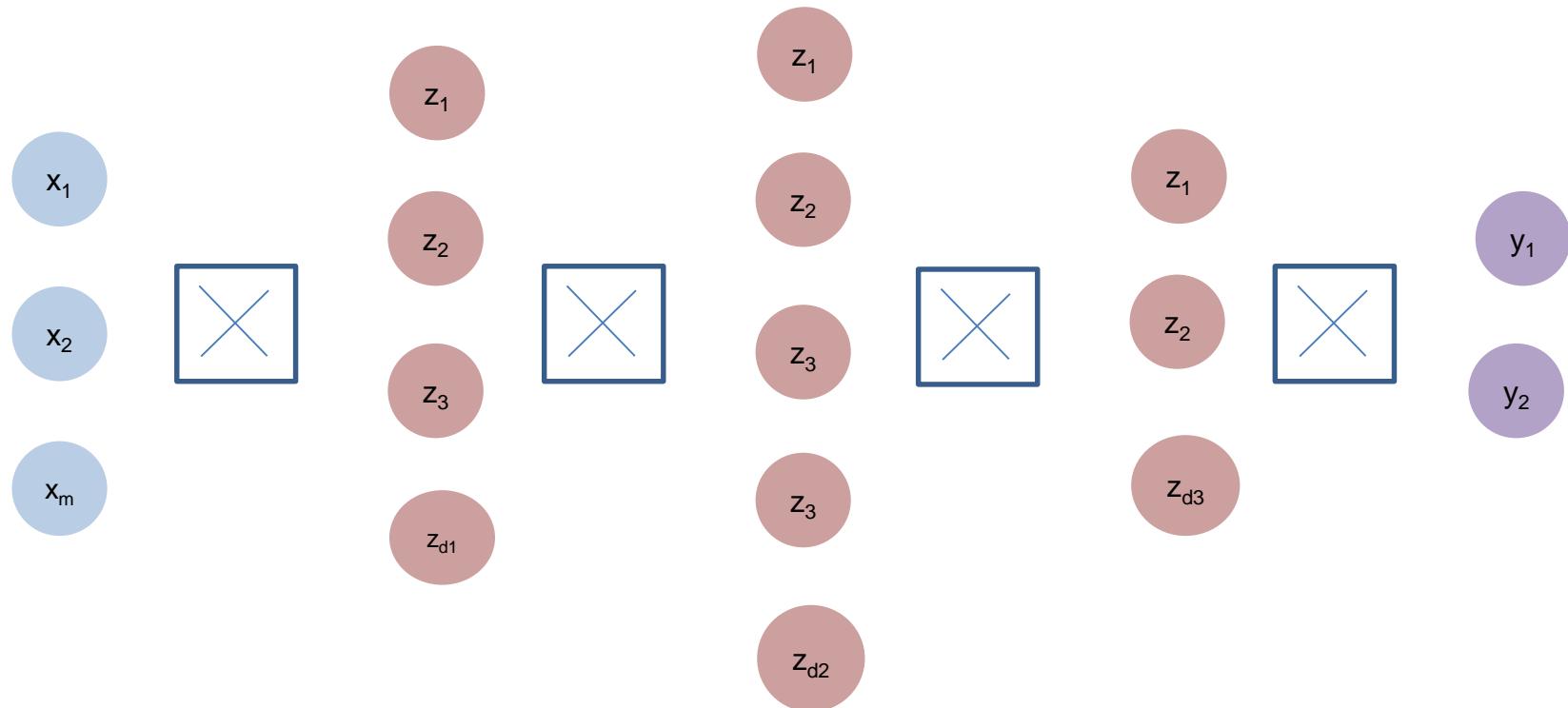


$$\begin{aligned} z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^m x_j w_{j,2}^{(1)} \\ &= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \end{aligned}$$

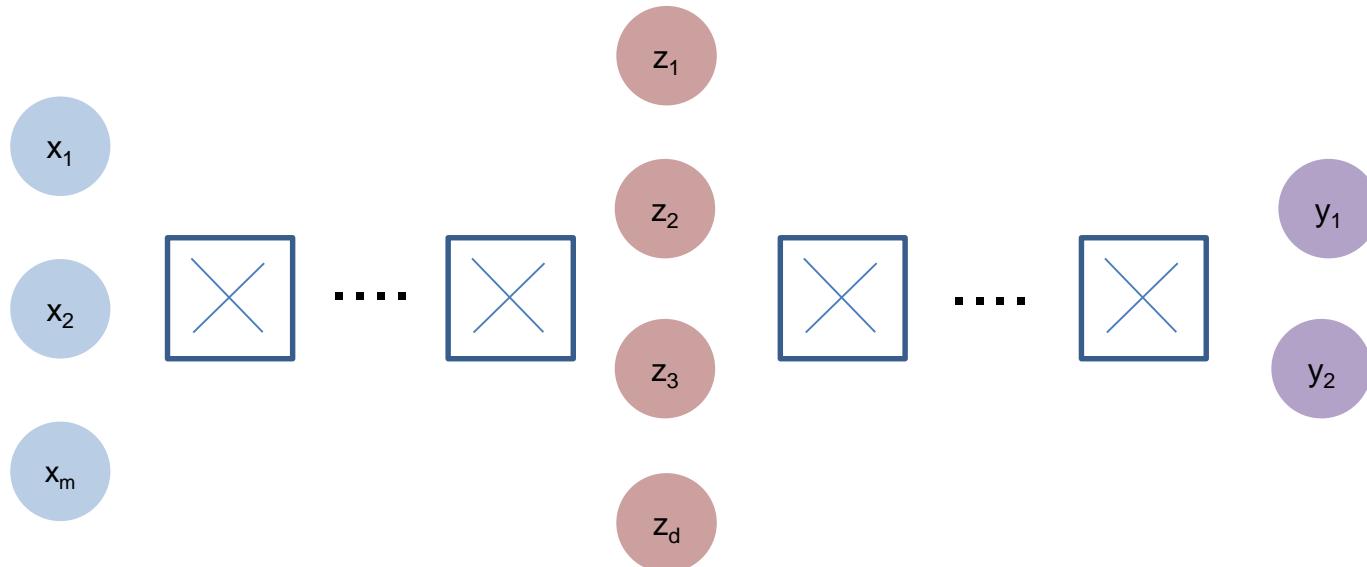
Single layer Neural Network



Deep Neural Network



Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$

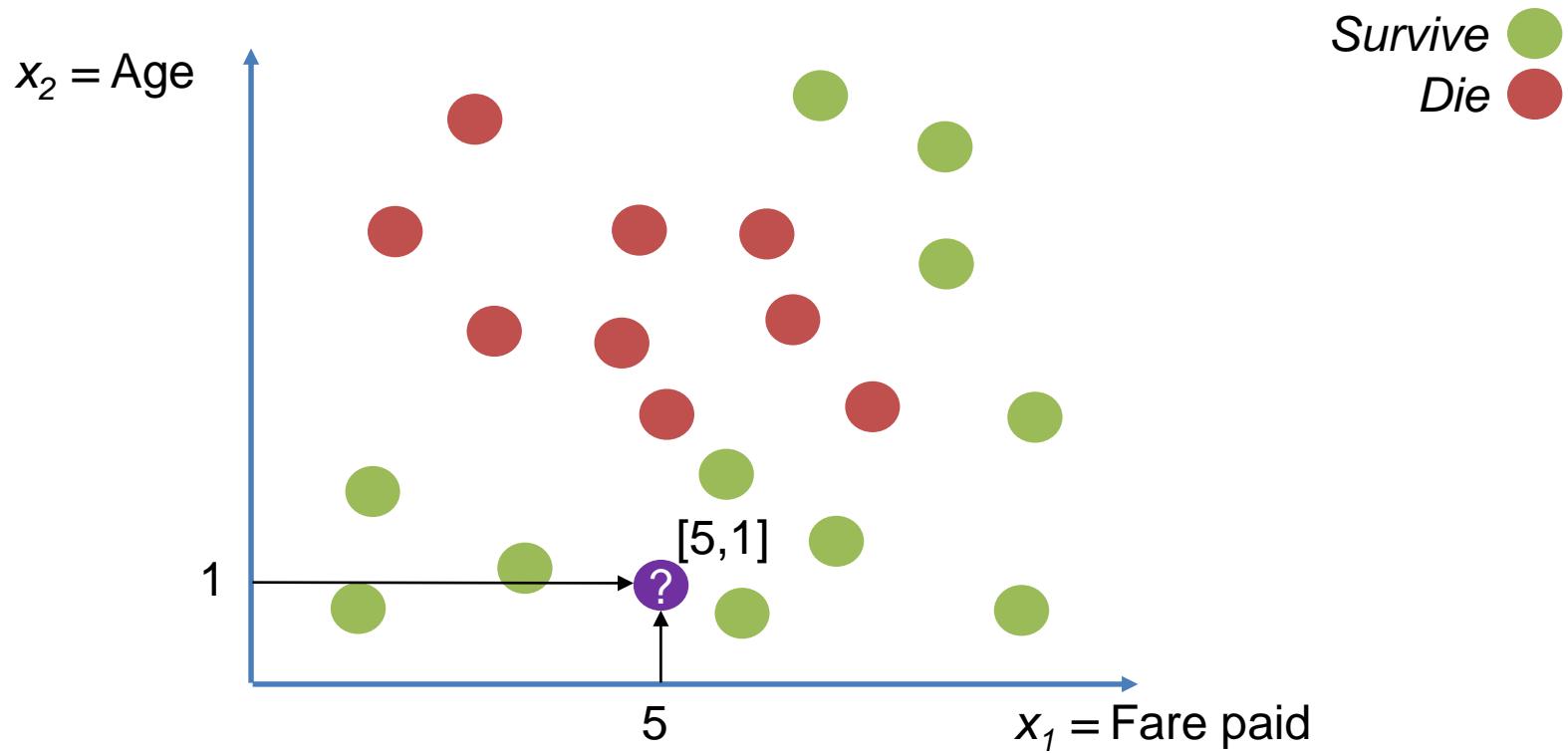
Lets apply a Neural Network

Lets play with the classic Titanic dataset: Predict who will survive!

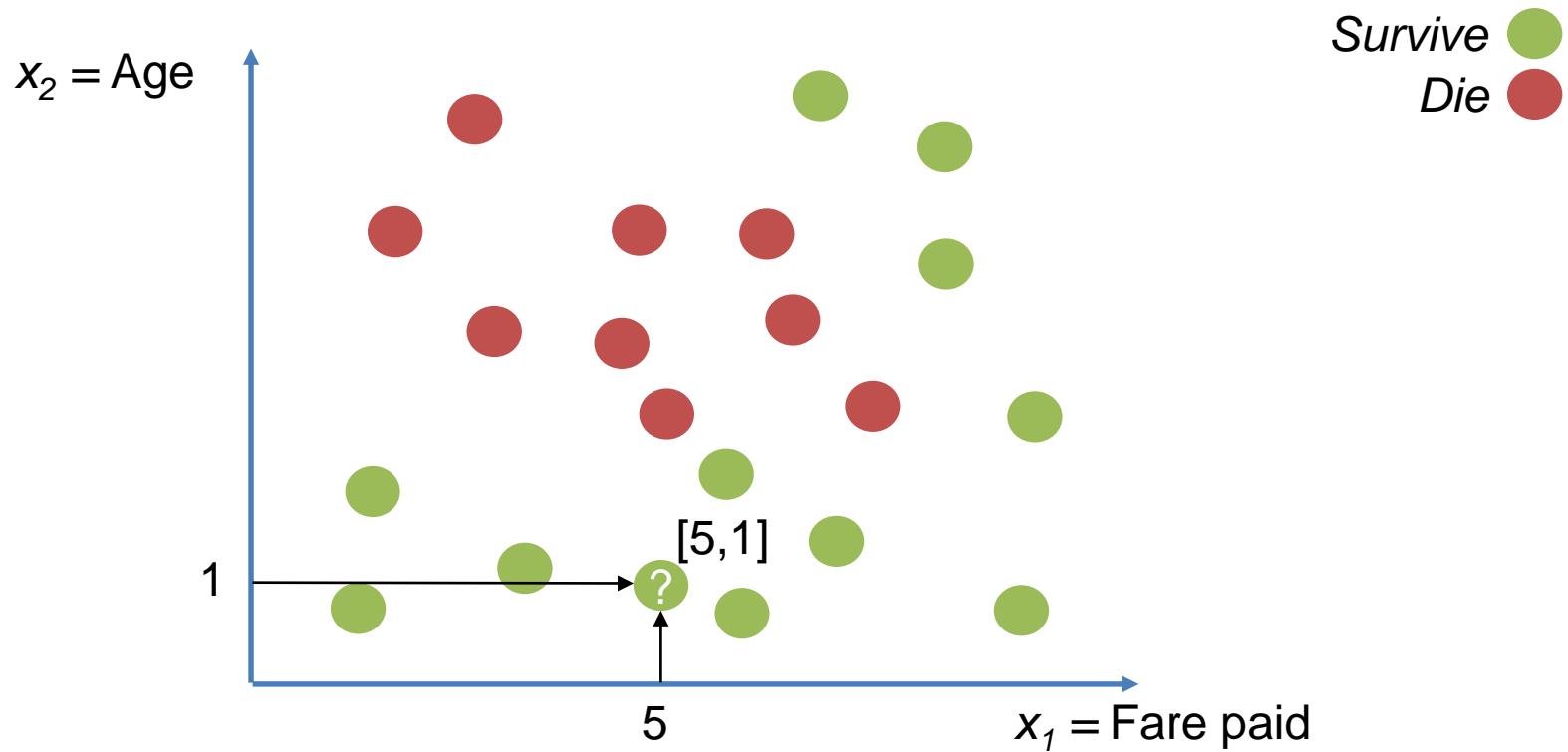
Lets only consider a two feature model:

$$\begin{aligned}x_1 &= \text{Fare paid} \\x_2 &= \text{Age}\end{aligned}$$

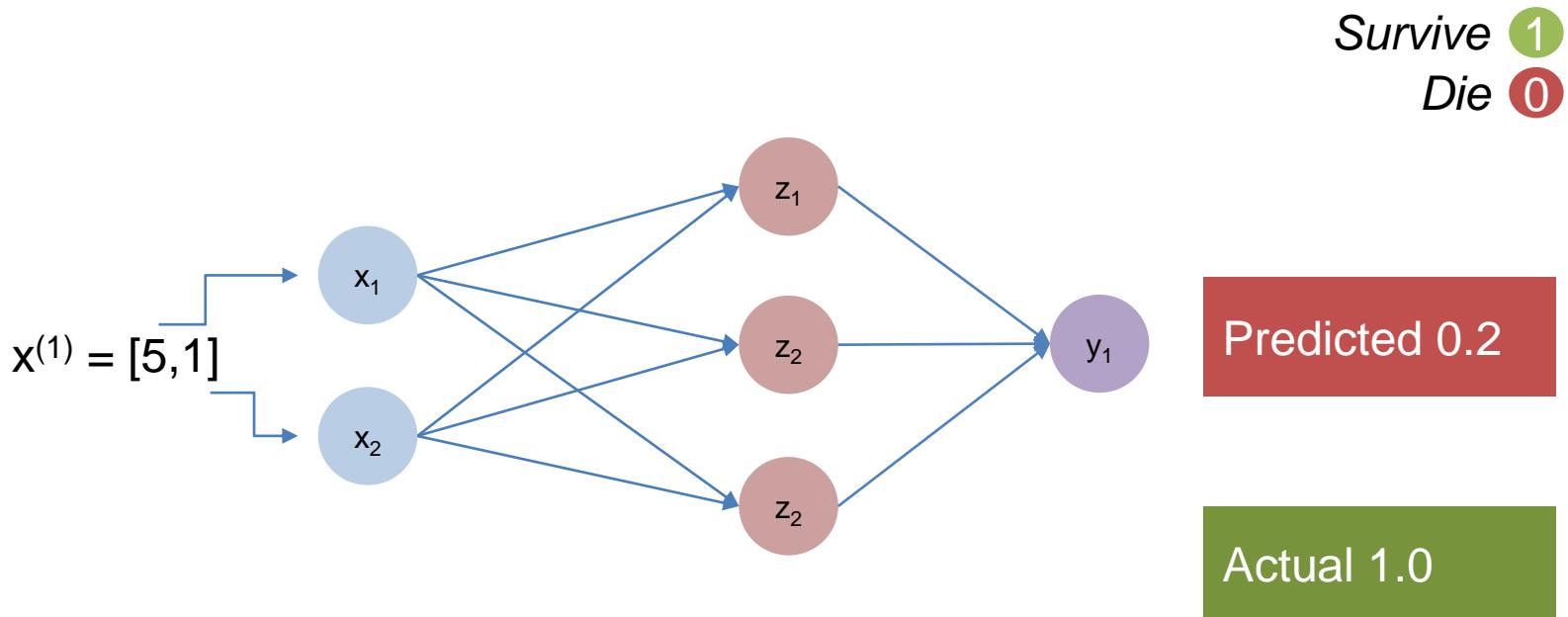
Lets apply a Neural Network



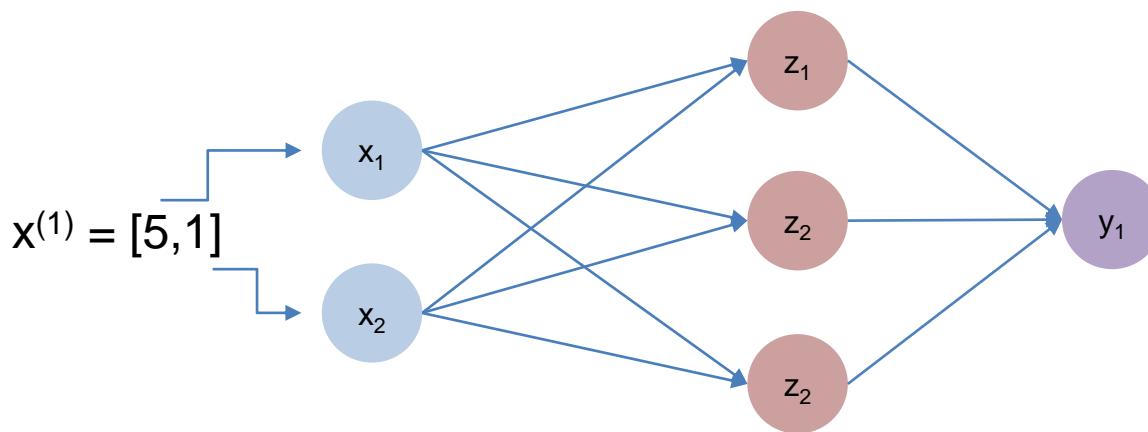
Lets apply a Neural Network



Example



Example



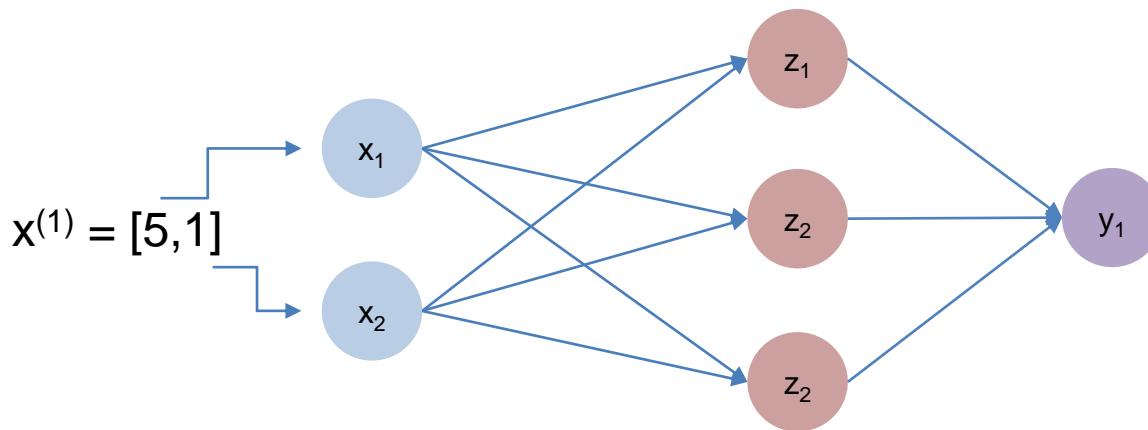
Predicted 0.2

Actual 1.0

The parameters are currently random!
We need to **train** our network.

Quantifying loss

We can use the errors from our predicted value relative to our actual value.
We use a loss function to define our loss.

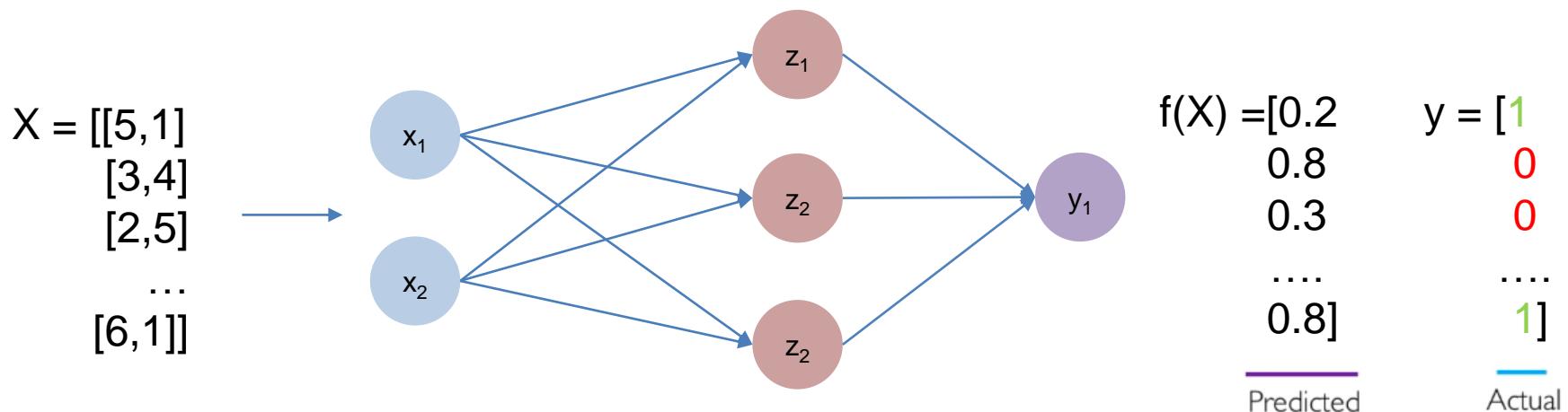


$$\mathcal{L} \left(\underbrace{f(x^{(i)}; \mathbf{w})}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}} \right)$$

Empirical Loss

Loss functions are also known as: Objective functions, cost functions, empirical risk

Empirical loss: *The mean loss across all samples*



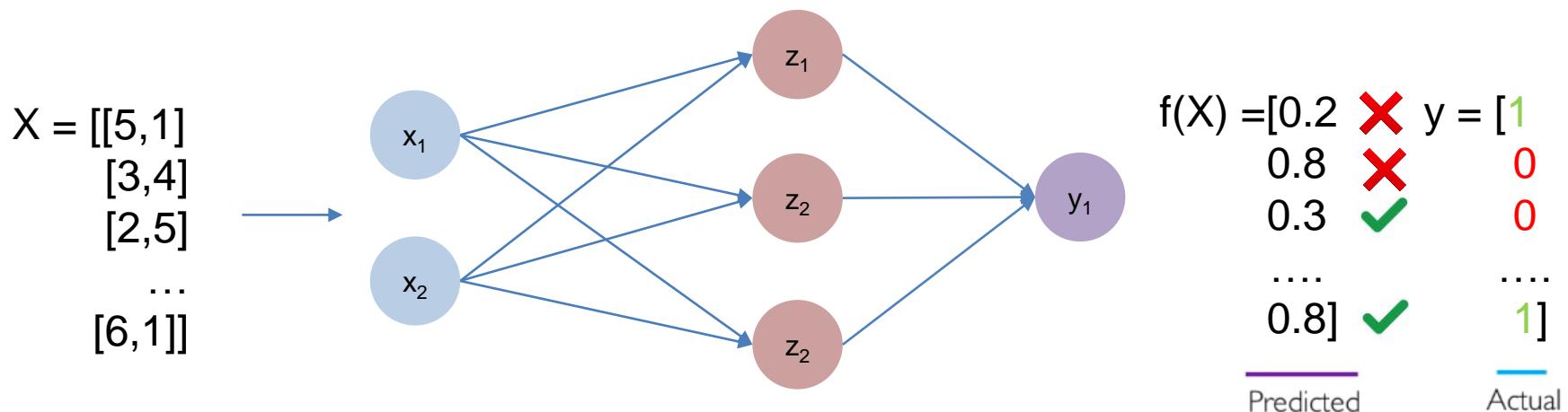
$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

Predicted Actual

Empirical Loss

Loss functions are also known as: Objective functions, cost functions, empirical risk

Empirical loss: *The mean loss across all samples*

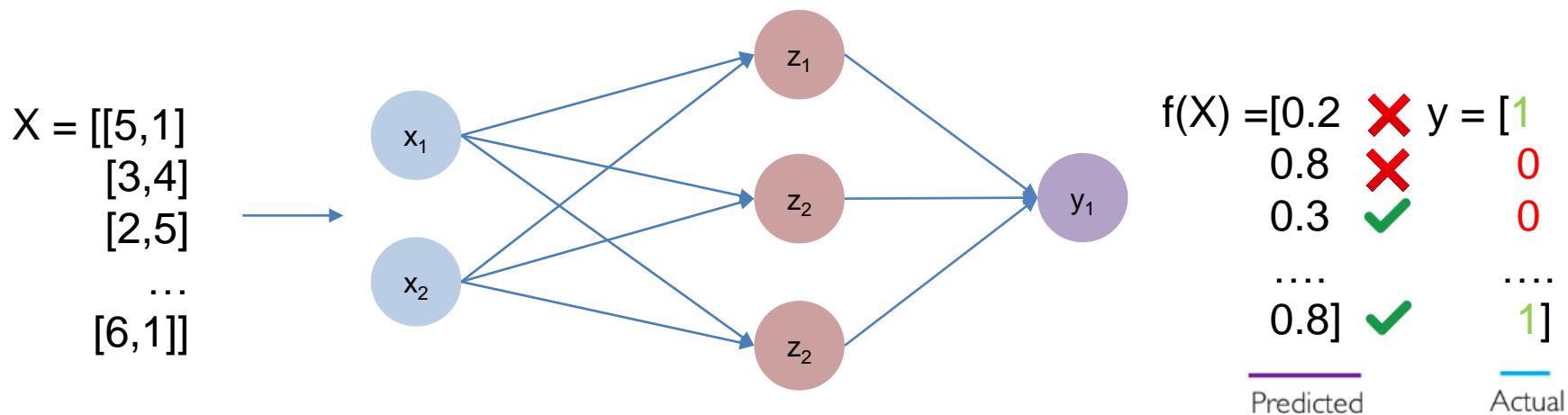


$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

Predicted Actual

Binary Cross Entropy Loss

Comparing models that output a probability between 0 and 1

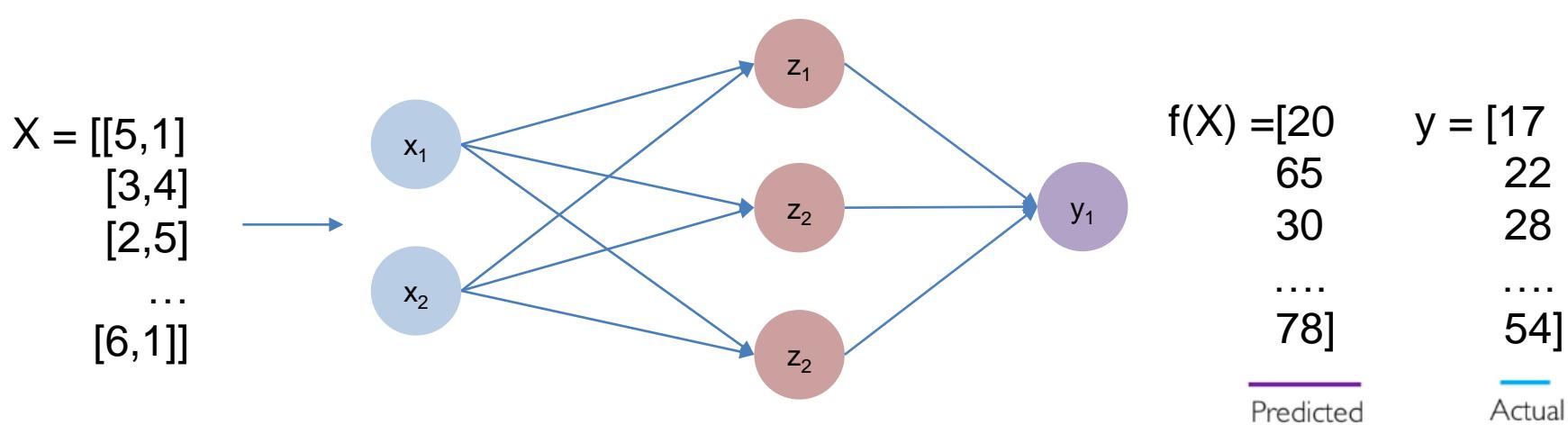


$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)} \log(f(x^{(i)}; \mathbf{W}))}_{\text{Actual}} + \underbrace{(1 - y^{(i)}) \log(1 - f(x^{(i)}; \mathbf{W}))}_{\text{Actual}}$$

Predicted Predicted

Mean Square Error loss

Instead of 0 or 1, we might have a regression model for continuous output values



$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \left(w^T x(i) - y(i) \right)^2$$

Training Neural Network

Use the loss to train the network.

Can we find the weights that achieve the lowest loss?

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$



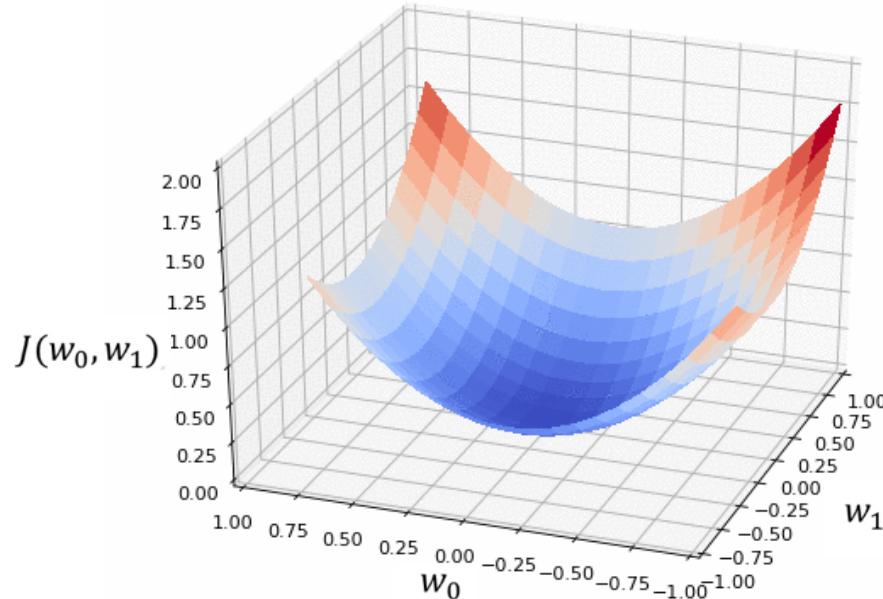
Remember:

$$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\}$$

Training Neural Network

Use the loss to train the network.

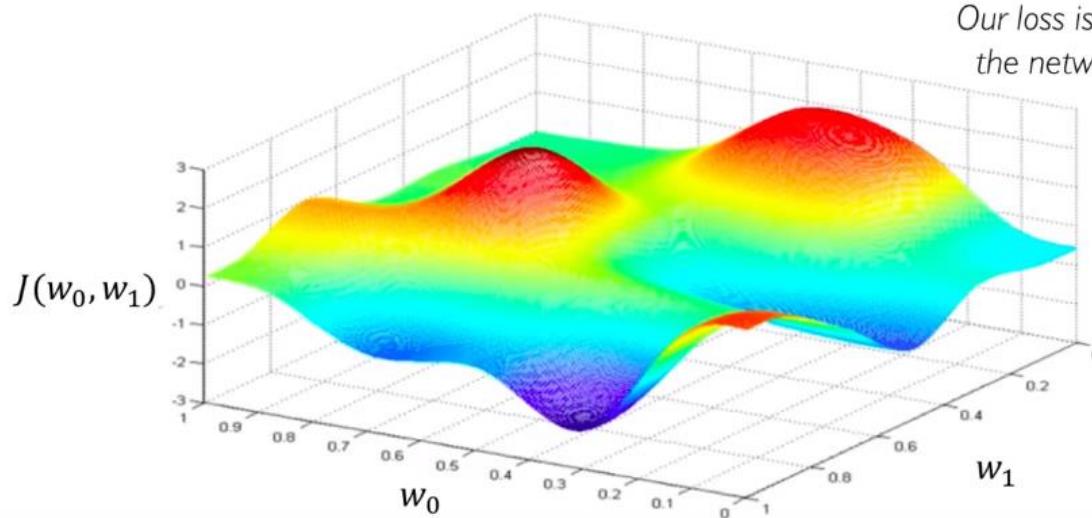
Can we find the weights that achieve the lowest loss?



Gradient Descent

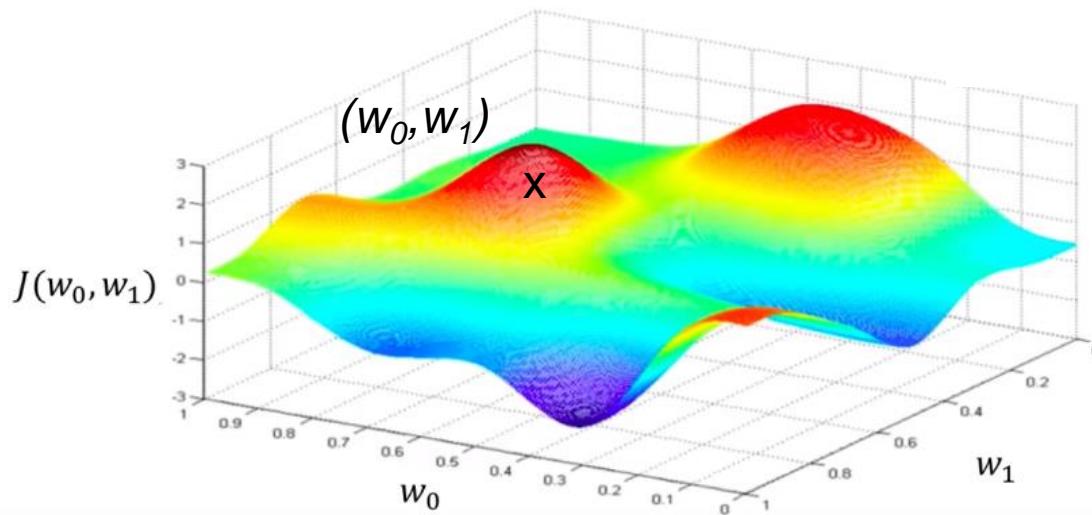
$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

Remember:
Our loss is a function of
the network weights!



Gradient Descent

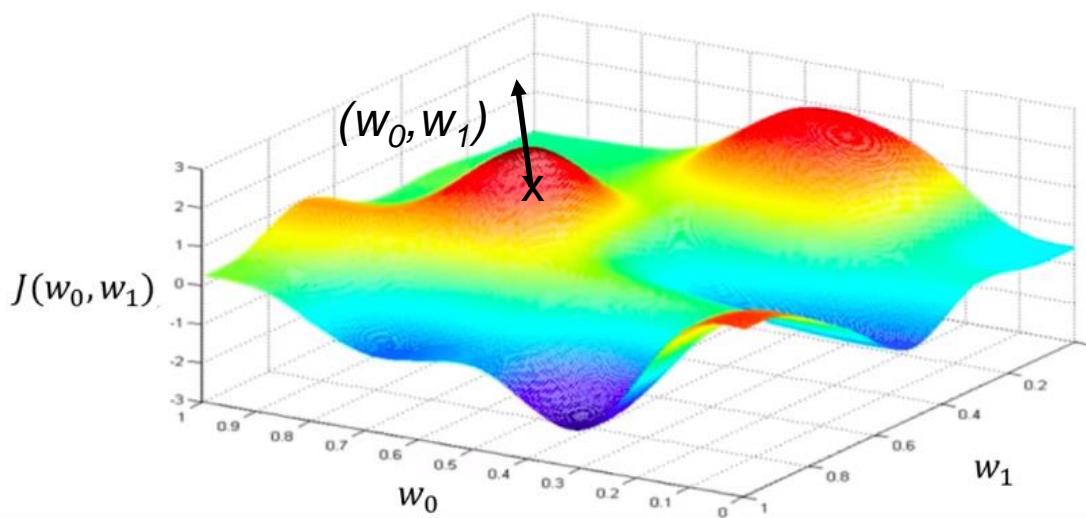
Compute the gradient at (w_0, w_1) $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$



Gradient Descent

Compute the gradient at (w_0, w_1)

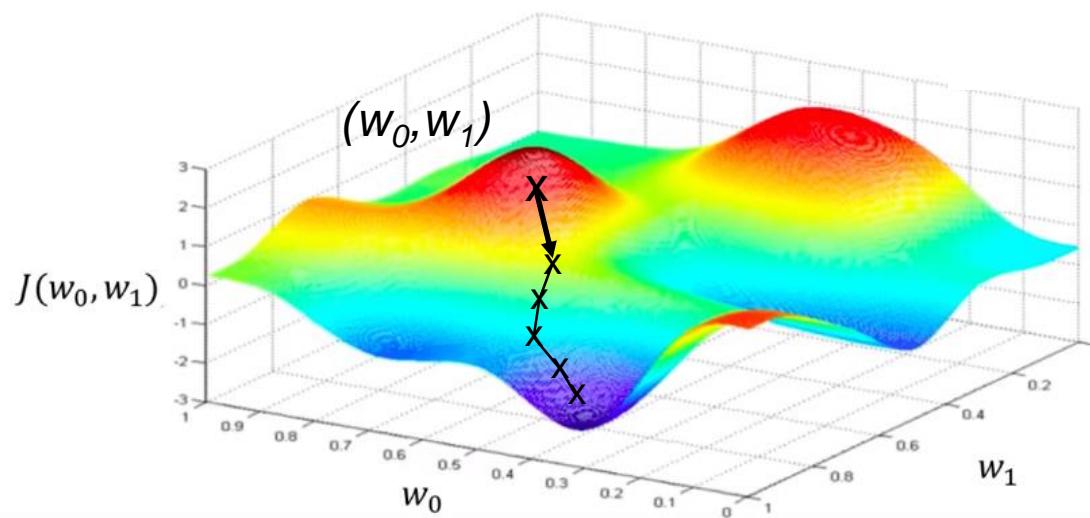
$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$



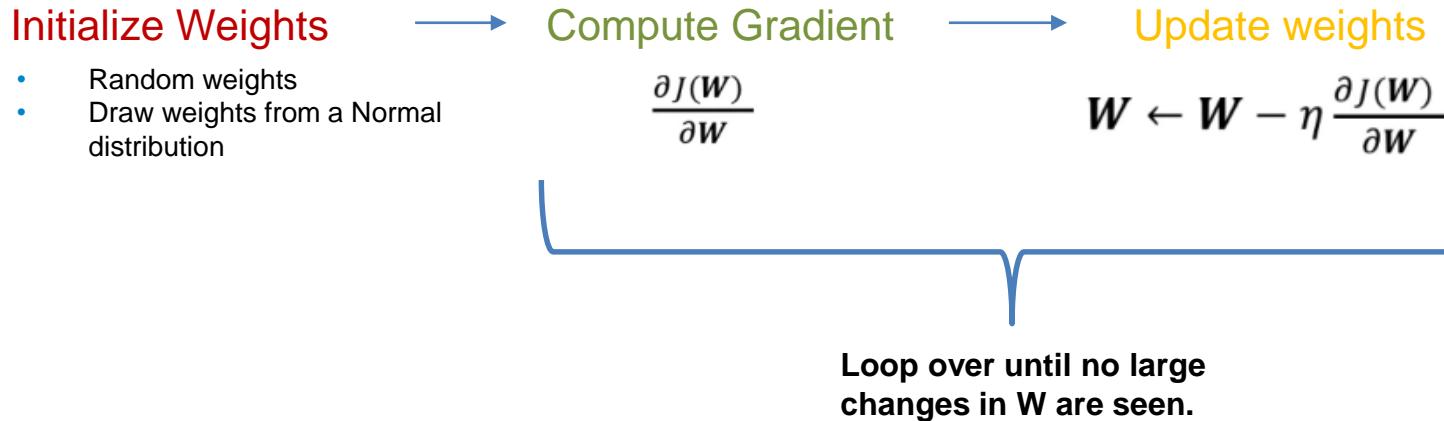
Notice this is the direction of maximum descent!

Gradient Descent

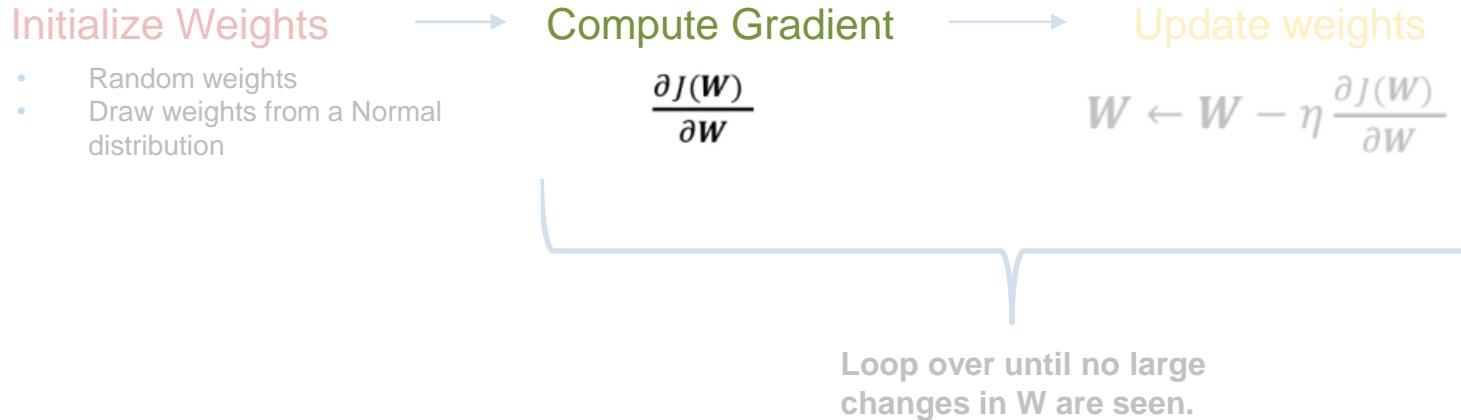
Take the opposite direction of maximum gradient



Summary of gradient descent



How do we compute the gradient?



How do we compute the gradient?

Backpropagation

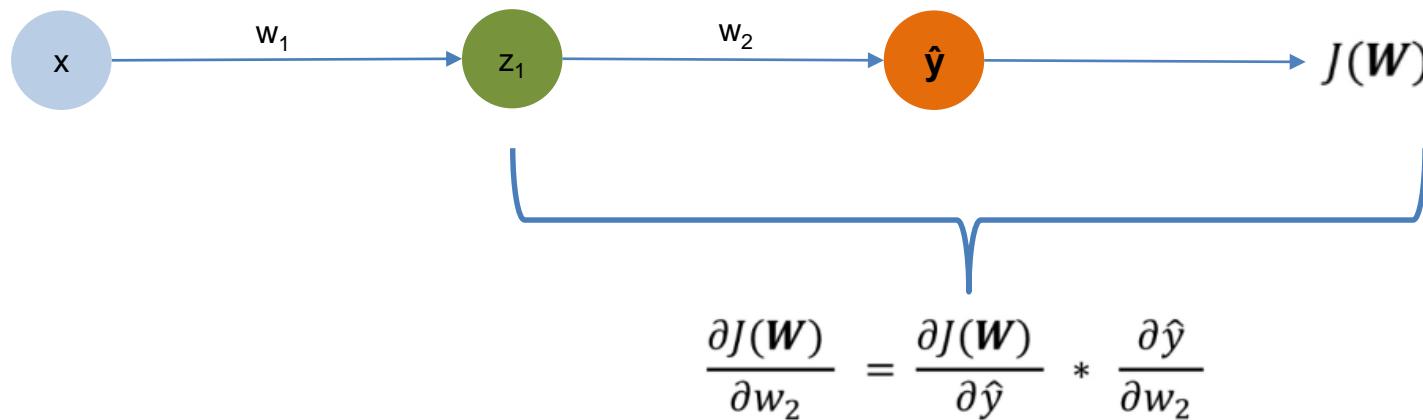


Compute how a small change in a weight, such as w_2 , affects the final loss.

Lets calculate the gradient of the loss given w_2 .

How do we compute the gradient?

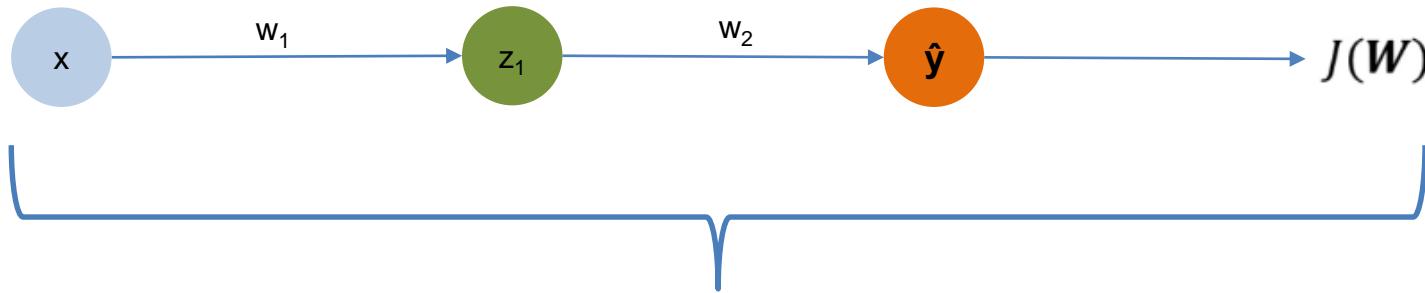
Backpropagation



Compute the chain rule!

How do we compute the gradient?

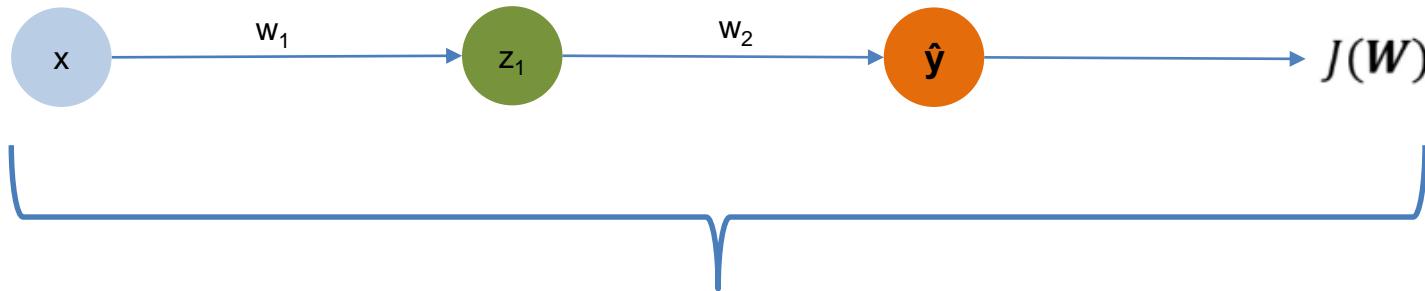
Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

How do we compute the gradient?

Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

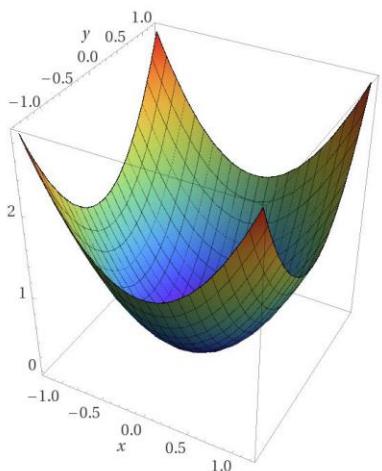
Back propagating the errors to the original input.

Repeat for every weight in the network!

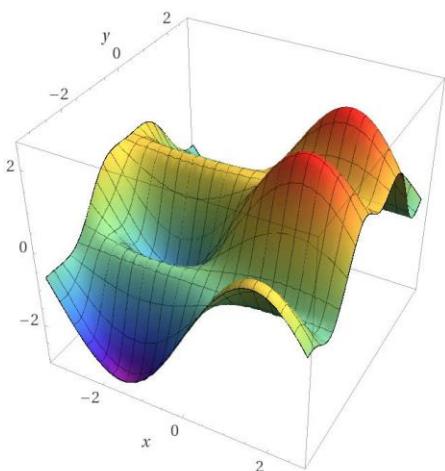
Loss Landscape

In practice training a real neural network is highly complex

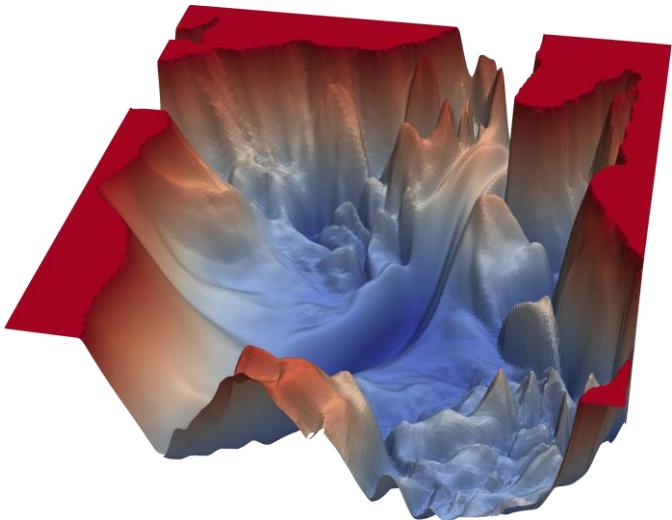
Many local minima. Finding true minimum is difficult.



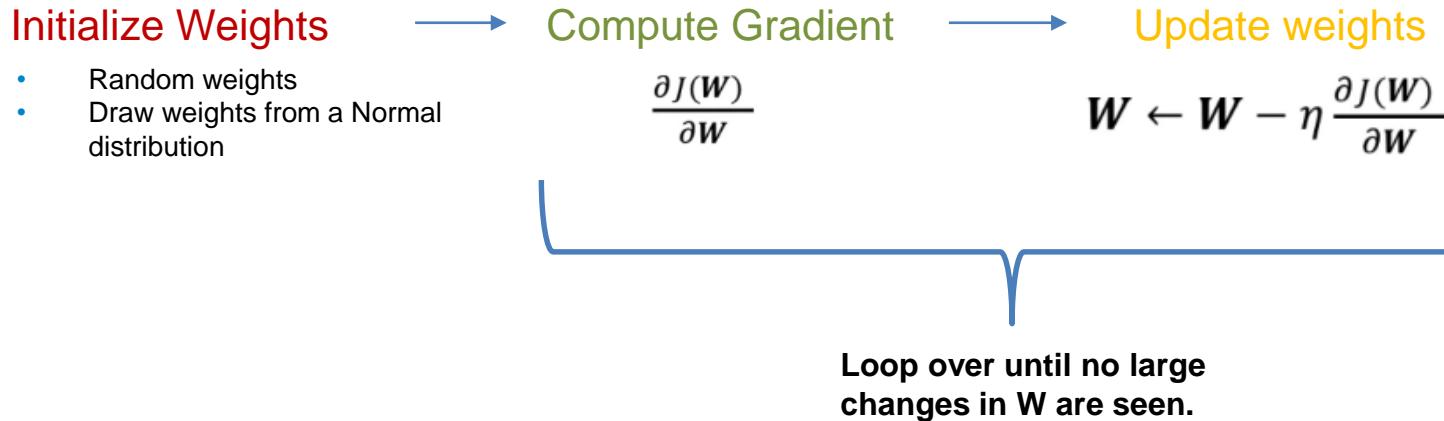
Computed by Wolfram|Alpha



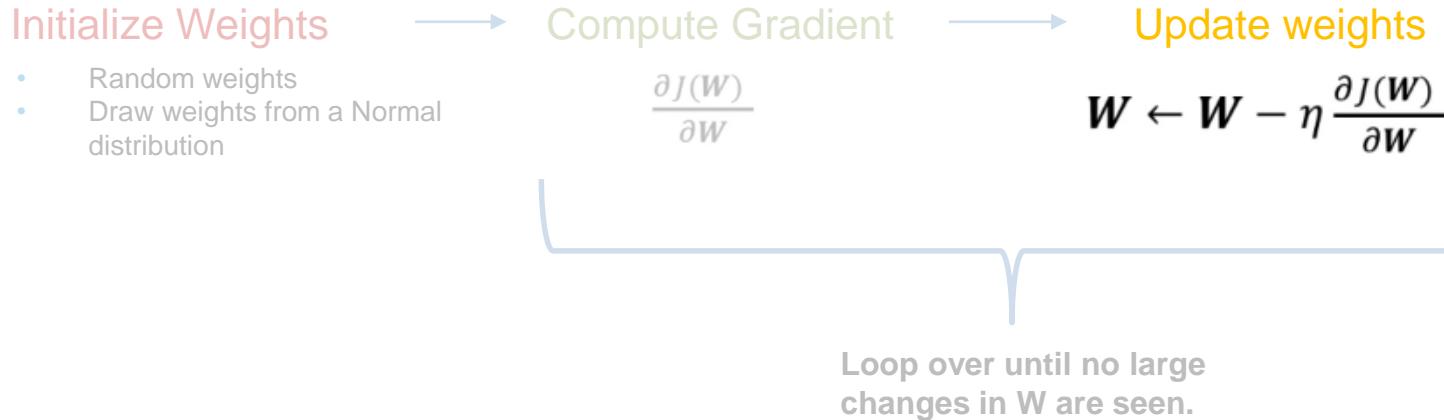
Computed by Wolfram|Alpha



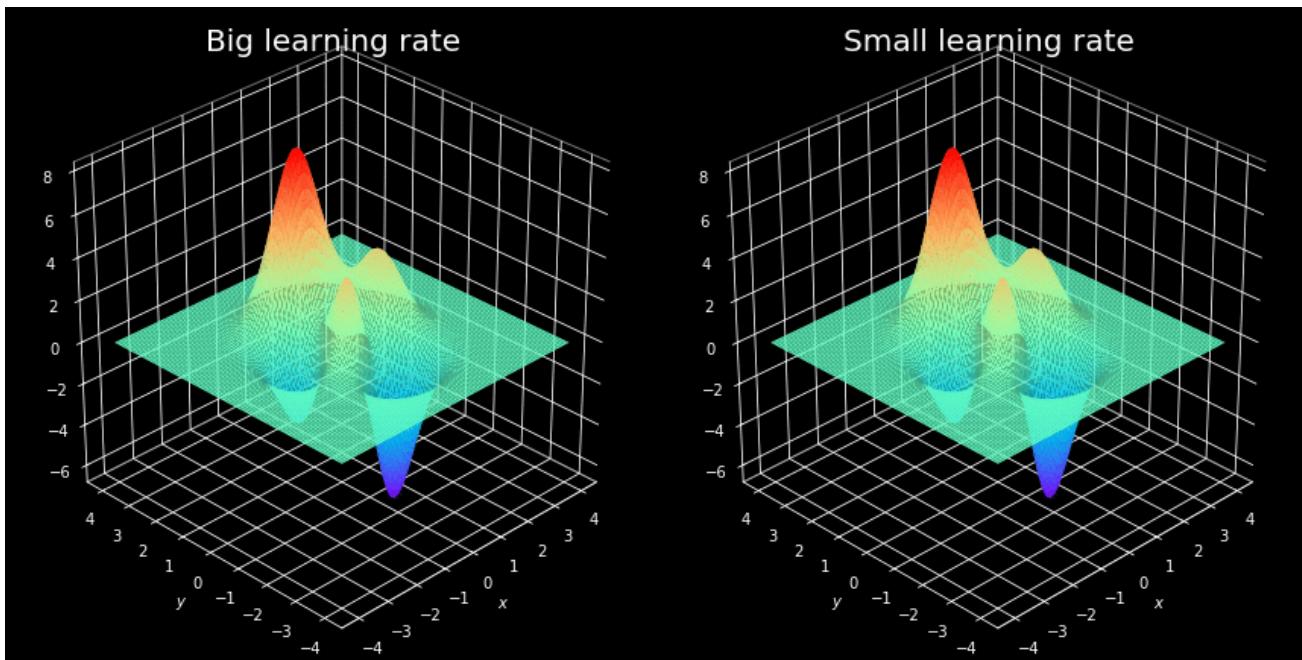
Summary of gradient descent



Summary of gradient descent



Learning rate



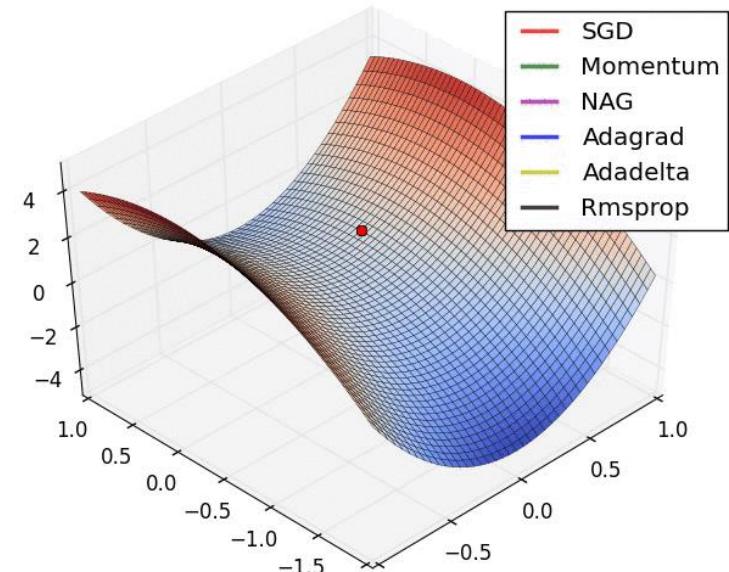
Learning rate

Optimising the learning rate?

Option 1: Fixed Learning Rates

Option 2: Adaptive Learning Rate algorithms

- Magnitude of gradient
- Size of weights
- Current learning rate
- Etc...



Tips for training: Computing gradients

Computing gradient across all data points is expensive to compute

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

Initialize Weights → Compute Gradient → Update weights

- Random weights
- Draw weights from a Normal distribution

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$



Loop over until no large changes in \mathbf{W} are seen.

Tips for training: Computing gradients

Computing gradient across all data points is expensive to compute

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

Initialize Weights

- Random weights
- Draw weights from a Normal distribution

Pick a single data point i

Compute Gradient

$$\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}}$$

Update weights

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

Stochastic gradient descent!

Easy to compute but noisy

Loop over until no large changes in W are seen.

Tips for training: Computing gradients

Computing gradient across all data points is expensive to compute

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

Initialize Weights

- Random weights
- Draw weights from a Normal distribution

Pick a batch B data points

Compute Gradient

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(\mathbf{W})}{\partial \mathbf{W}}$$

Update weights

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

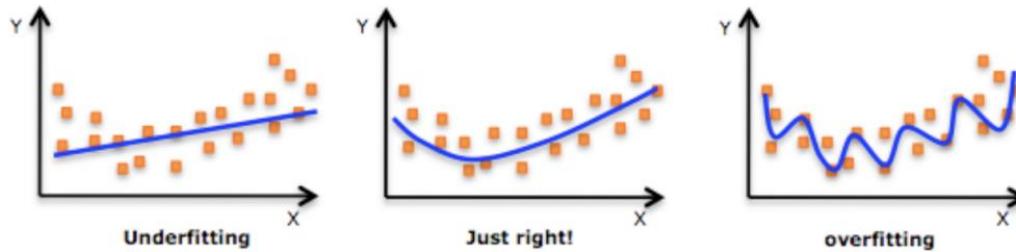
Batch gradient descent

Fast to compute and much better estimate than stochastic gradient descent

Loop over until no large changes in \mathbf{W} are seen.

Tips for training: Overfitting

Regression



Classification



Tips for training: Overfitting

Similar to other algorithms (SVMs, Ridge Regression, etc.), we can implement regularization

What is it?

It constrains our optimization problem to discourage complex models

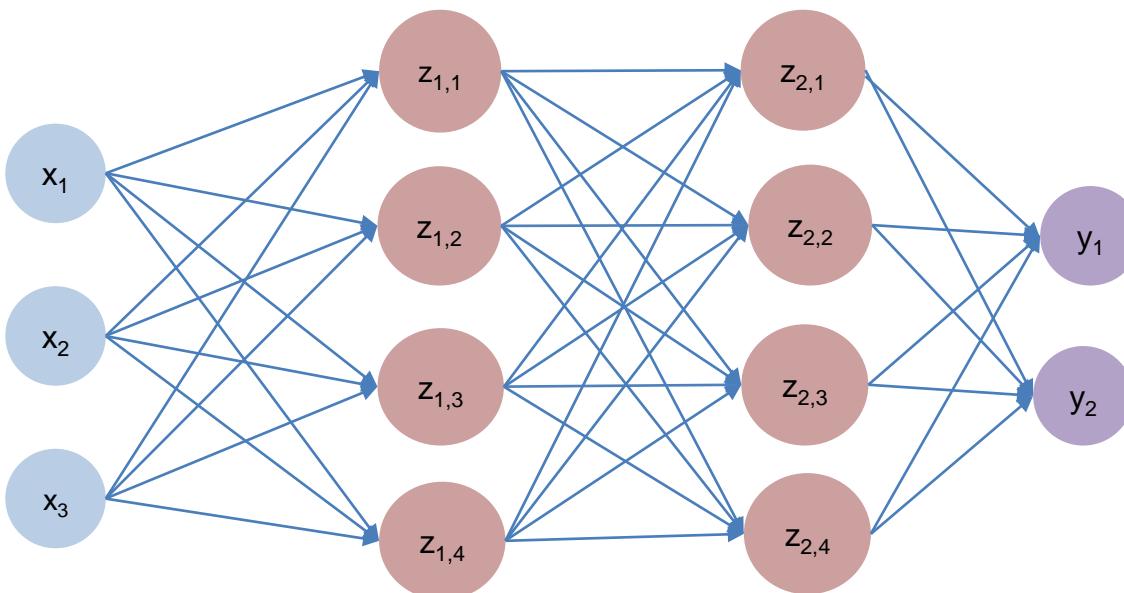
Why do we use it?

We need to make sure that we are producing a model that is as close to the generating function of the data.

We want our model to generalize to unseen data!

Dropout

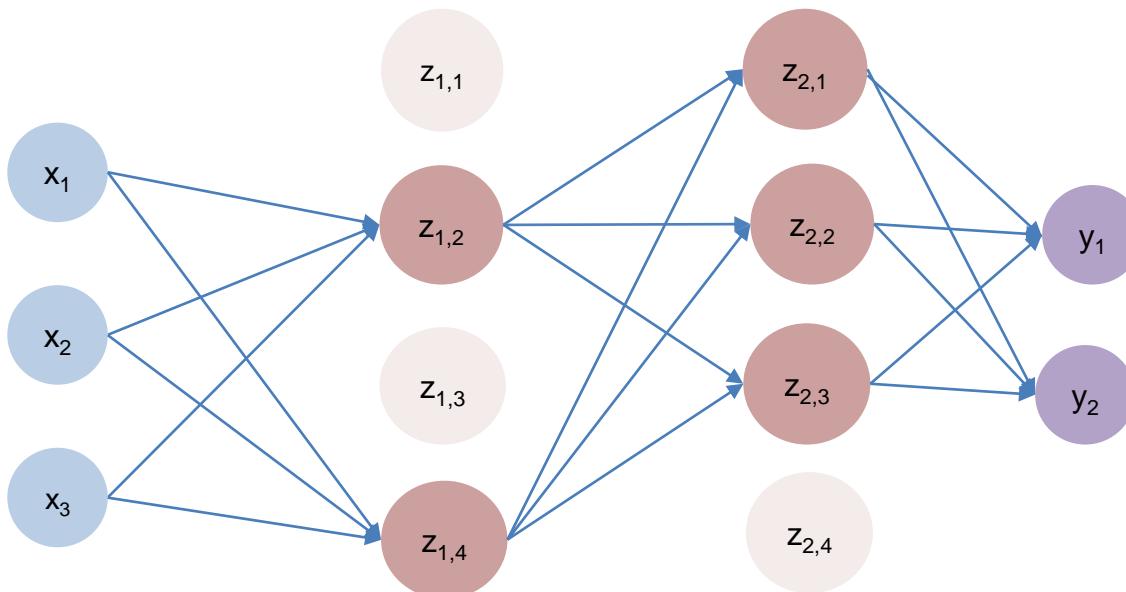
Dropout randomly sets some activation neurons to 0



Dropout

Dropout randomly sets some activation neurons to 0

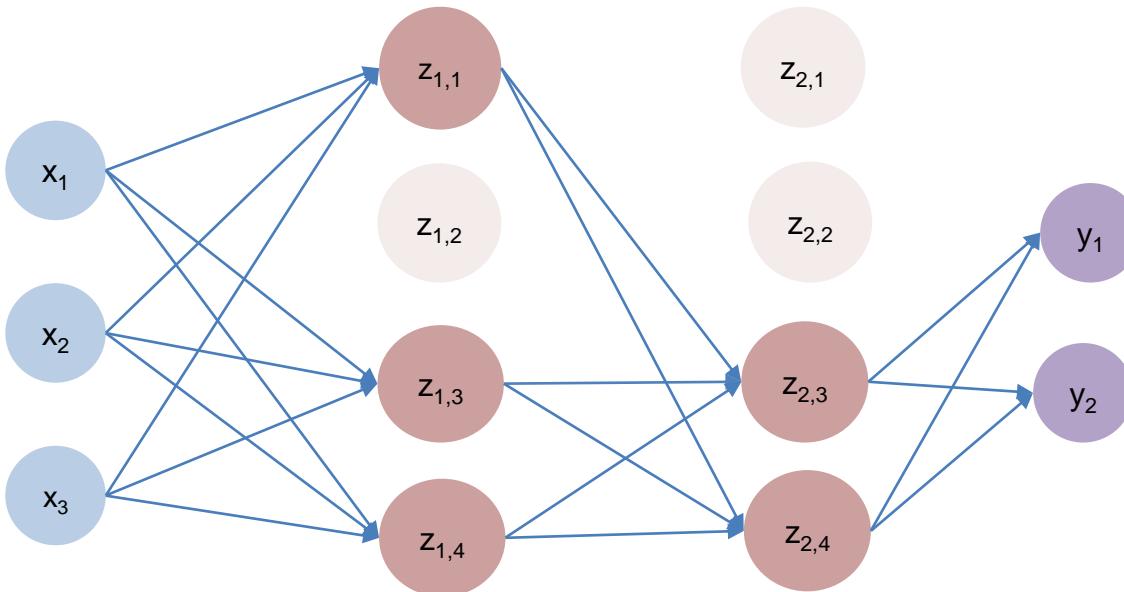
Typically 50% of neurons in each layer
Prevents reliance on single nodes



Dropout

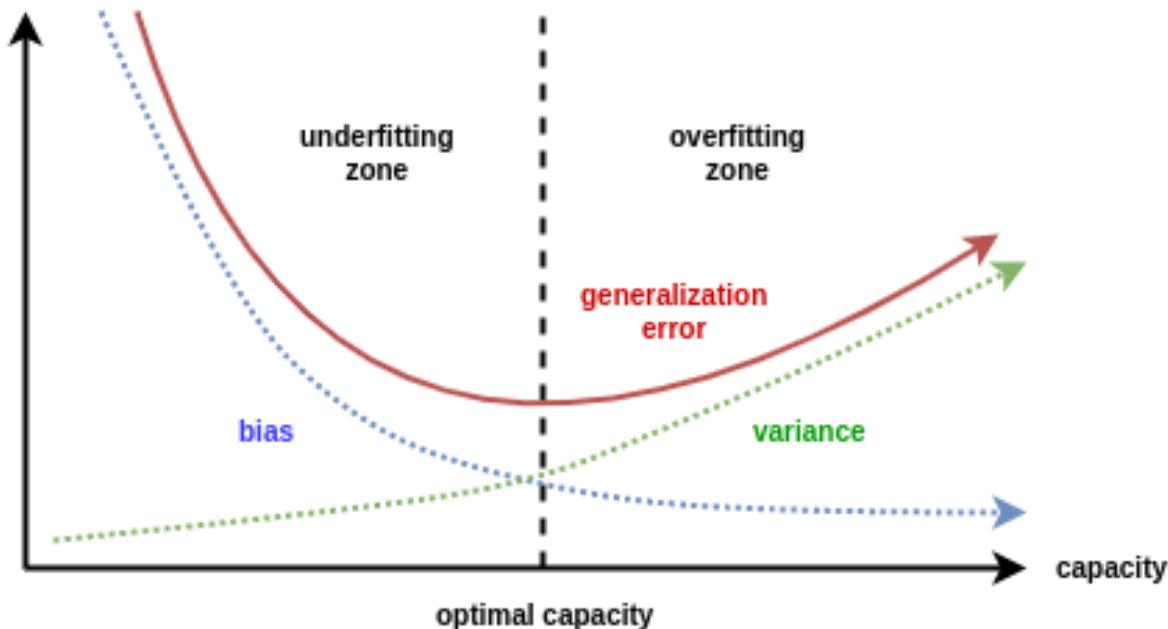
Dropout randomly sets some activation neurons to 0

Typically 50% of neurons in each layer
Prevents reliance on single nodes



Early Stopping

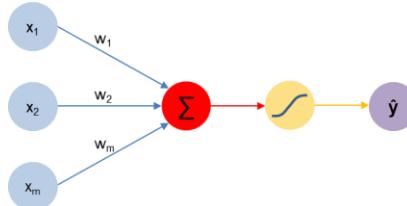
If the model trains for long enough the a very complex and unbiased model can be learned but the variance or error increases as seen in the overfitting zone.



Quick Review

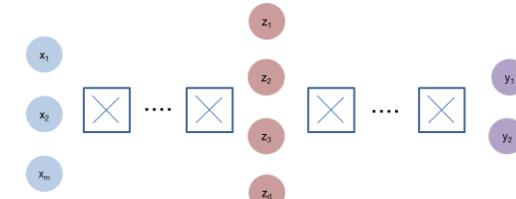
Perceptron

- A linear sum
- Non-linear activation function



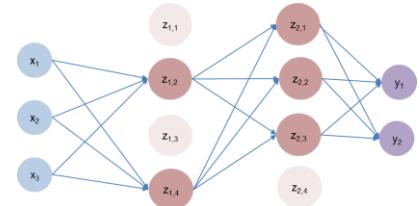
Neural Network

- Stacking of perceptrons
- Optimisation through back propagation



Training

- Regularisation
- Optimization
- Learning rate



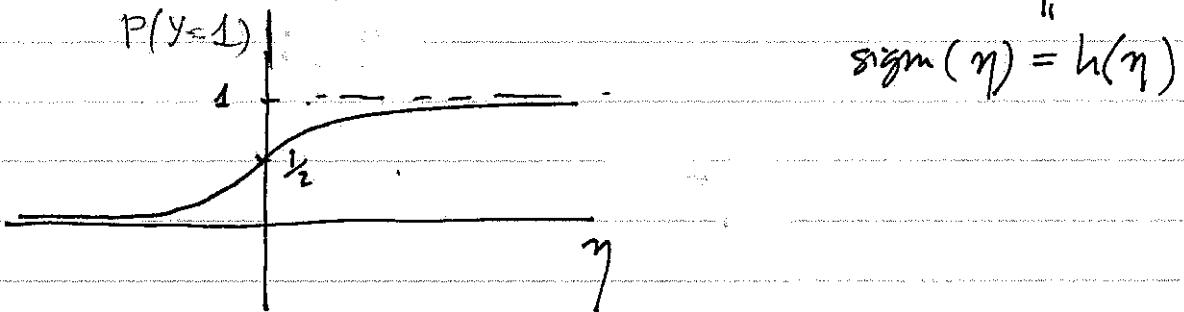
Logistic regression as a "neural network"

Reminder: $(x_1^{(i)}, \dots, x_p^{(i)})$ $y^{(i)} \in \{0, 1\}$ $i=1, \dots, N$

Assume observable is Bernoulli: $P(Y=y | \vec{x}) = P(Y=1)(1-P(Y=0))$
 $= \text{Ber}(\vec{x}) / P(Y=1)$

If we express probability in terms of log-odds:

$$\eta = \log \frac{P(Y=1)}{P(Y=0)} = \log \frac{P(Y=1)}{1-P(Y=1)} \Rightarrow P(Y=1) = \frac{1}{1+e^{-\eta}}$$



Model: log odds is linear function of descriptors:

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(N)} & \dots & x_p^{(N)} \end{bmatrix}, \quad \eta = \vec{x}^T \cdot \vec{\beta} \quad \vec{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{pmatrix} \quad \vec{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$$

Infer $\vec{\beta}$ from data by maximizing likelihood

$$P(\vec{y} | X, \vec{\beta}) = \prod_{i=1}^N \text{Ber}(y^{(i)} | h(\vec{x}^{(i)} \cdot \vec{\beta}))$$

$\text{Ber}(\cdot) \equiv$
 Bernoulli

$$L = \sum_{i=1}^N \left[y^{(i)} \log[h(\vec{x}^{(i)} \cdot \vec{\beta})] + (1-y^{(i)}) \log[1-h(\vec{x}^{(i)} \cdot \vec{\beta})] \right]$$

optimisation: $\nabla_{\vec{\beta}} L |_{\vec{\beta}_{\log}^*} = 0$ Normal equations $X^T [\vec{y} - h(X \vec{\beta}_{\log}^*)] = \vec{0}$

$$h(X \vec{\beta}_{\log}^*) = h(\vec{x}^{(i)} \cdot \vec{\beta}_{\log}^*)$$

and the problem is convex with Hessian:

This is a concave function that can be maximized globally

$$H = \nabla_{\tilde{\beta}} [\nabla_{\tilde{\beta}}^T L] = -X^T [\text{diag}(\tilde{h}) \cdot [I - \text{diag}(\tilde{h})]] X$$

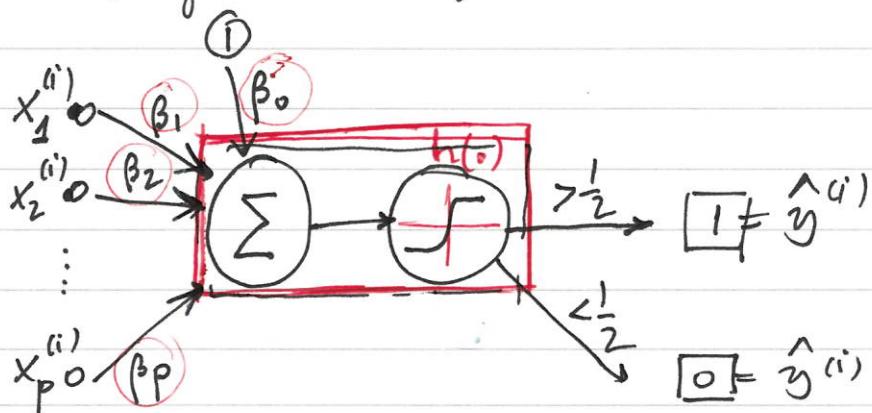
optimize using Newton or gradient methods, etc.
to obtain $\tilde{\beta}_{\log}^*$

classifier:

Given \bar{x}^{in} , compute

$$P(y | \bar{x}^{in}) = \frac{1}{1 + e^{-\bar{x}^{in} \cdot \tilde{\beta}_{\log}}} \quad \begin{cases} > \frac{1}{2} & \hat{y} = 1 \\ < \frac{1}{2} & \hat{y} = 0 \end{cases}$$

Diagrammatically :



where $\tilde{\beta}$ have to be optimized:

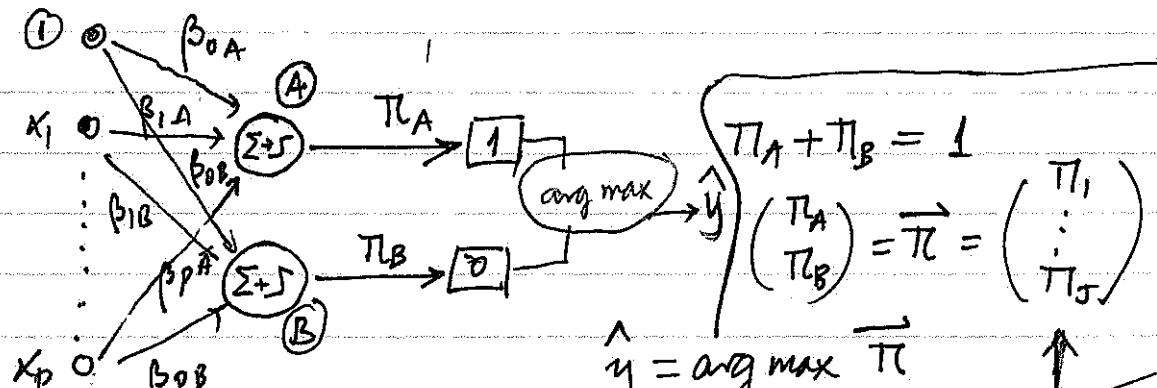
$$\max_{\tilde{\beta}} L = \sum_{i=1}^N \left[y^{(i)} \log [h(\bar{x}^{(i)} \cdot \tilde{\beta})] + (1-y^{(i)}) \log [1-h(\bar{x}^{(i)} \cdot \tilde{\beta})] \right]$$

log likelihood is equal
to minus cross-entropy between

$$\{y^{(i)}\} \text{ and } \{h(\bar{x}^{(i)} \cdot \tilde{\beta})\}$$

| Max of L is minimizing cross-entropy !

More generally:



$$\hat{y} = \arg \max_{j} \pi_j$$

classes

In general
for J classes

J=2 in this
case

Infer $\tilde{\beta} = \begin{bmatrix} \vec{\beta}_A & \vec{\beta}_B \end{bmatrix}$

$p \times J$

J is number of classes.

In this case $\vec{\beta}_A$ and $\vec{\beta}_B$ are related:

Original formulation

$$P(Y=1) = \frac{e^{x^T \vec{\beta}_A}}{e^{x^T \vec{\beta}_A} + 1}$$

$$1 - P(Y=1) = \frac{1}{e^{x^T \vec{\beta}_A} + 1}$$

$$\vec{\beta}'_A = \frac{\vec{\beta}_A}{2} = -\vec{\beta}'_B$$

Rewritten as

$$P(Y=1) = \frac{e^{x^T \vec{\beta}'_A}}{e^{x^T \vec{\beta}'_A} + e^{x^T \vec{\beta}'_B}}$$

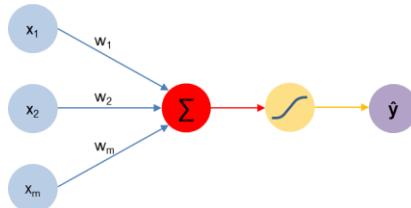
$$P(Y=0) = \frac{e^{x^T \vec{\beta}'_B}}{e^{x^T \vec{\beta}'_A} + e^{x^T \vec{\beta}'_B}}$$

Rewritten in
this form

Quick Review

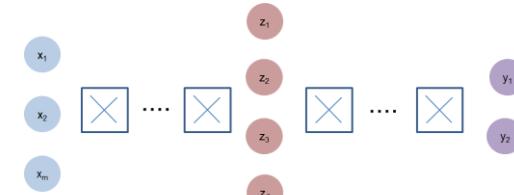
Perceptron

- A linear sum
- Non-linear activation function



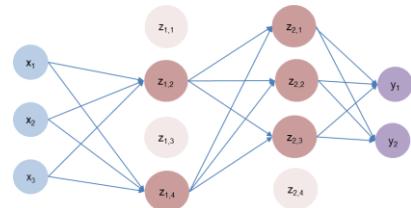
Neural Network

- Stacking of perceptrons
- Optimisation through back propagation



Training

- Regularisation
- Optimization
- Learning rate



Convolutional Neural Networks

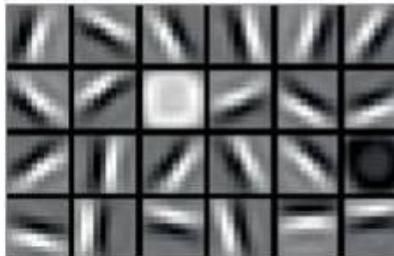
Until now we looked at fully connected multi-layer perceptrons.

However these aren't particularly good with images on their own!

Millions of images

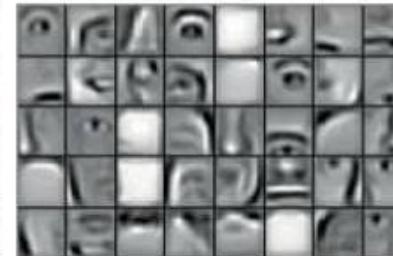


Low level features



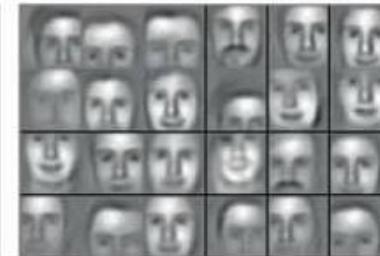
Lines & Edges

Mid level features



Eyes & Nose & Ears

High level features



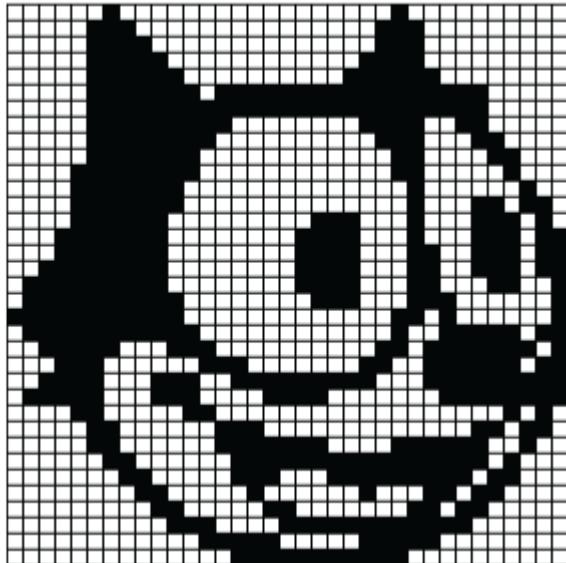
Facial Structure

Convolutional Neural Networks

How can we help a computer ‘see’?

Black: 0

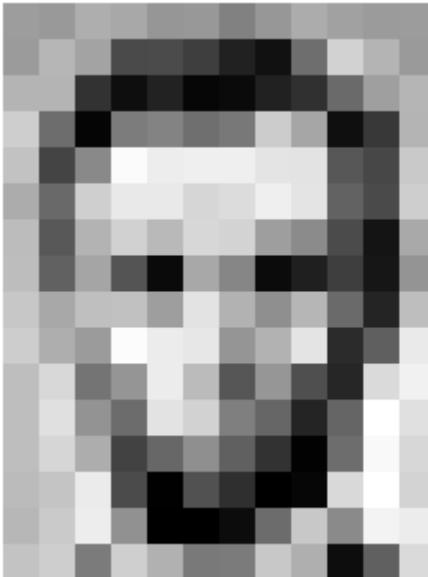
White: 1



Convolutional Neural Networks

How can we help a computer ‘see’?

Grey scale



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	94	6	10	33	48	106	159	181
256	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	257	299	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

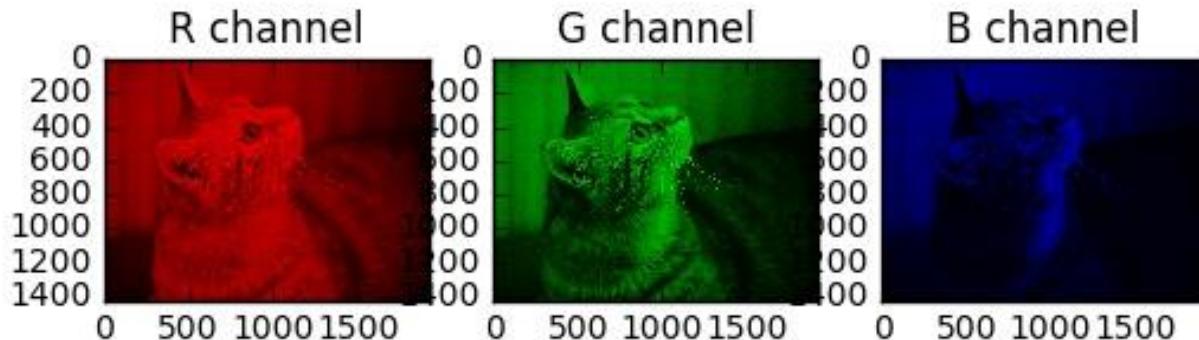
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	94	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Convolutional Neural Networks

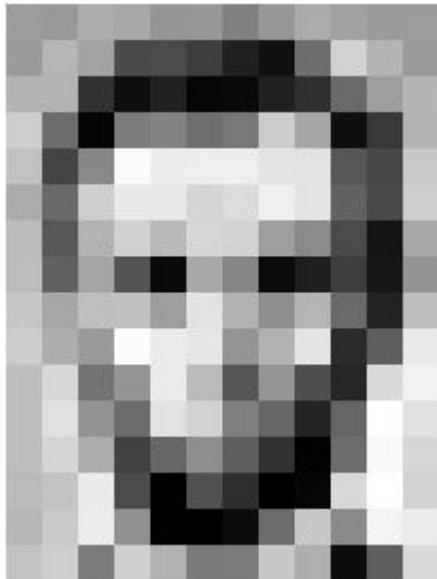
How can we help a computer ‘see’?

For colour images: break into RGB channels

We get a 3d matrix that describes our image.



Convolutional Neural Networks



Input image



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	198	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	236	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

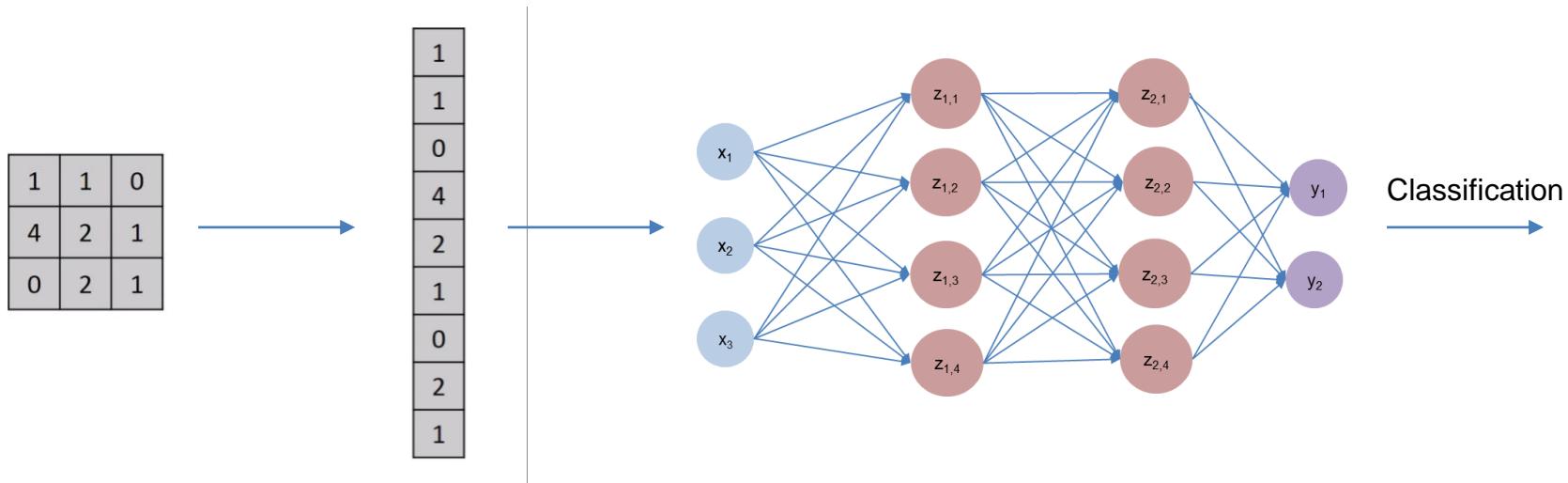
Pixel representation

Classification

Lincoln
Trump

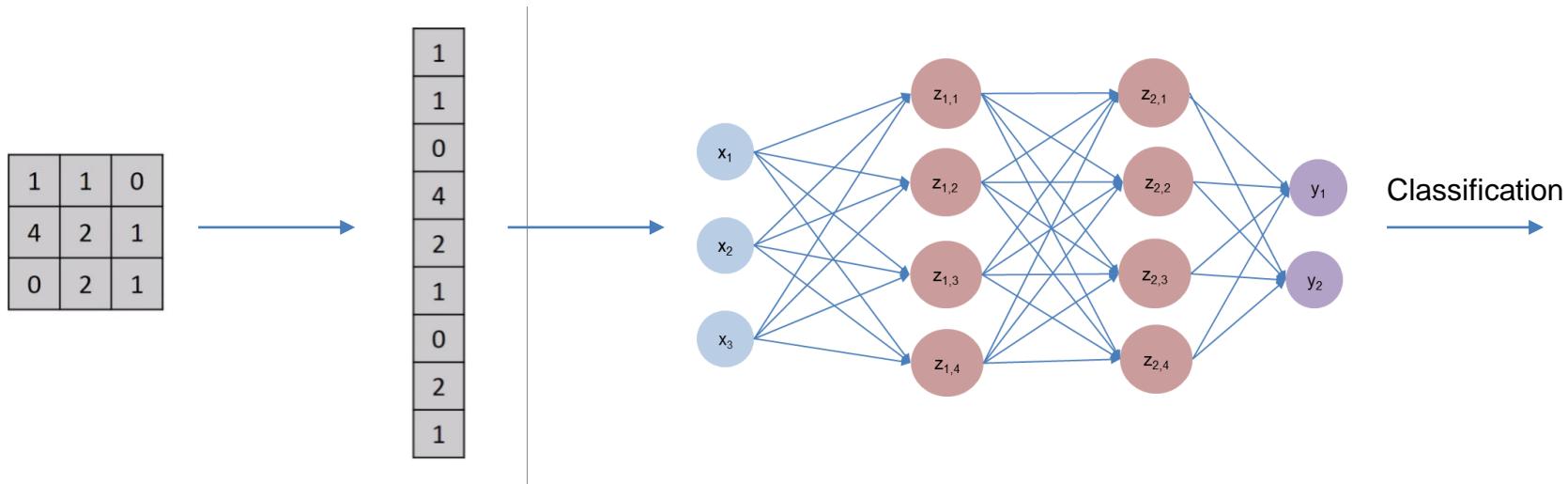
Why the convolutions?!

We could vectorise the image?
Stick it straight into an ANN



Why the convolutions?!

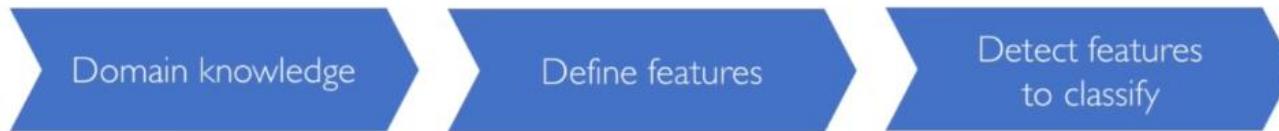
We could vectorise the image?
Stick it straight into an ANN



- We lose the spatial relationships between pixels!
- We introduce a very large number of parameters

Why the convolutions?!

We could define features manually...



Why the convolutions?!

We could define features manually...



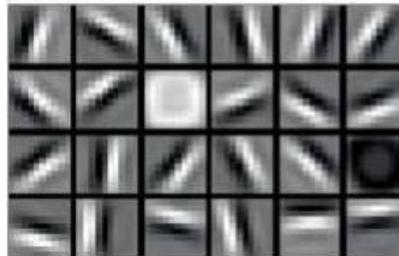
Learning feature representations

A hierarchy of features that help us describe the image

Millions of images



Low level features



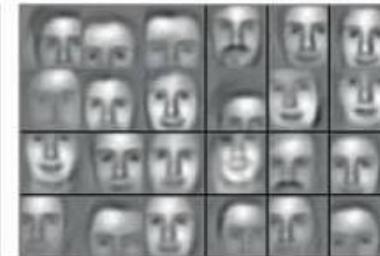
Lines & Edges

Mid level features



Eyes & Nose & Ears

High level features



Facial Structure

Identifying similar objects

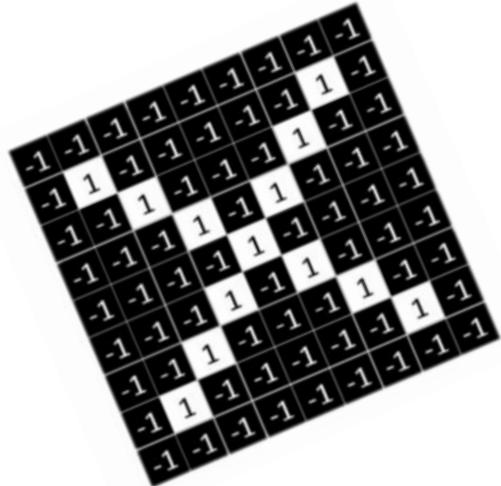
Both images show an X.

Directly comparing the elements of each image would suggest they are different images
We want to identify common features across the two images!

Identifying similar objects

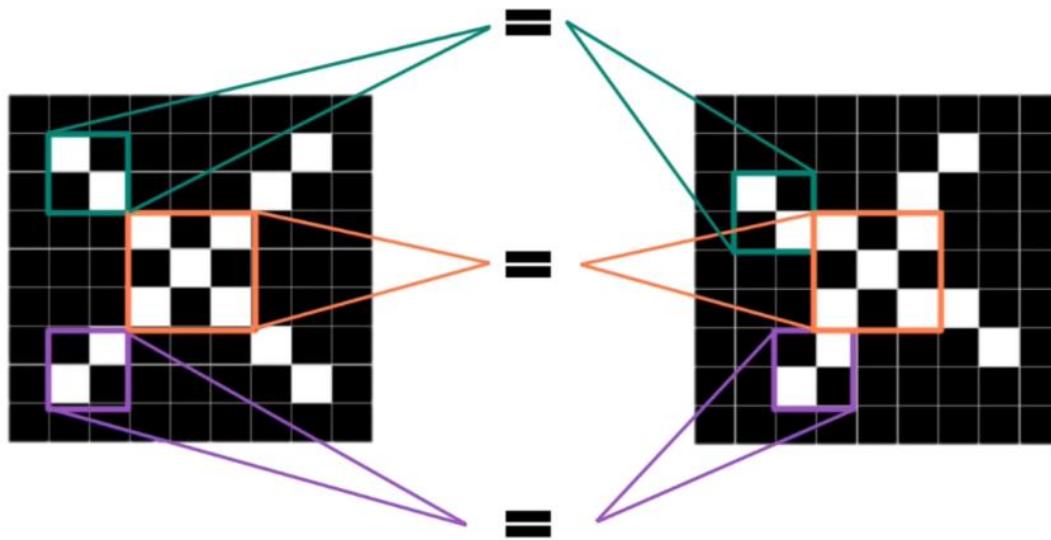
Both images show an X.

Directly comparing the elements of each image would suggest they are different images
We want to identify common features across the two images!



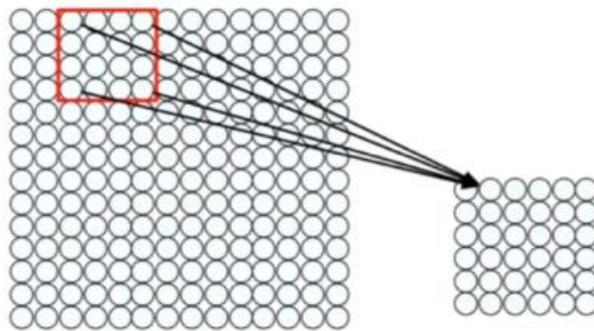
Identifying similar objects

There are common features in both that we can learn.



Learning feature representations

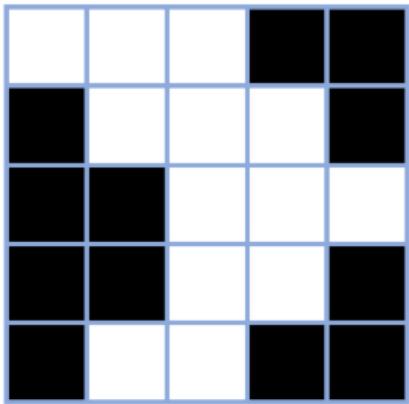
Apply a filter → Shift filter across image → Construct a new ‘convolved’ matrix



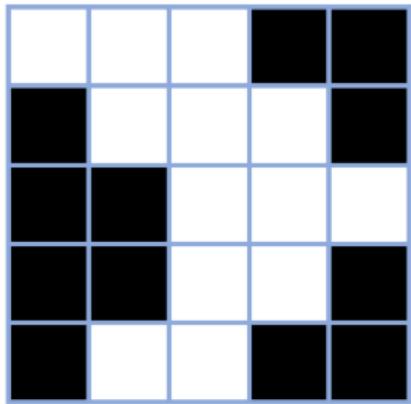
Example:

- 4x4 filter
- Apply filter to the input image
- Element-wise multiplication of pixels with filter matrix
- Shift by 1 pixel to the right and repeat

Convolution operation



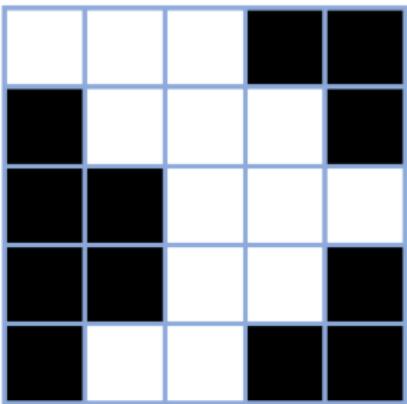
Convolution operation



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$$kernel = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Convolution operation



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

$$kernel = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

1 <small>×1</small>	1 <small>×0</small>	1 <small>×1</small>	0	0
0 <small>×0</small>	1 <small>×1</small>	1 <small>×0</small>	1	0
0 <small>×1</small>	0 <small>×0</small>	1 <small>×1</small>	1	1
0	0	1	1	0
0	1	1	0	0

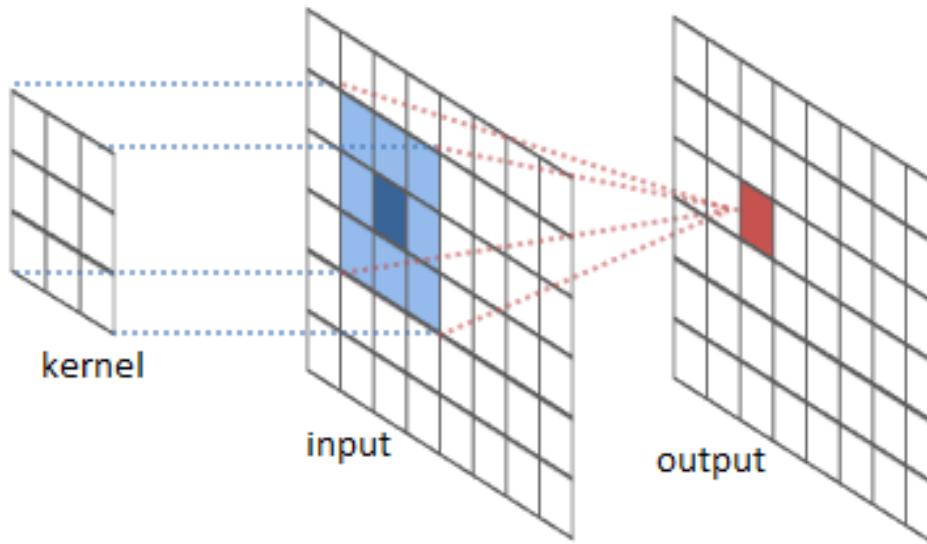
Image

4		

Convolved Feature

Kernels

3x3 filter Input image Convolved output



Kernels



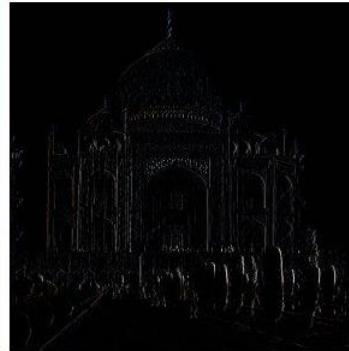
(a) Original image.

0	0	0
0	1	0
0	0	0



(b) Blurred.

1	1	1
1	1	1
1	1	1



(c) Detect vertical edges.

0	0	0
-1	1	0
0	0	0



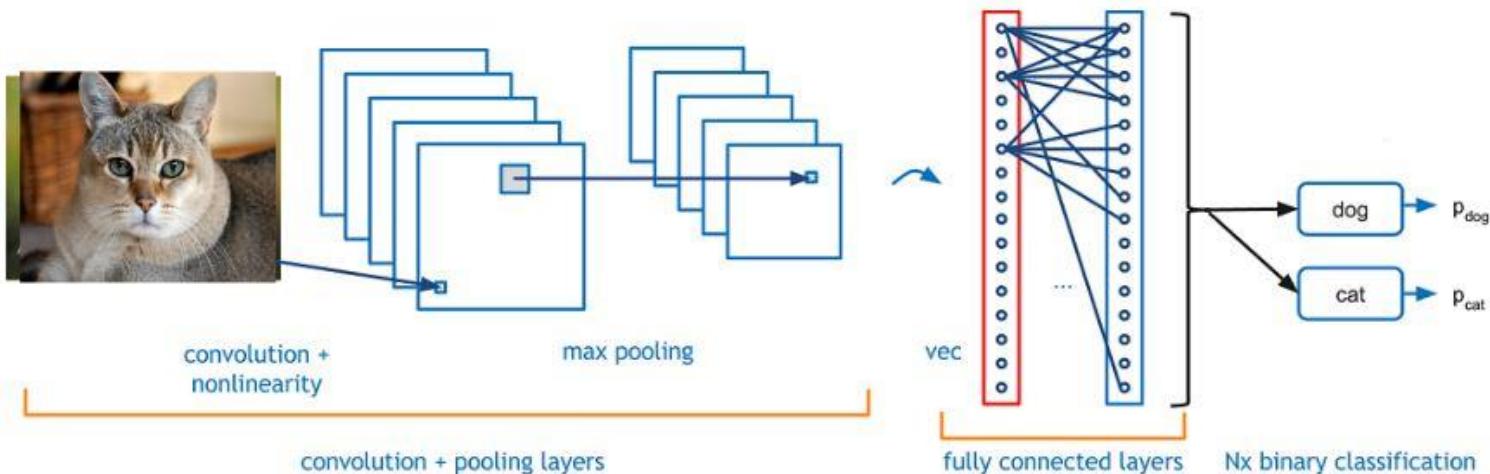
(d) Detect all edges.

0	1	0
1	-4	1
0	1	0

c) we subtract two adjacent pixels. When side by side pixels are similar, this gives us approximately zero. On edges, however, adjacent pixels are very different in the direction perpendicular to the edge. Knowing that results differs from zero will result in brighter pixels, you can already guess the result of this type of kernel.

4 main steps for CNN classification

1. Convolution – Apply filters (*we learn the filters!*)
2. Non-linear function – Often ReLU
3. Pooling operation – reduce matrix size, often using max pooling
4. Fully connected MLP



Learning weights of filter

In lecture 1 we saw a neuron in a hidden layer.

Each neuron, weighted combination of inputs plus a bias put through a non-linear function

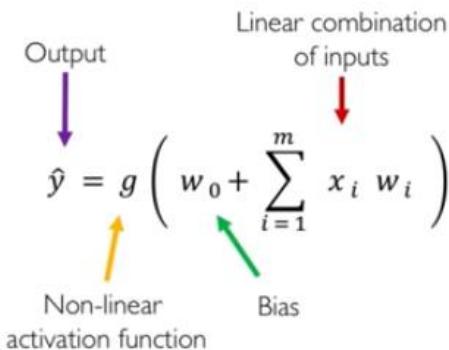
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

Output

Linear combination of inputs

Non-linear activation function

Bias



Learning weights of filter

In lecture 1 we saw a neuron in a hidden layer.

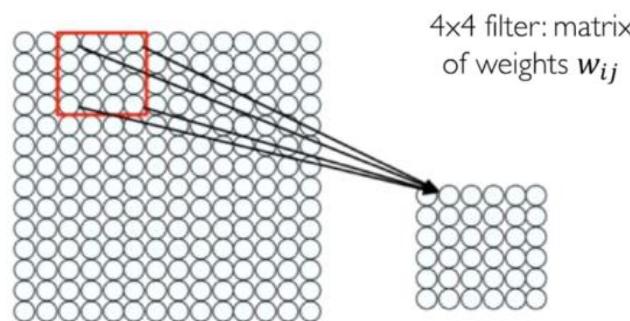
Each neuron, weighted combination of inputs plus a bias put through a non-linear function

$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

Output
Non-linear activation function
Linear combination of inputs
Bias

Here each neuron only ‘sees’ a patch before it. Not fully connected.
Defines local connectivity.

1. Apply a window of weights
2. Computer linear combinations
3. Activating with non-linear function

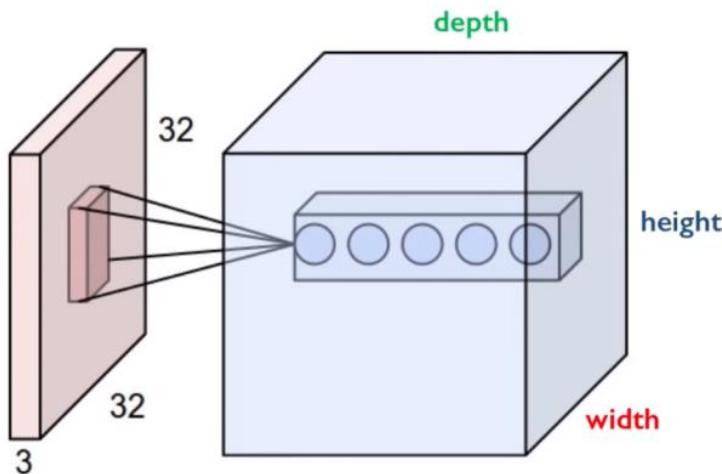


$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p, j+q} + b$$

for neuron (p,q) in hidden layer

We can learn many filters

Using multiple features gives us an output **volume** instead of matrix.



Layer Dimensions:
 $H \times W \times D$

H and D are spatial dimensions of our images

D = number of filters

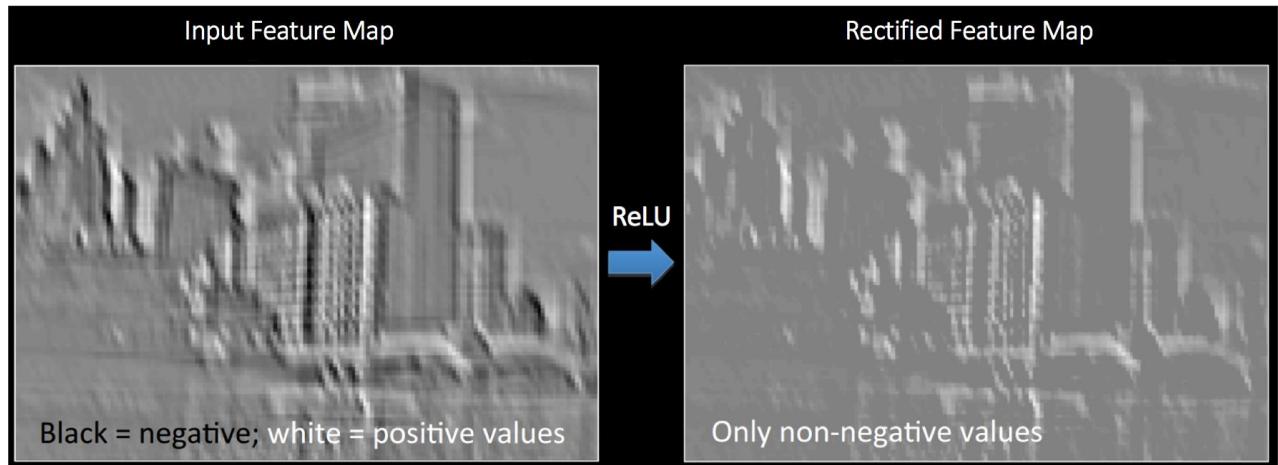
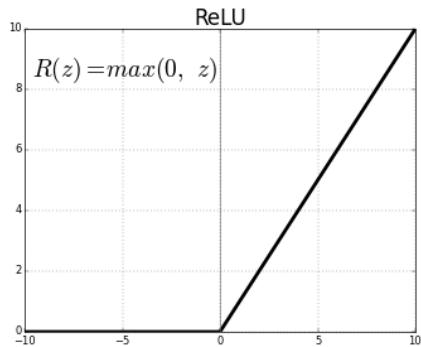
Stride:
Filter step size

Receptive field:
Locations in input image that a node is connected to.

Introducing non-linearity

ReLU function is most common for CNN's

It sets all negative values in our feature map to zero!



Pooling operations

Rectified feature map			
1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

max pooling with 2x2 filters
and stride 2

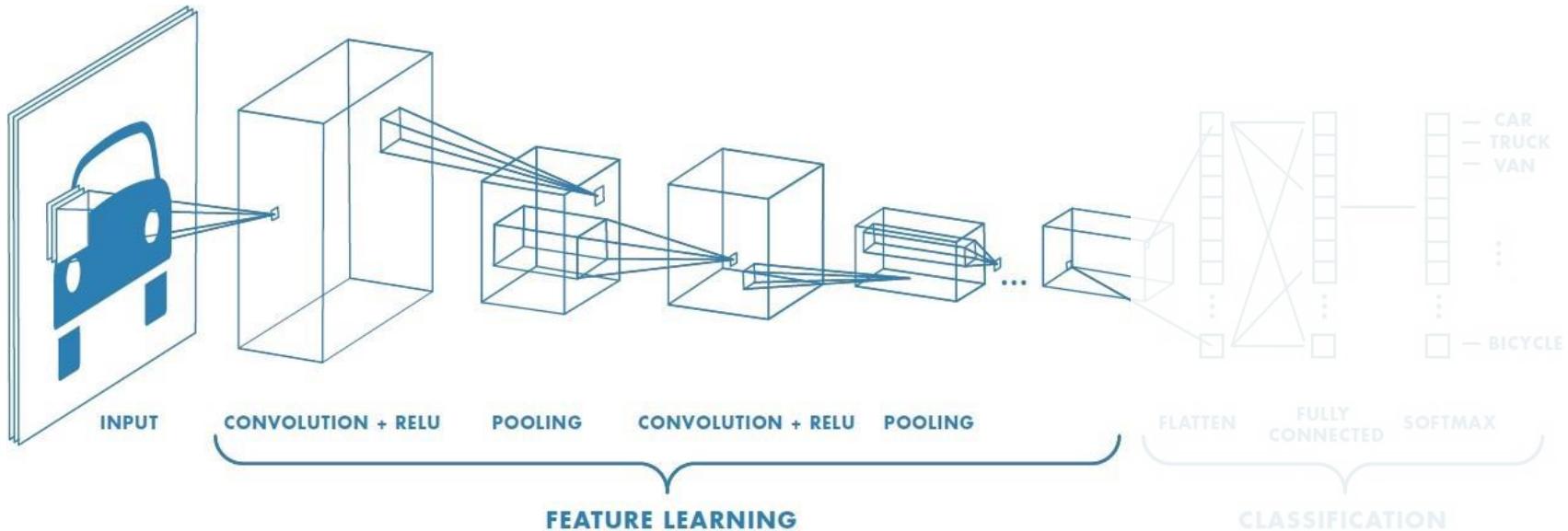
$$\text{Max}(3, 4, 1, 2) = 4$$

6	8
4	7

Why?

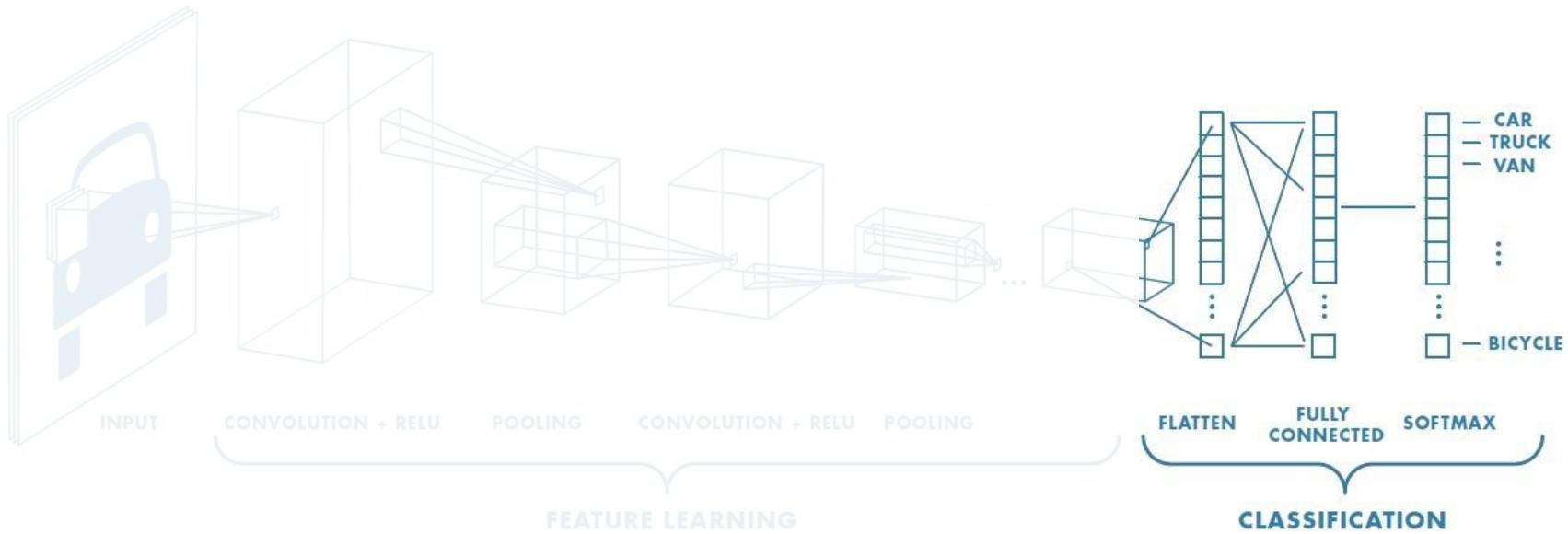
1. Reduce dimensionality
2. Preserve spatial invariance

CNN as a whole



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** (real world data isn't linear!)
3. Reduce dimensionality and preseve
4. Preserve spatial invariance using **pooling**

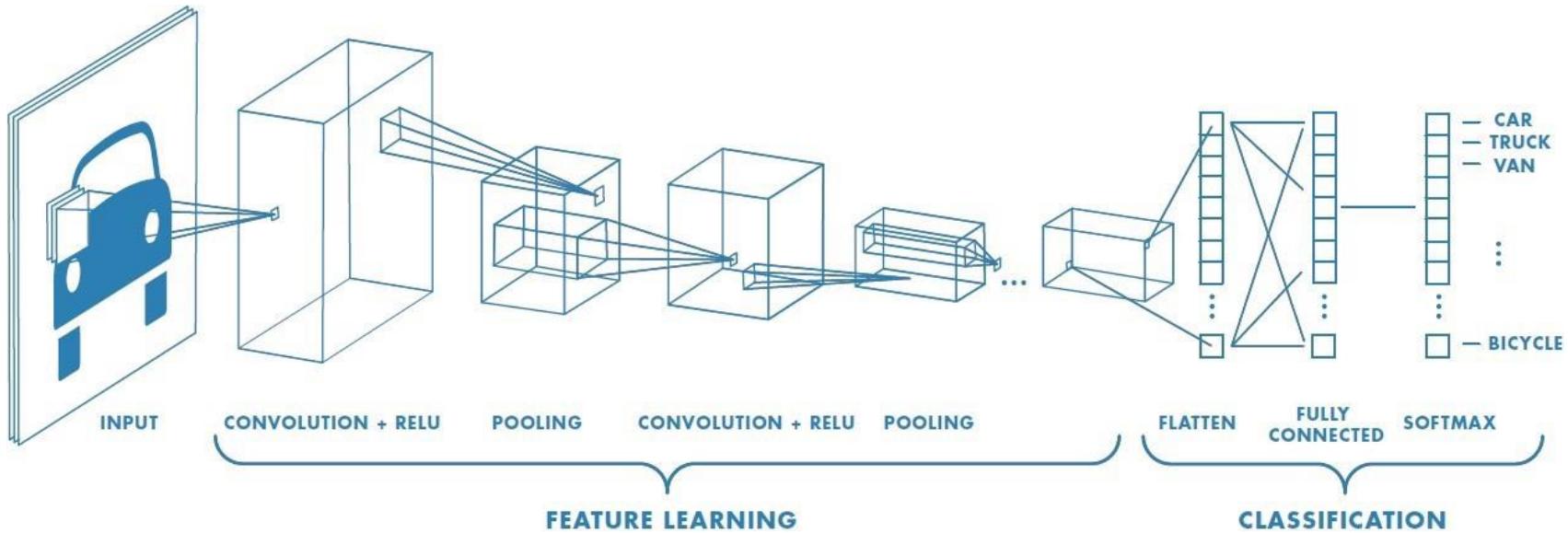
CNN as a whole



1. Outputs from feature learning are input into fully connected ANN
2. Fully connected layers users the generated features for classifying the input image
3. Express output as a probability of image belonging to a class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNN as a whole

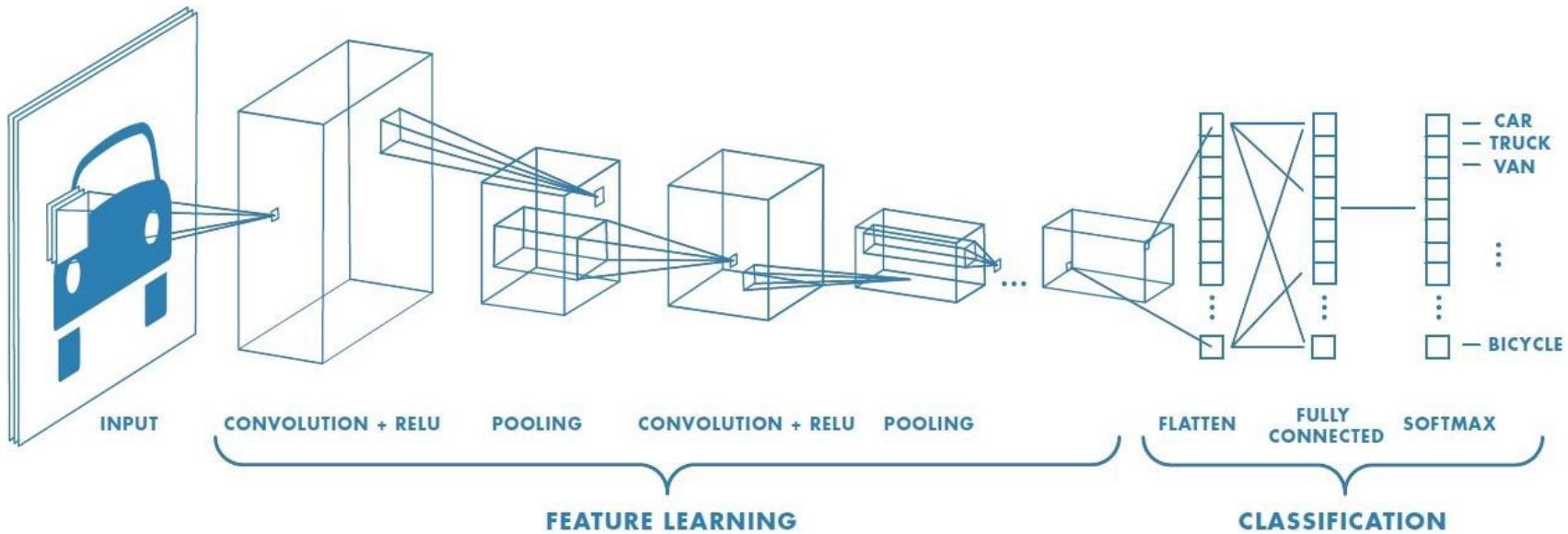


How do we train? – Exactly the same as the ANN. Use back propagation!!

Use some loss function – cross entropy loss.

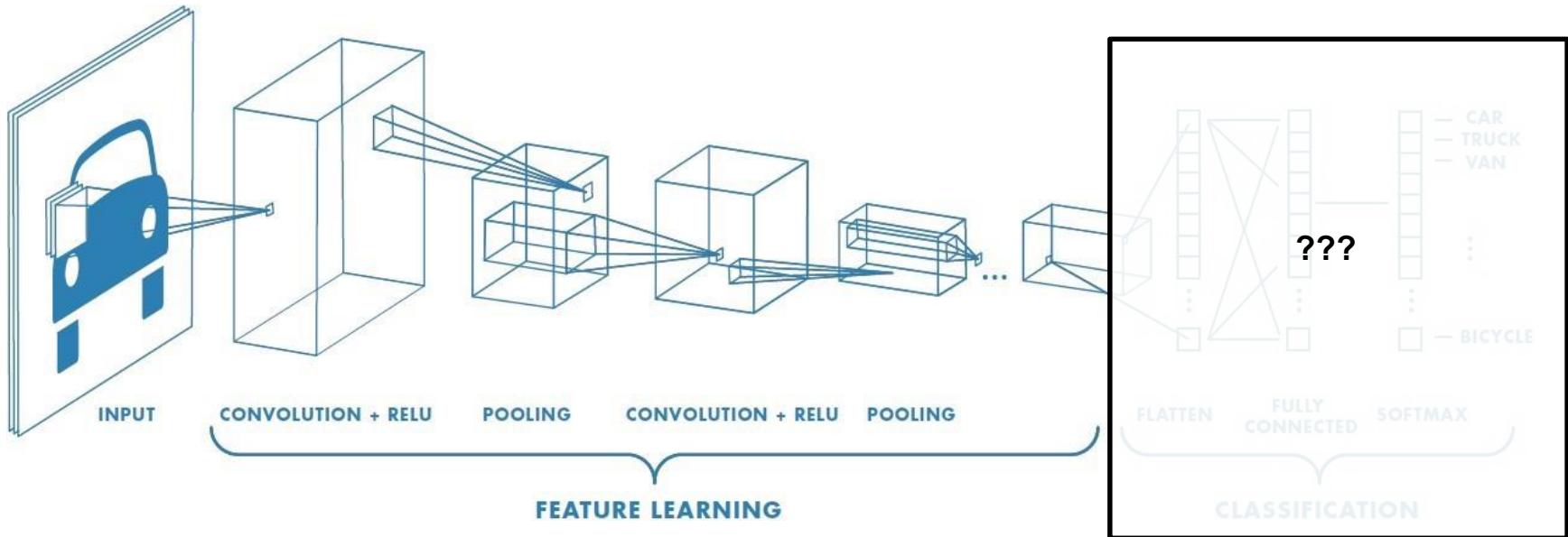
$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

CNN as a whole



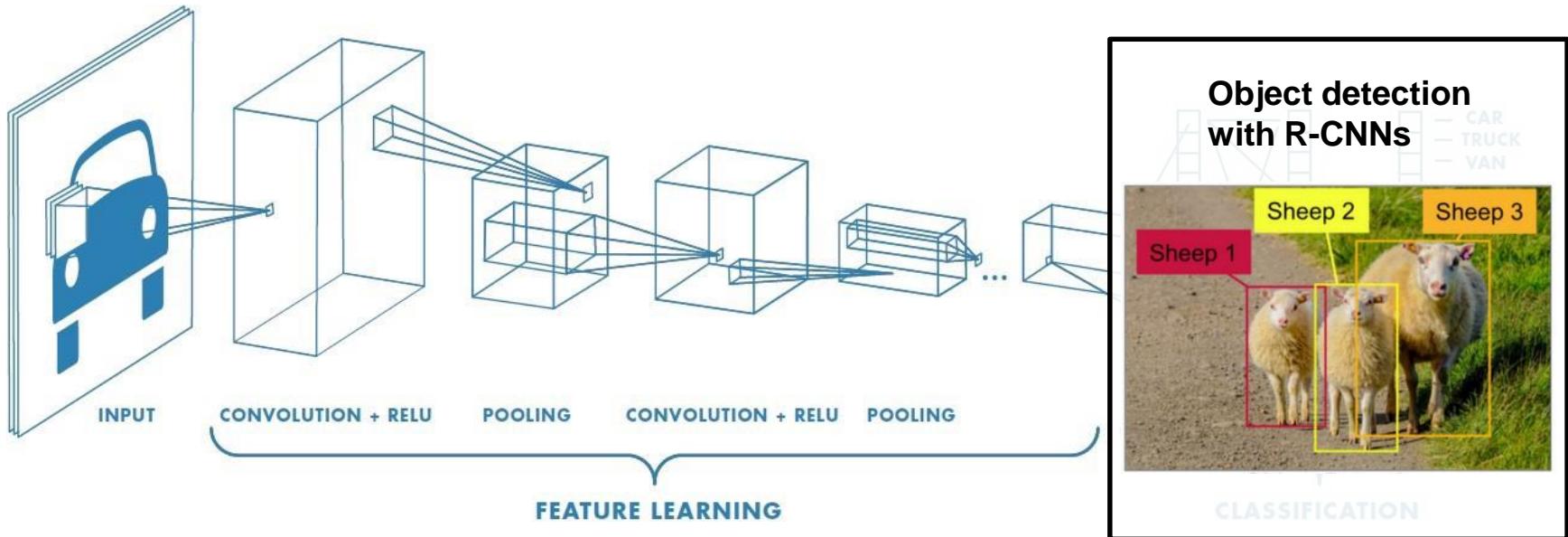
This full architecture is purposed for **image classification!**

An Architecture for many applications



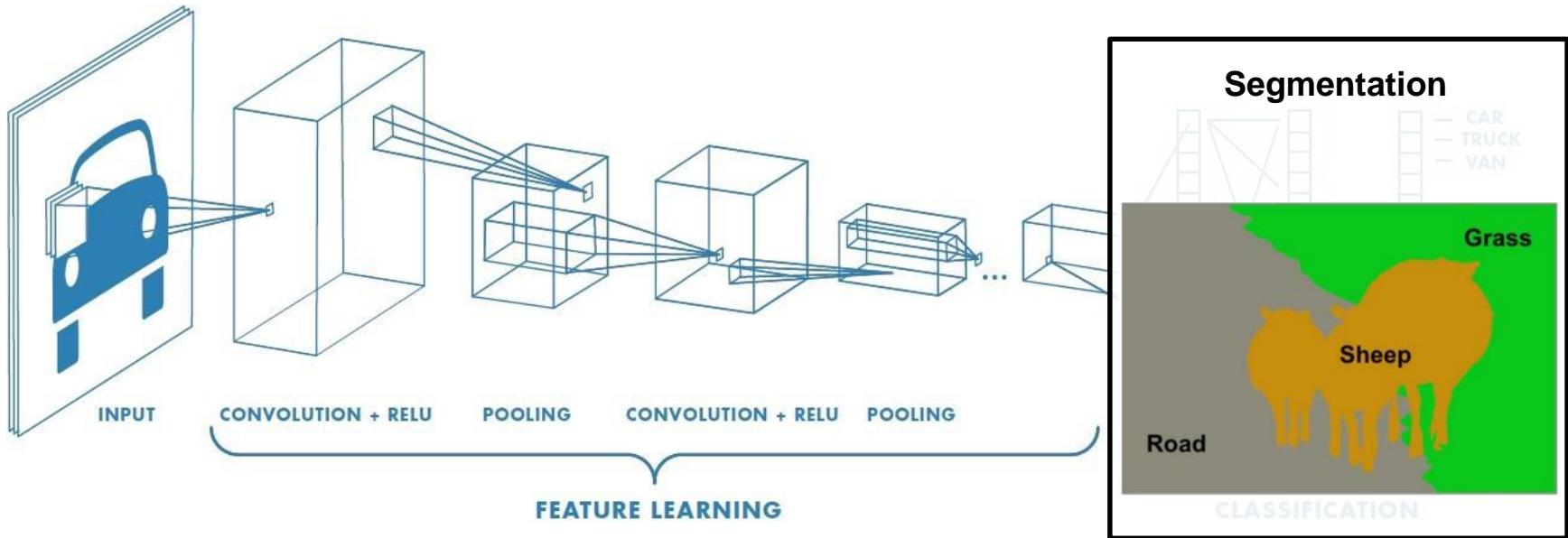
The convolution and pooling stages (*feature learning*) can be used for many other applications!

An Architecture for many applications



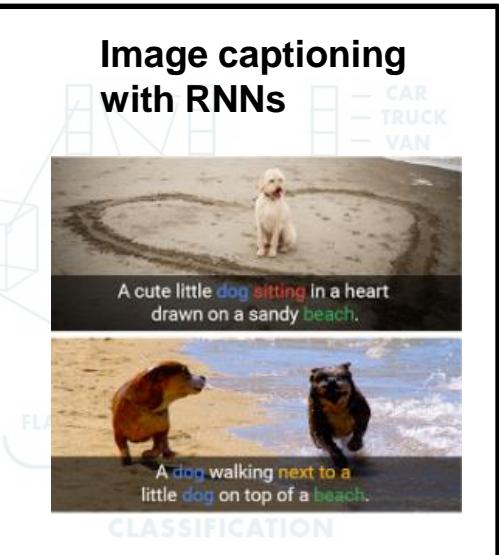
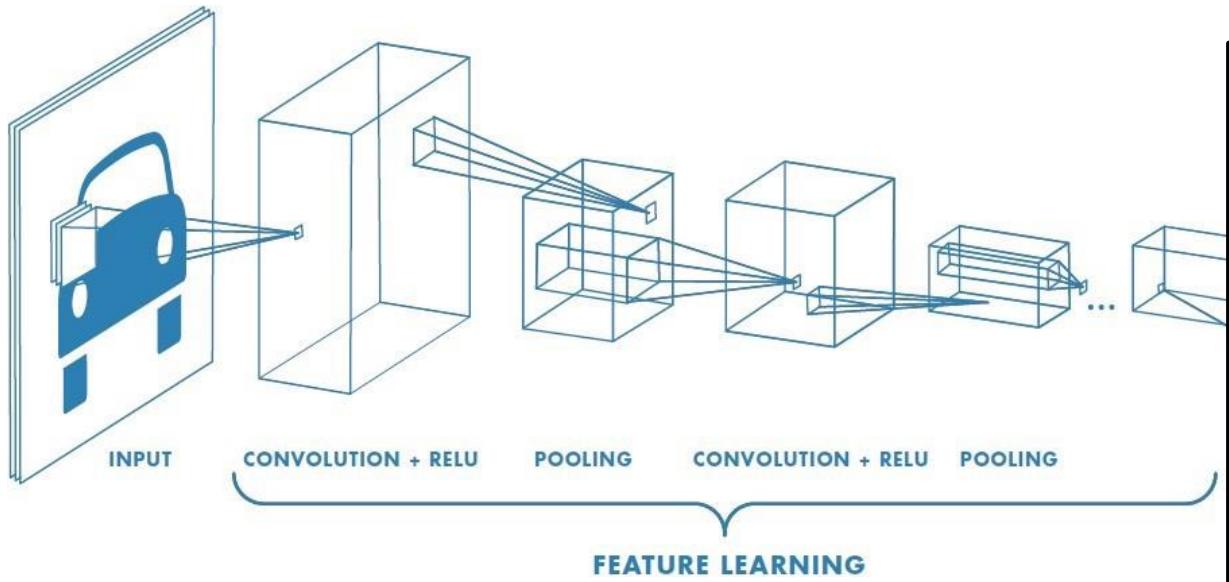
The convolution and pooling stages (*feature learning*) can be used for many other applications!

An Architecture for many applications



The convolution and pooling stages (*feature learning*) can be used for many other applications!

An Architecture for many applications

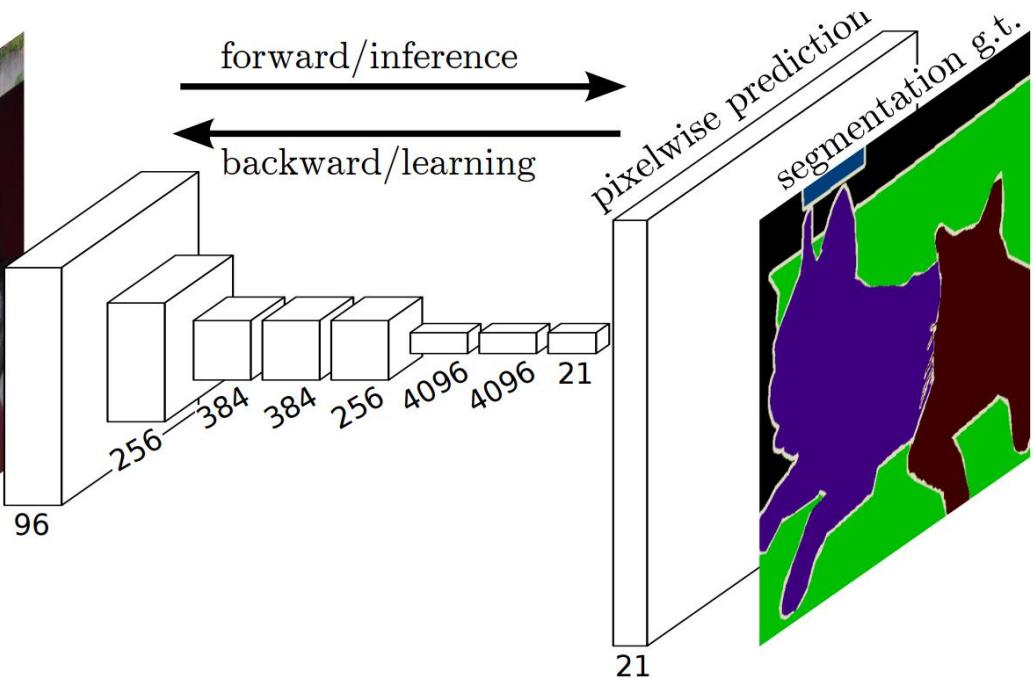


The convolution and pooling stages (*feature learning*) can be used for many other applications!

Semantic Segmentation

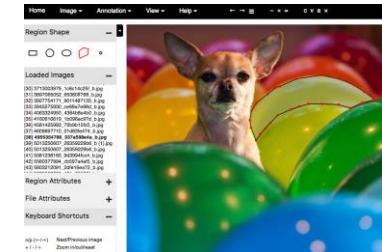
Assign each pixel in the image a class!

This has the flavour of an autoencoder



Creating your labelled datasets can be time consuming....

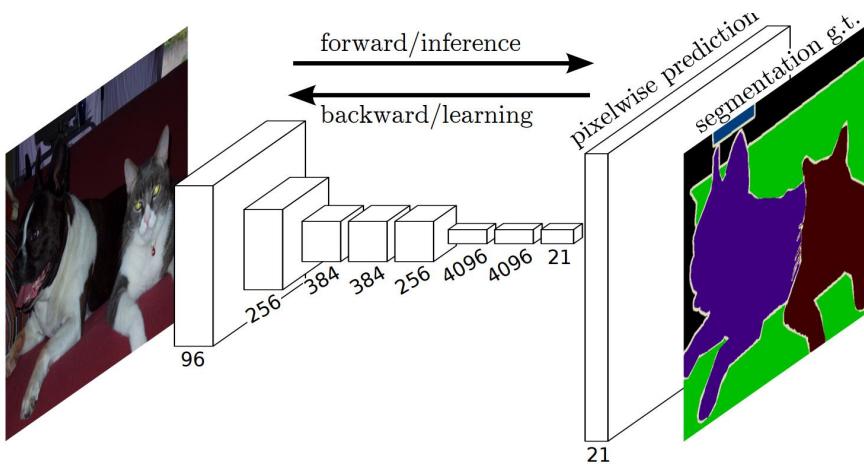
Annotate images manually...



Semantic Segmentation

Assign each pixel in the image a class!

This has the flavour of an autoencoder

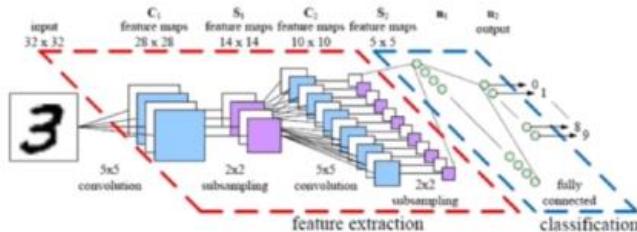


We used **down sampling** (convolutions and max pooling) to capture semantic/contextual information

We then implement **up sampling** to recover spatial information! Take our learned features and map them back into the original spatial image.

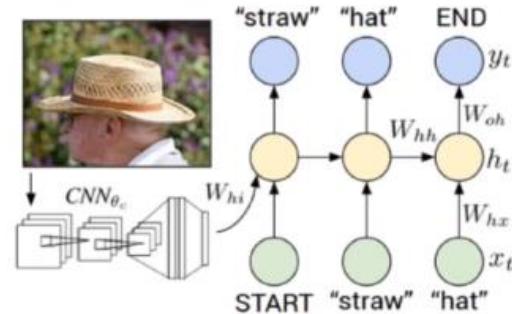
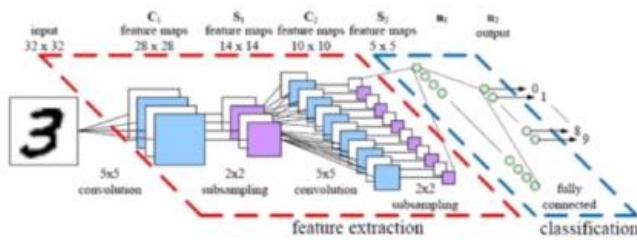
Object Detection and Instance Segmentation

Identify objects and the pixels that relate to that object!



Object Detection and Instance Segmentation

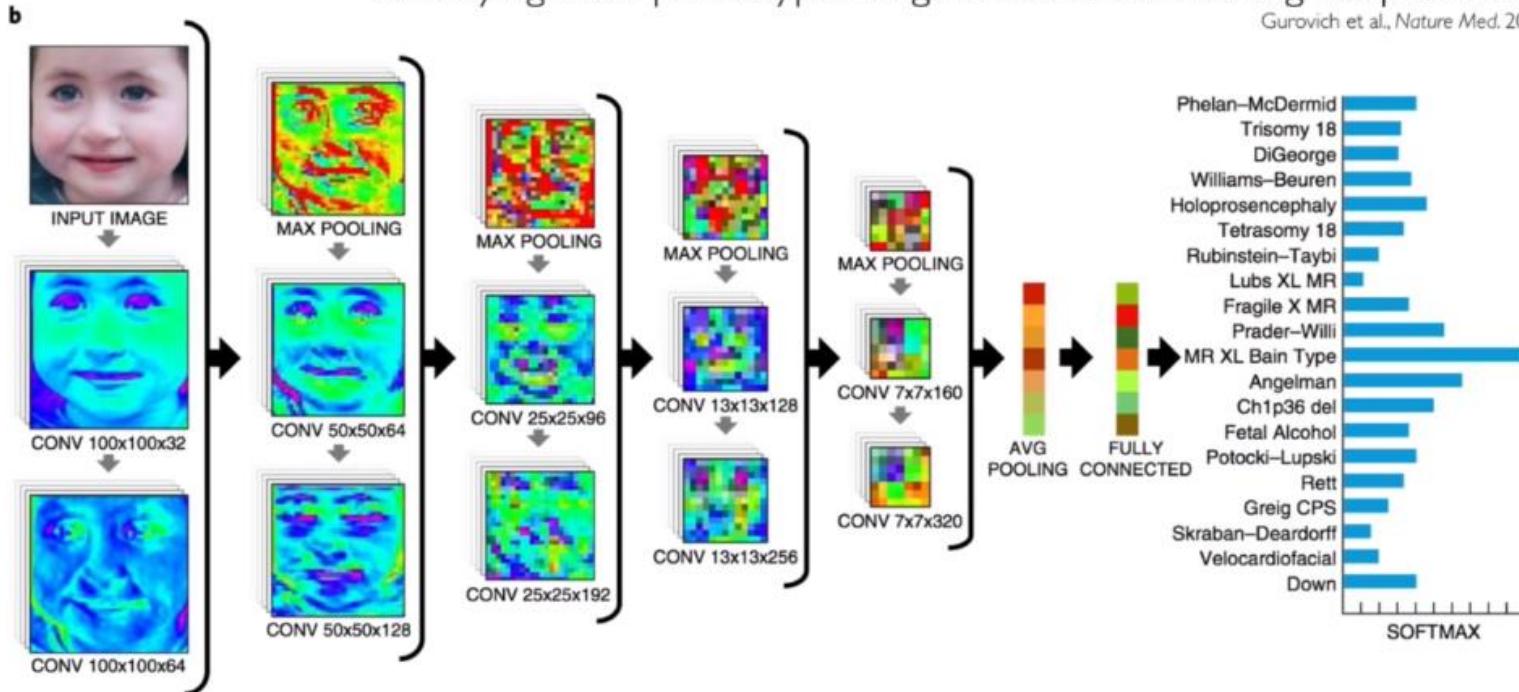
Identify objects and the pixels that relate to that object!



Application of CNN

Identifying facial phenotypes of genetic disorders using deep learning

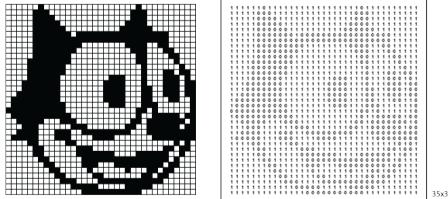
Gurovich et al., *Nature Med.* 2019



Quick Review

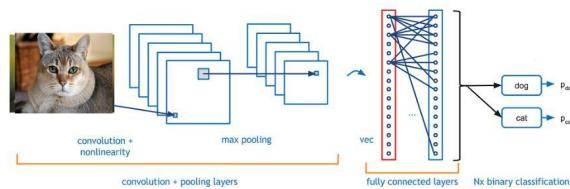
Foundations

- Representing images for computer vision
- Convolution feature learning and pooling operations



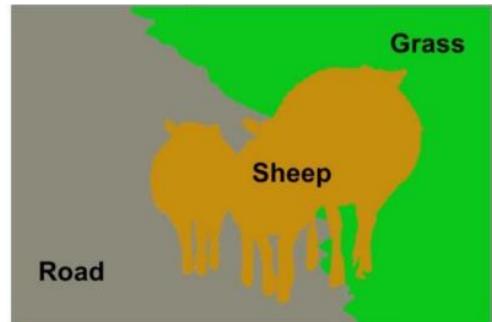
CNNs

- The complete architecture
- Stacking multiple kernels into 3-dimensions



Applications

- Segmentation
- Object detection
- Image captioning



Recurrent Neural Networks

We have considered images... what about time-series data?

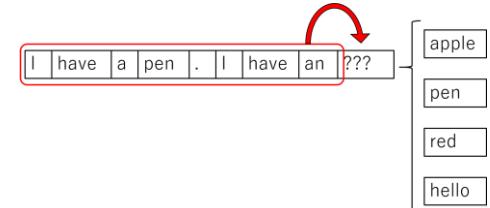
Stock Data



Audio



Text



Predicting next step in a trajectory

I took my dog for a

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

Define a window of words
to make a prediction of
the next word.

One-hot encode the
words ‘for’ and ‘a’

[1 0 0 0 0 0 1 0 0]

for

a



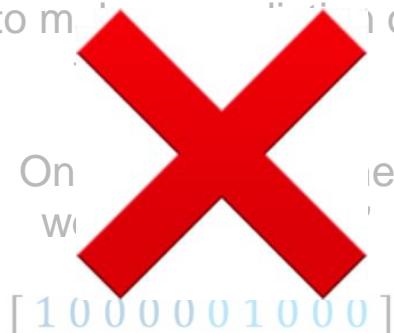
prediction

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

Define a window of words
to make prediction of



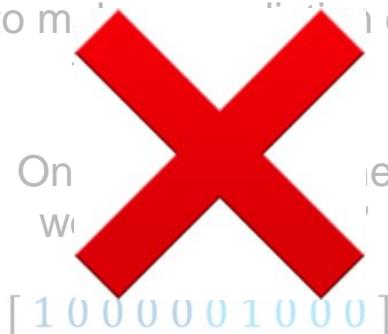
'for' and 'a' aren't particularly predictive of the next word 'walk'. We would need to define a larger window

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

Define a window of words
to model the context of



Can't model long term dependencies

"China is where I grew up, but I now live in London. I speak fluent Chinese."

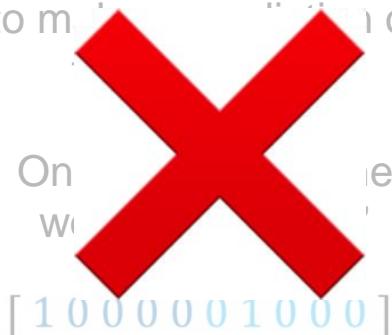
We need to retain information from the distant past to predict the next correct word.

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

Define a window of words
to make prediction of



prediction

Idea 2

Use entire sequence as
set of counts

'bag of words'

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]



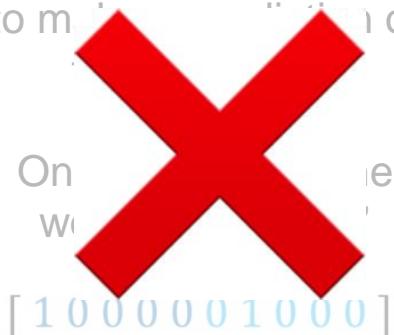
prediction

Predicting next step in a trajectory

I took my dog for a walk

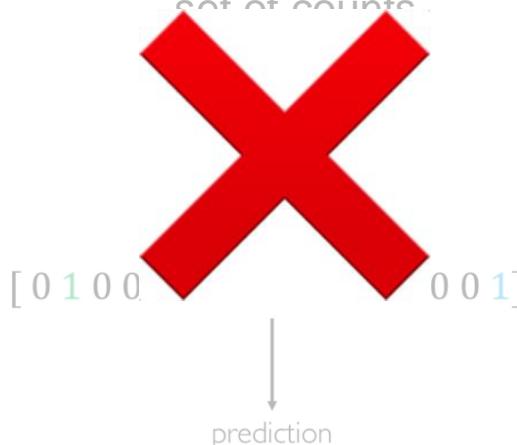
Idea 1

Define a window of words
to make prediction of



Idea 2

Use entire sequence as
set of counts.



It lost the sequential
information!

e.g.

The food was good, not bad at all.

Vs

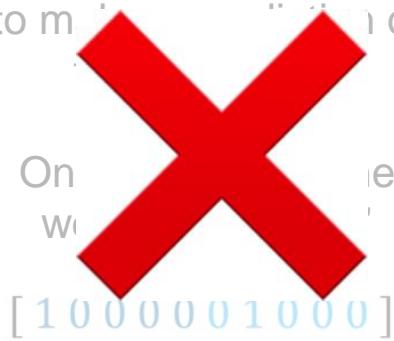
The food was bad, not good at all.

Predicting next step in a trajectory

I took my dog for a walk

Idea 1

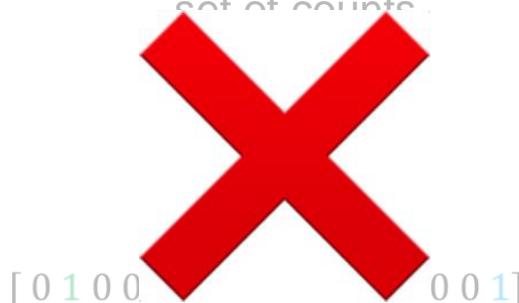
Define a window of words
to make prediction of



prediction

Idea 2

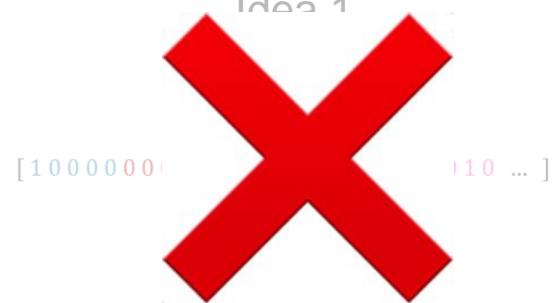
Use entire sequence as
set of counts.



Introduces separate parameters for words
at different position in sequence.

Idea 3

Extend the window for
Idea 1



Features we learn about the sequence won't
transfer if they appear elsewhere in the sequence.

Predicting next step in a trajectory

Traditional feed-forward neural network isn't satisfactory for this problem.

We saw that standard ANN architecture wasn't suitable for images

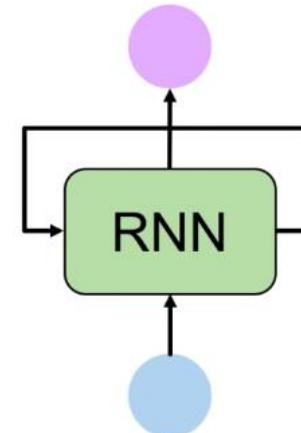
Similar problem for time-series!

Design Criteria

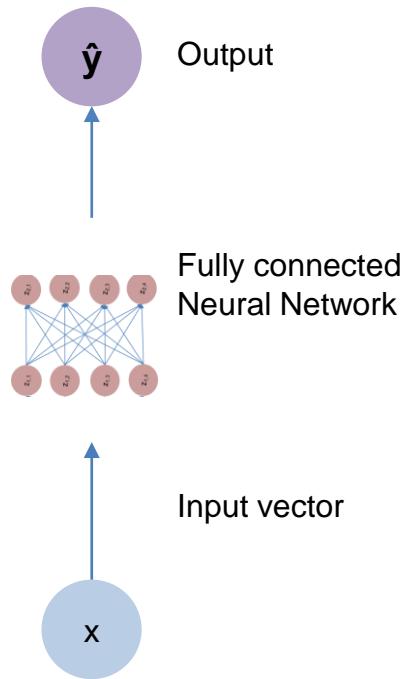
We want a model that can:

1. Take **variable length** sequences
2. Track ('remember') **long term dependencies**
3. Maintain information about **order**
4. **Share parameters** across the sequence

Use a Recurrent Neural Network (RNN).



Sequence modelling

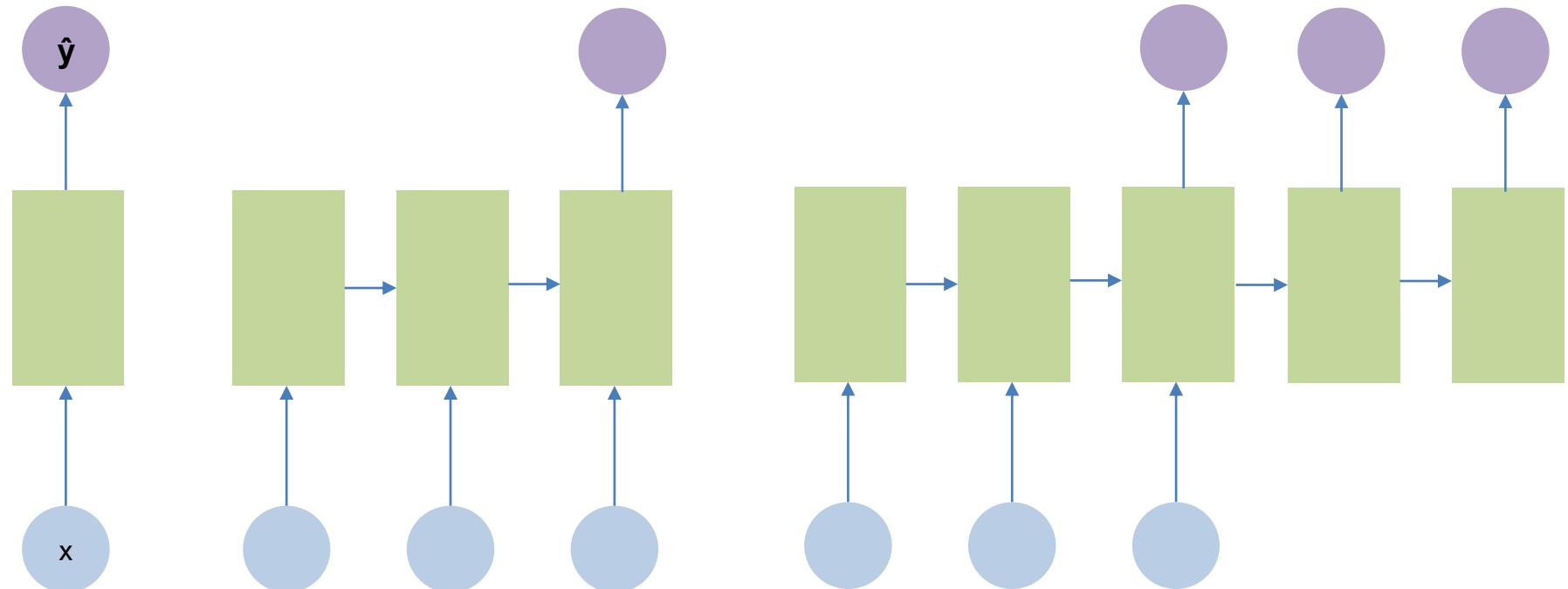


Sequence modelling



Our basic
Neural Network

Sequence modelling



Our basic
Neural Network

Many to one
e.g. Sentiment Analysis

Many to many
e.g. predict stock prices

Recurrent Neural Network



Our basic
Neural Network

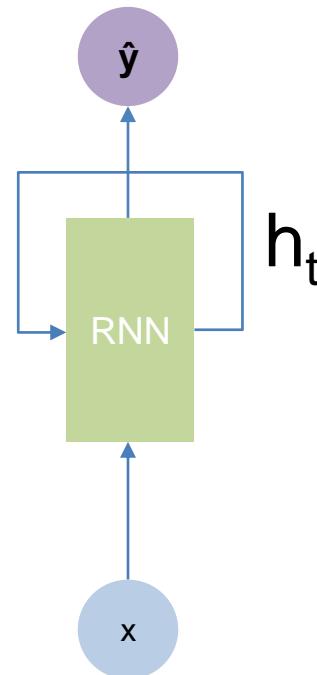
Recurrent Neural Network



Our basic
Neural Network

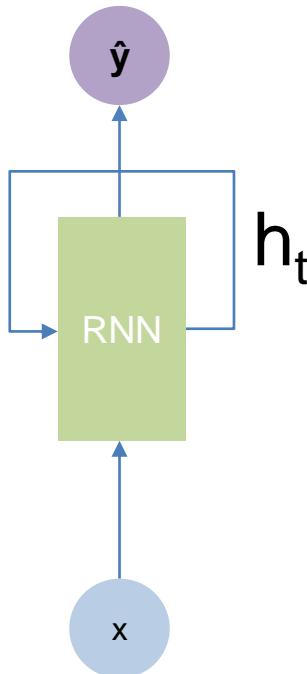
Passing
information
internally from one
step in the network
to the next.

This loop creates a
recurrent relation.



Recurrent neural
network

Recurrent Neural Network



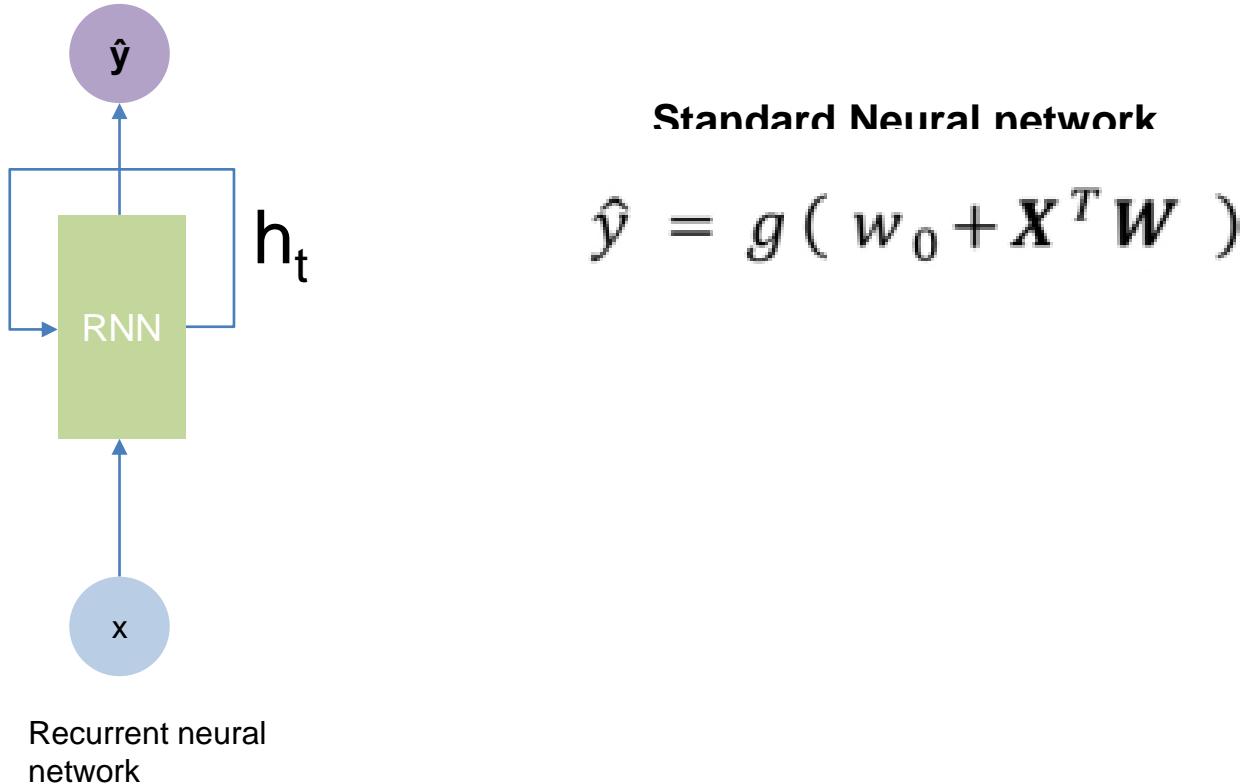
Recurrent neural network

$$h_t = f_W(h_{t-1}, x_t)$$

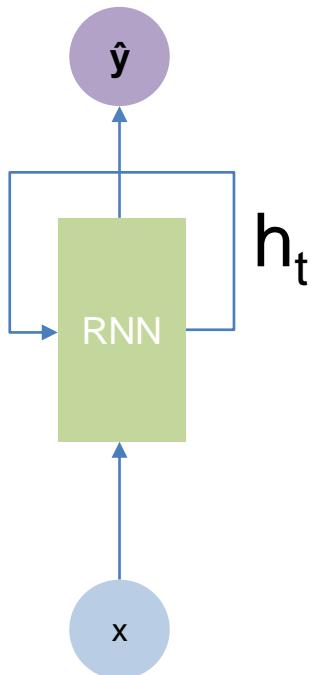
new state function parameterized by W
old state input vector at time step t

Some weighted function (learnt weights) that updates the state using the current state and the next step in the sequence.

Recurrent Neural Network



Recurrent Neural Network



Recurrent neural
network

Standard Neural network

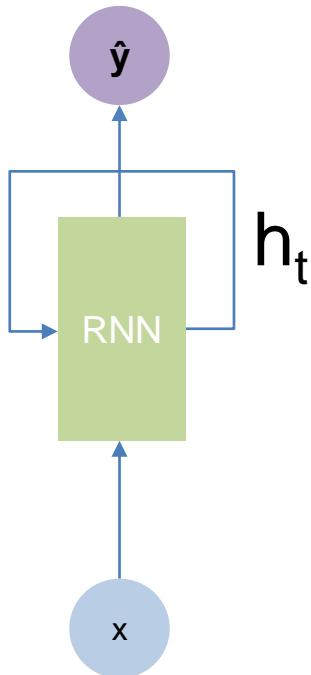
$$\hat{y} = g(w_0 + X^T W)$$

Recurrent Neural Network

Update Hidden State

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Recurrent Neural Network



Recurrent neural
network

Recurrent Neural Network

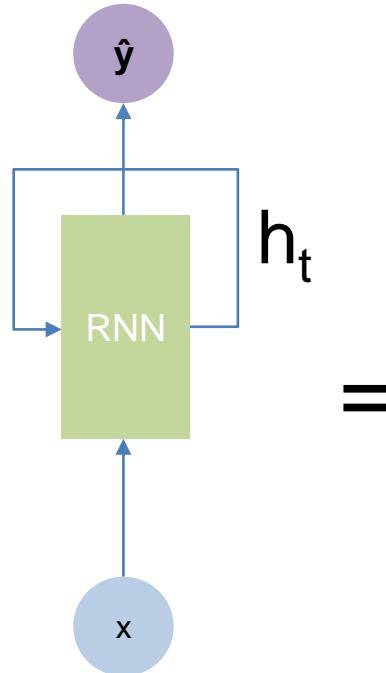
Update Hidden State

$$h_t = \tanh(\mathbf{W}_{hh}h_{t-1} + \mathbf{W}_{xh}x_t)$$

Two weight matrices:

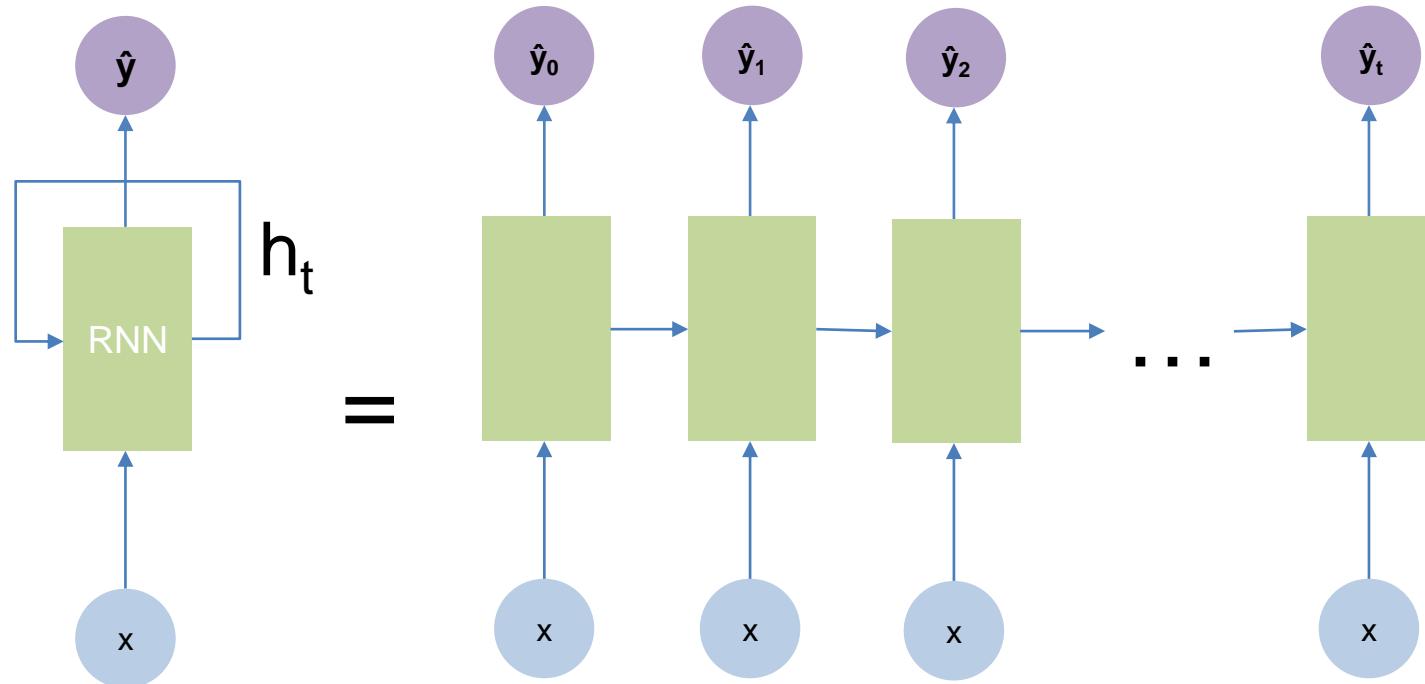
- 1) Applied to hidden state
- 2) Applied to input

Unravelling RNN



Recurrent neural
network

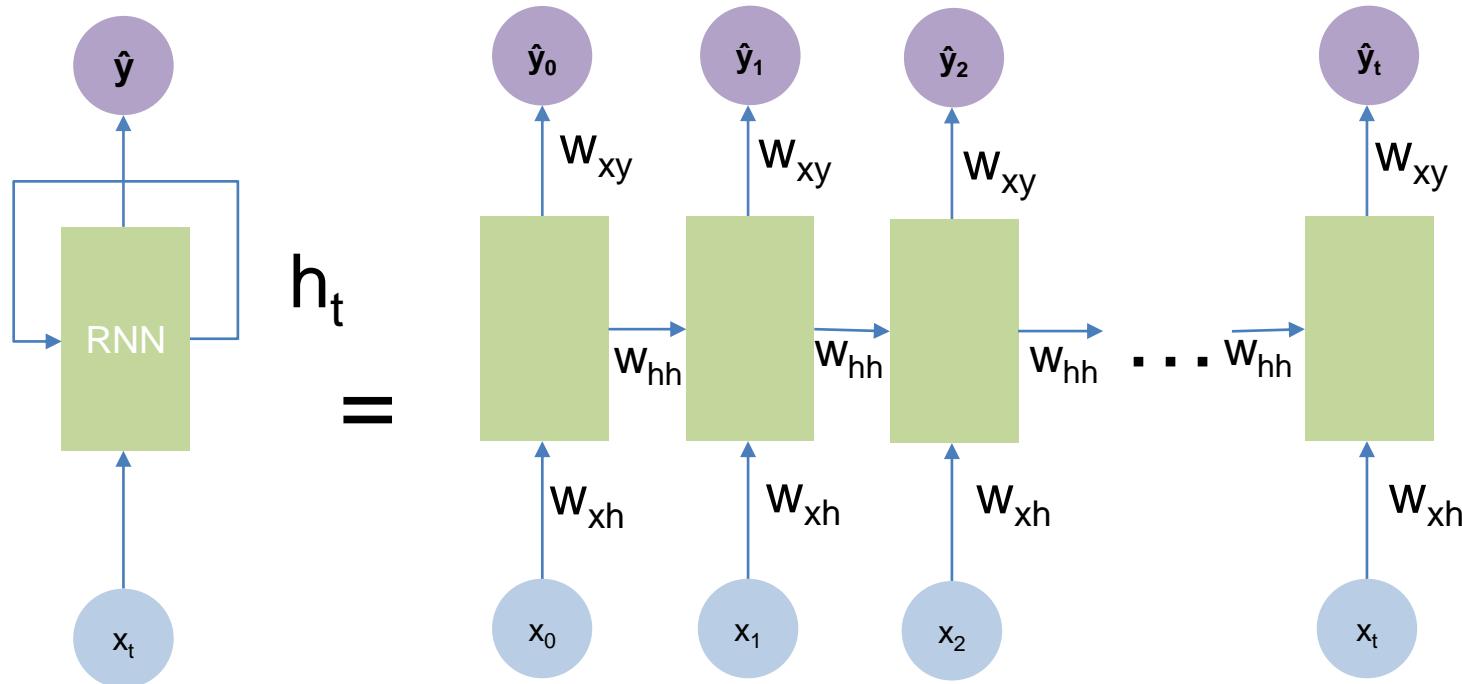
Unravelling RNN



Recurrent neural
network

The Loop can be seen as a chain like structure

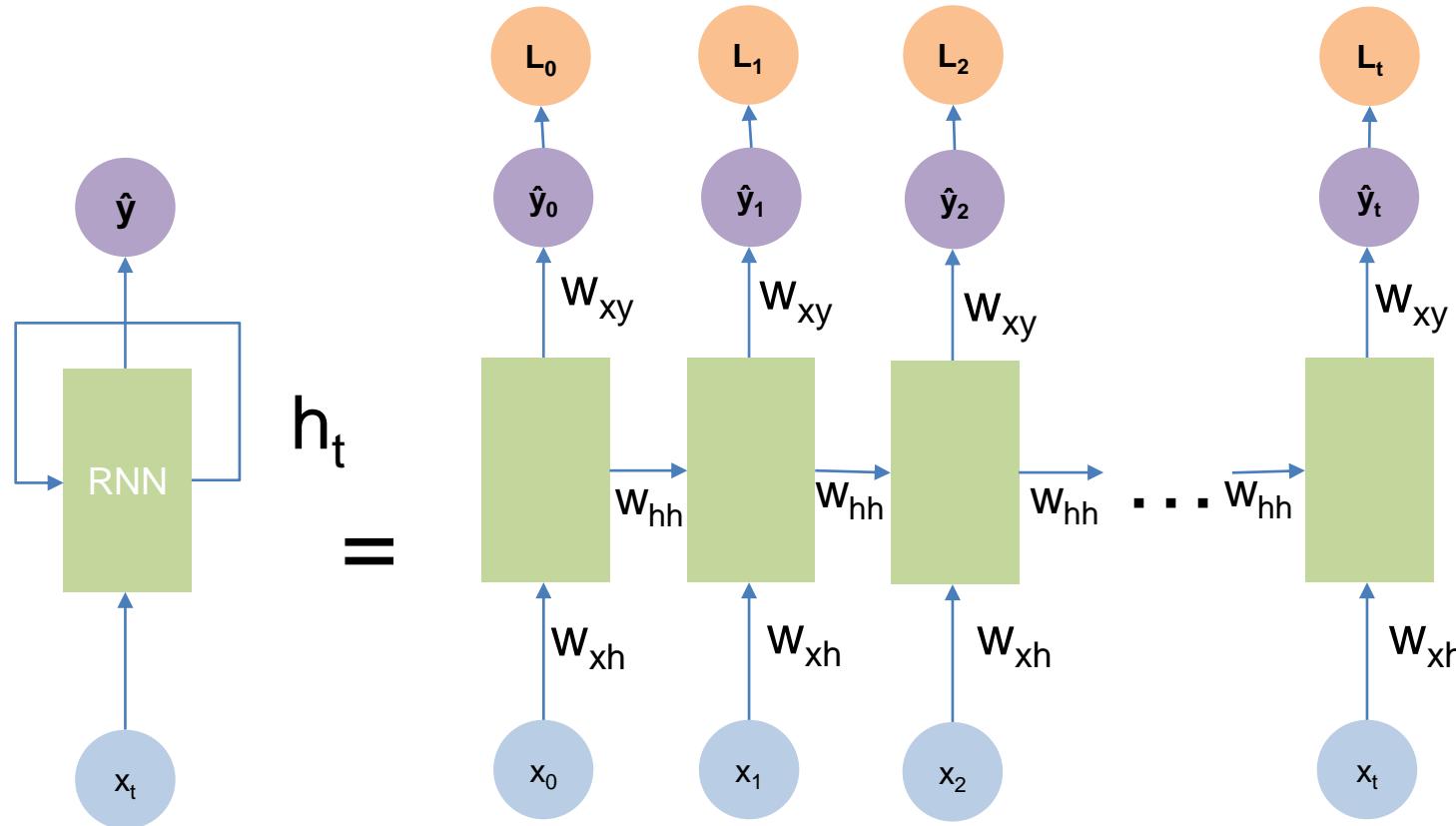
Unravelling RNN



Recurrent neural
network

The Loop can be seen as a chain like structure

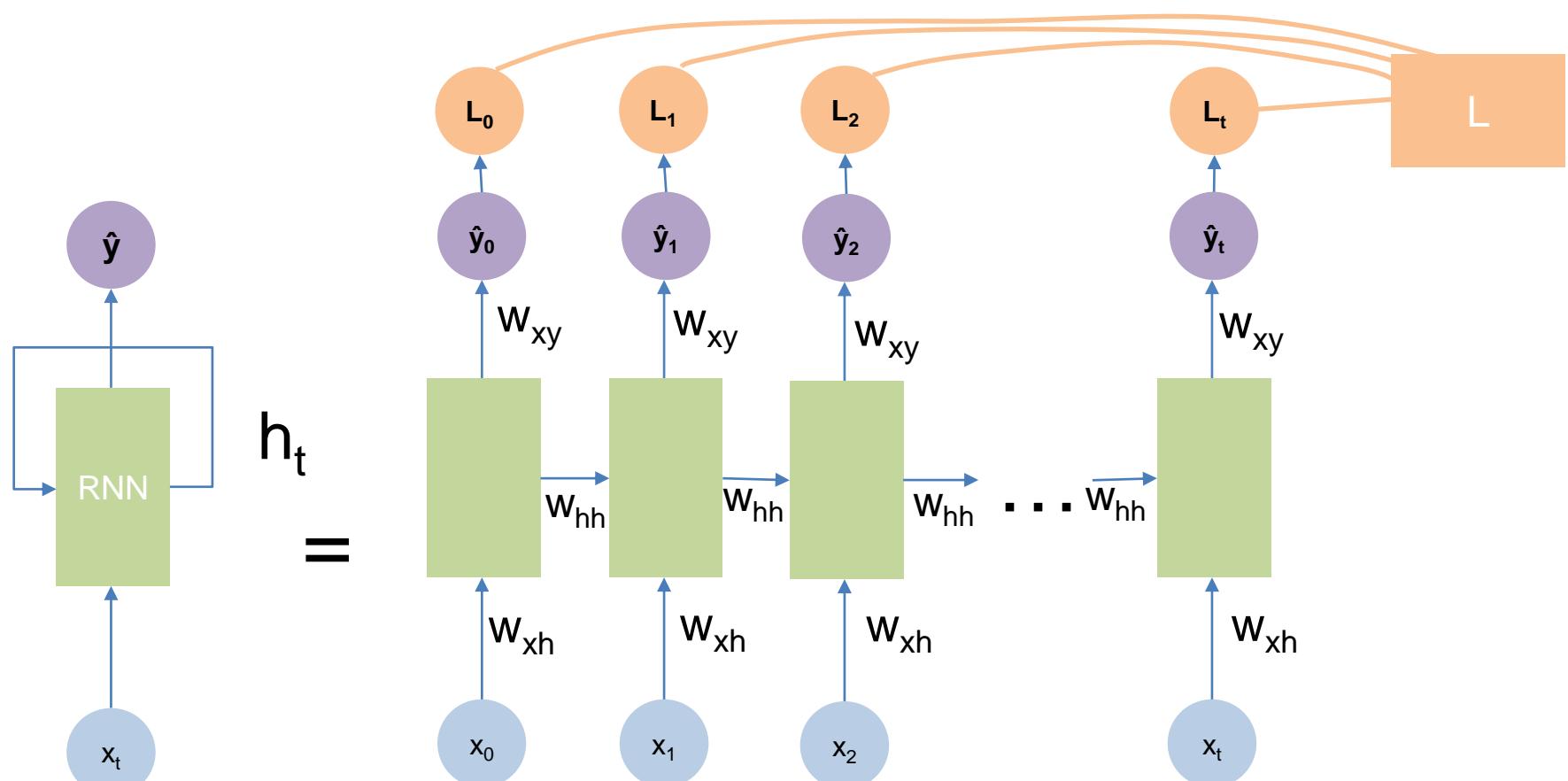
Using the same weight matrices through our network!



Recurrent neural network

The Loop can be seen as a chain like structure

Using the same weight matrices through our network!

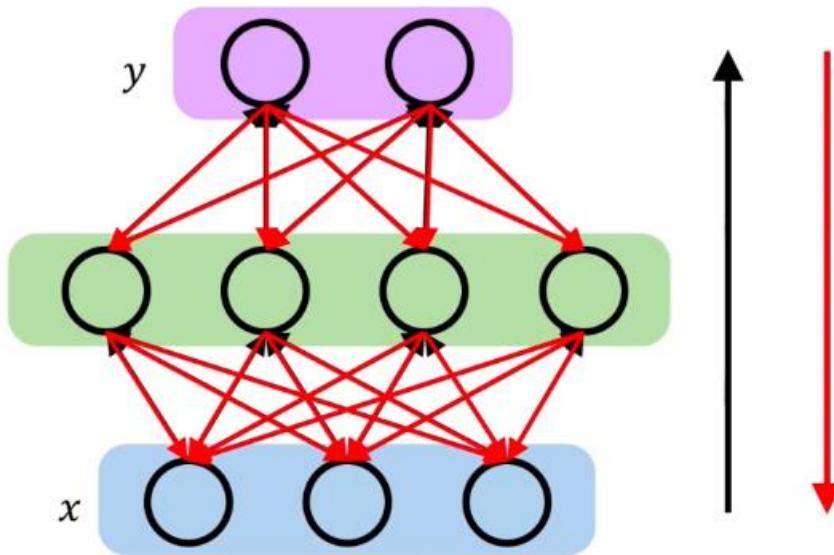


Recurrent neural
network

Individual contributions to the loss across all steps in time!

Back propagation through time

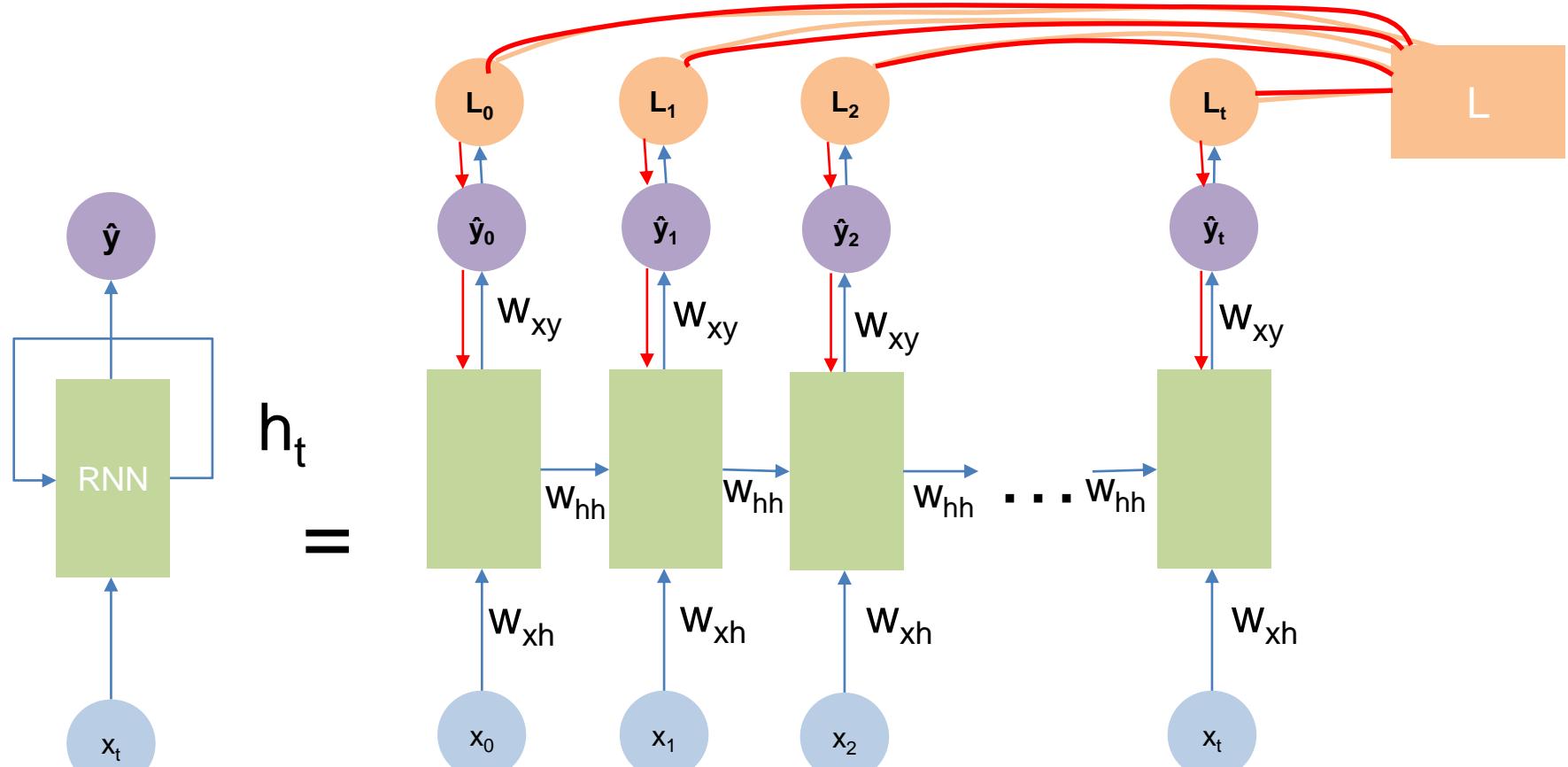
Recall the backpropagation in feed forward neural networks



Backpropagation algorithm:

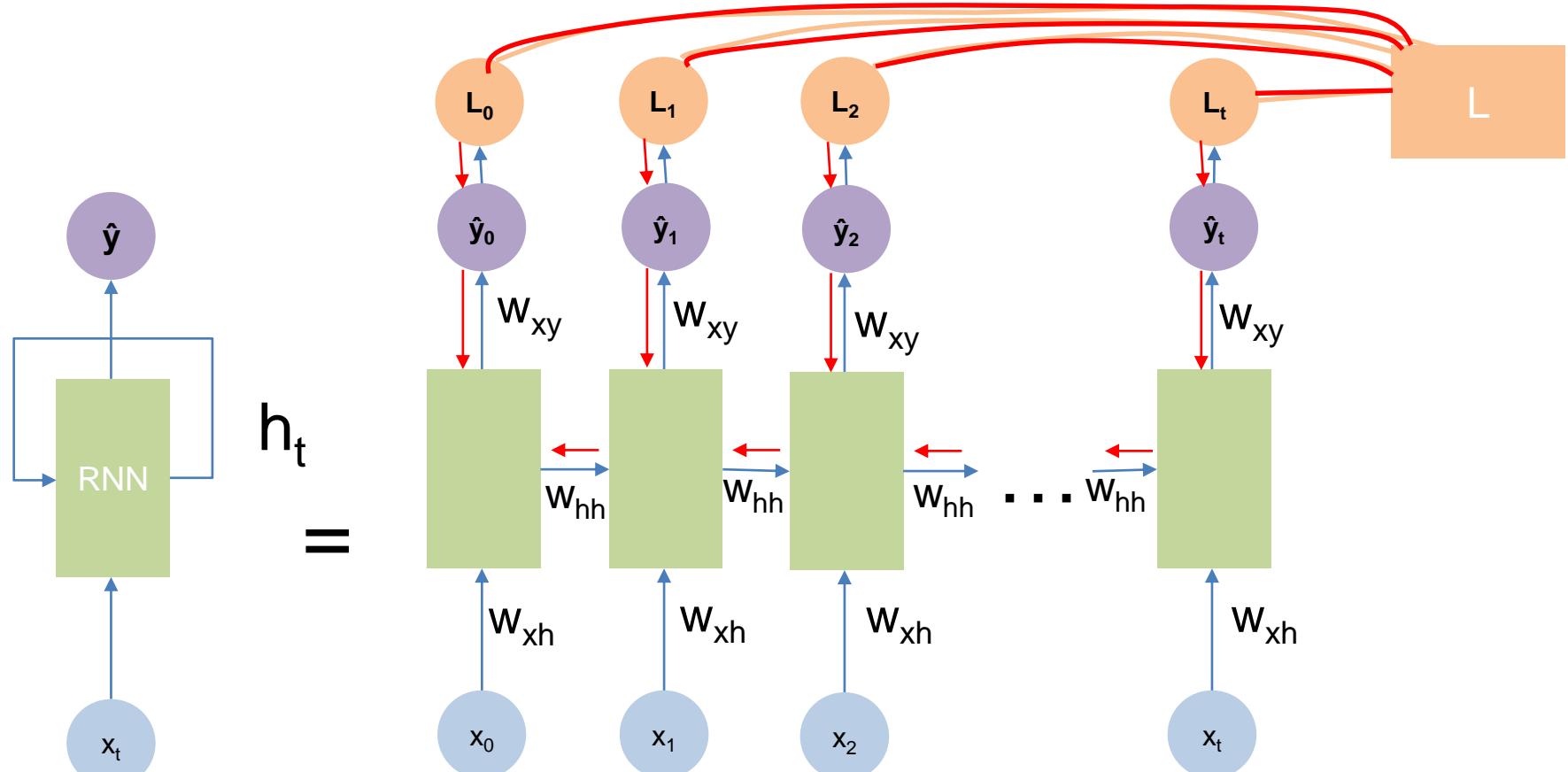
1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$



Recurrent neural
network

Individual contributions to the loss across all steps in time!

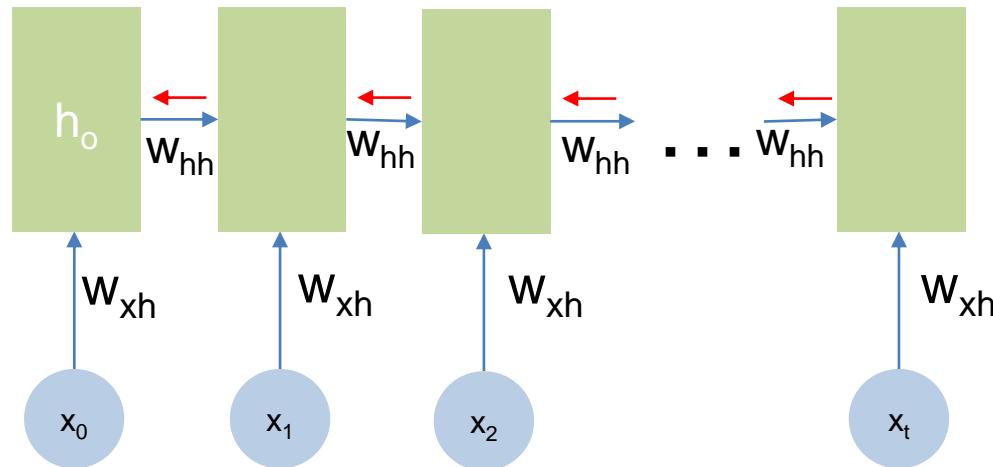


Recurrent neural
network

Individual contributions to the loss across all steps in time!

We have a weight matrix W_{hh} that is repeated many times.

Calculating gradient wrt to h_o involves many factors of W_{hh}



MATH96007 - MATH97019 - MATH97097

**Methods for Data Science
2019-20**

Prof Mauricio Barahona

THE PROCESS OF DATA SCIENCE

- ▶ The science: what's a good question and what's the dataset you will use to answer it?
- ▶ What is the input and what are you trying to determine
- ▶ Data handling, cleaning and exploration: computer programming and knowing things
- ▶ THE MATH and the main analysis:
 - ▶ statistics, computing: what do you compute, why, and what does it mean?
 - ▶ machine learning: supervised, unsupervised..
 - ▶ networks: characteristics, analysis, comparisons
- ▶ Communication: words, visualisations, summaries to tell the story

Aims and outcomes

- Learn mathematical concepts underpinning methods in learning from data
- The process of conceptualisation of the analysis
- The process of explanation of the analysis
- The mathematical justification of the analysis
- Getting a good exposure to current methods and their use in practice
- Challenges in dealing with (realistic) data

What the tools are, how they work mathematically, and how to analyse a dataset and clearly communicate the results

Data

WHY DO YOU NEED EXPLORATORY DATA ANALYSIS?

You will not have perfect data. EDA helps you to:

- ▶ find mistakes! (negative heights? copies of columns?
unreasonable values? missing values?)
- ▶ check your assumptions
- ▶ get an idea of whether the signal you want is actually in your data
 - ▶ if so, where?
 - ▶ maybe your problem is simple - just one predictor works
 - ▶ direction and size of relationships between predictors and outcome
- ▶ select your methods and tools
- ▶ find relationships between predictors

DATA AND TYPES OF EDA

Data: usually a big spreadsheet, table, data frame full of numbers and categories.

Typically one row per subject (patient, diamond, etc), and one column per variable (all the predictors and the outcome).

Reading 37 columns and 117,000 rows of a spreadsheet is not helpful.

We need to summarise and visualise the dataset in lots of ways to explore it, choose our method, find mistakes, etc.

CATEGORICAL DATA

- ▶ A *categorical* variable takes on one of a limited number of values (usually fixed)
- ▶ The distribution is really just the counts or frequencies of the different values
- ▶ Example: team name, which county someone lives in, breed of a dog
- ▶ Example: modules are mathematics, physics, computer science.
- ▶ Predicting a categorical variable is *classification*: we want to classify a new point into the right category.

QUANTITATIVE DATA

- ▶ A *quantitative variable* is numerical, and represents a measurable quantity
- ▶ Continuous examples: height, salary, stock price, profit, speed, blood count
- ▶ Ordinal examples (ordered): number of bedrooms, number of players, number of people in a city (nearly continuous for practical purposes)
- ▶ The distribution has a mean, variance, skewness etc.

Formalising the process

DATA: PREDICTOR VARIABLES

What might your *predictor* variables look like in practice?

- ▶ numbers
 - ▶ continuous measurements like height, price, profit, concentration
 - ▶ discrete measurements (integers): number of people, counts
- ▶ text: tweets, emails, patient records
- ▶ images: handwriting, medical images, photos
- ▶ even DNA sequences..
- ▶ combinations of these things

Randomness happens. It happens both in the *process* of getting the data, and in noise in the actual observations.

DATA: OUTCOME VARIABLES

What might your *outcome* (or “label”) variables look like?

- ▶ Continuous numbers. In this case the problem is *regression*
- ▶ Categorical (success/failure, what type of fruit, what is in the image): in this case the problem is *classification*

Learning from data

Inputs belong to an input space: $x_i \in X$. For example, if you have p different continuous features (height, weight, grade on exam, peak exhaled air flow, blood hemoglobin...) for each data point, then $x_i \in R^p$.

x_i could also contain some other data that are not real numbers: county of residence, gender, company name...

Outputs y can be continuous or categorical: some number or feature or group, some knowledge about the data point.

Fundamentally, machine learning is about finding ways to link the predictors x_i to the outcomes y_i so that we can make new predictions.

Supervised learning:

- ▶ There is a known *outcome variable*: the truth.
 - ▶ emails where we know which ones are spam
 - ▶ patients for whom we know their eventual outcome (heart attack or not)
 - ▶ patients where we know their glucose level
 - ▶ stocks and their prices 6 months later
 - ▶ tweets and their author (election team or Donald himself)
- ▶ We want to be able to predict that outcome for *new* observations of the input (predictor) variables

Unsupervised learning:

- ▶ There is not a known outcome variable that we want to predict
- ▶ Instead, we want to understand how the data are organised, clustered, related
 - ▶ use diverse measurements (blood, tumour shapes, gene expression) to identify similar types of breast cancers
 - ▶ explore differences in the bacterial communities in the guts of healthy individuals vs unhealthy
 - ▶ gain intuition for very high-dimensional data; make it more tractable

Ex: ARE THESE QUESTIONS FOR SUPERVISED OR UNSUPERVISED LEARNING?

1. I own a grocery chain and I use loyalty cards to track customer purchases. What are the major customer profiles? And what do they tend to buy together?
2. You have 6000 photos of dogs, cats, lizards and monkeys. Build a tool that can tell me which of these animals a newly-posted photo contains.

SUPERVISED LEARNING

The goal is to create a function $y = f(x)$ relating *predictor* variables x to *outcome* variables y .

The data, called a *training set*, is a set of N input-output pairs, or data points:

$$x_i, y_i, \quad i = 1,..N$$

The goal is to be able to predict y_{new} from a new observation x_{new} .

Somehow we should be able to take uncertainty into account.

CLASSIFICATION VS REGRESSION

Regression: predict a *quantitative* outcome (response) variable

- ▶ how much will increasing dosage affect the heart rate?
- ▶ how much does having an additional room increase the value of a house?
- ▶ does having a better shelf location impact sales of a product?

Classification: predict a *qualitative* outcome:

- ▶ Someone arrives at A&E with some symptoms. What medical condition do they have, of the ones that match the symptoms?
- ▶ Which emails are spam?
- ▶ Is a stock going to go up or down?

THE BASIC PROCESS FOR SUPERVISED LEARNING

1. Start with data (x_i, y_i) , $i = 1, \dots, N$. Each x_i usually a lot of features (components).
2. Decide: Classification or regression?
3. Define a *loss function* $L(y, f(x))$
4. Minimise the *mean sample loss*: $E(L) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$
5. Try to also have a reasonable *expected test loss*:
 $E(L(y_{new}, f(x_{new})))$

For example: in regression, the mean sample loss is the mean squared error:

$$L_{MSE}(y, f(x)) = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2$$

Now on the visualiser

Until now:

Supervised learning :

$$\vec{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)}) ; y^{(i)} \quad i=1, \dots, N$$

given \vec{x}^{in} $f(\vec{x}^{in}) = \hat{y}$
 ↑
inferred from $(\vec{x}^{(i)}, y^{(i)}) \quad i=1, \dots, N$

Switch now
to

Unsupervised learning :

- There is no 'observable', y
- There is no ground truth.
- There are no examples.
- There is no training.

We have a series of samples :

$$\{\vec{x}^{(i)}\}_{i=1}^N$$

Data

$$\vec{x}^{(i)} = (x_1^{(i)}, \dots, x_p^{(i)}) \quad i=1, \dots, n$$

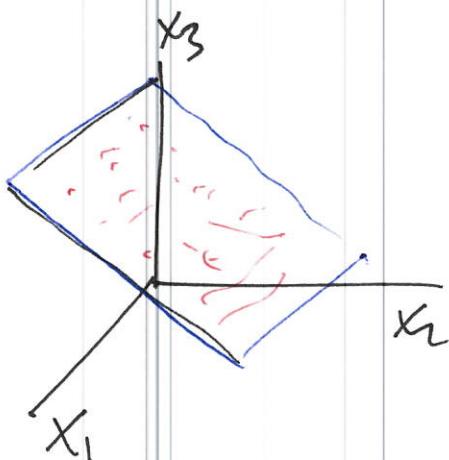
of different types

$$\begin{cases} \vec{x}^{(i)} \in \mathbb{R}^p \\ \vec{x}^{(i)} \in \{c_1, \dots, c_k\}^p \end{cases}$$

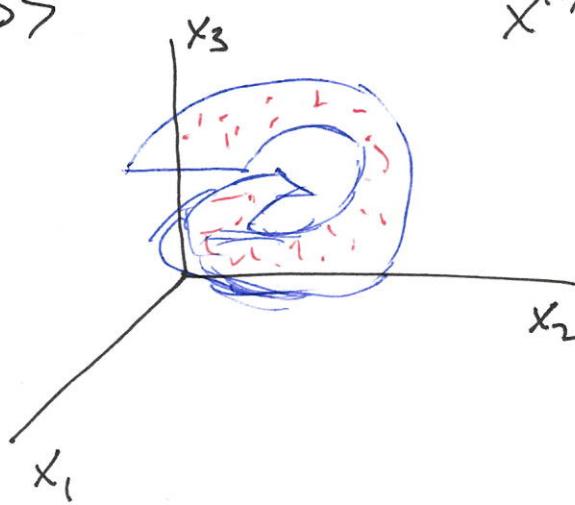
Types of tasks in unsupervised learning.

(1) Clustering : find groups of points that are more similar to each other within the group than to points outside the group.

(2) Dimensionality reduction:



$p >$



$$\vec{x}^{(i)} \in \mathbb{R}^3$$

Visualization .

Clustering :

Key ingredient : Similarity vs Dissimilarity
 "Distance".
 (not always a true metric)

$$D(\vec{x}^{(i)}, \vec{x}^{(j)})$$

Typical: A If $\vec{x} \in \mathbb{R}^P$

$$D(\vec{x}^{(i)}, \vec{x}^{(j)}) = \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$$

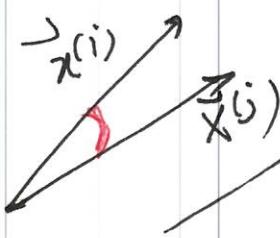
(i) dissimilarity:

$$= |\vec{x}^{(i)} - \vec{x}^{(j)}|$$

Distances.

(ii) similarities

cosine similarity:



$$S(\vec{x}^{(i)}, \vec{x}^{(j)}) = \frac{\vec{x}^{(i)} \cdot \vec{x}^{(j)}}{\|\vec{x}^{(i)}\| \|\vec{x}^{(j)}\|}$$

$$g(\vec{x}^{(i)}, \vec{x}^{(j)}) = \frac{\text{cov}(\vec{x}^{(i)}, \vec{x}^{(j)})}{\sqrt{\text{cov}(\vec{x}^{(i)}, \vec{x}^{(i)}) \cdot \text{cov}(\vec{x}^{(j)}, \vec{x}^{(j)})}}$$

statistical similarity:

B. If

\vec{x} are categorical :

* belongs to K classes

	C1	C2	C3
C1	0	D_{12}	D_{13}
C2		0	D_{23}
C3			0

$$K = 3$$

Distance matrix :
is based on
distance assigned
to feature classes.

C. Ordinal variables : ranked .

$$A \rightarrow B \rightarrow C \rightarrow D$$

$$k=4$$

$$i = 1 \quad 2 \quad 3 \quad 4$$

$$j = \frac{i - \frac{1}{2}}{K} \left[\frac{1}{8}, \frac{3}{8}, \frac{5}{8}, \frac{7}{8} \right]$$

Heuristics for clustering :

~~1. K-means~~

$$\left\{ \vec{x}^{(i)} \mid i=1^N \right\} \vec{x}^{(i)} \in \mathbb{R}^P$$
$$D(\vec{x}^{(i)}, \vec{x}^{(j)}) = \| \vec{x}^{(i)} - \vec{x}^{(j)} \|^2$$

Unsupervised learning:

(1) Clustering problem

General statement:

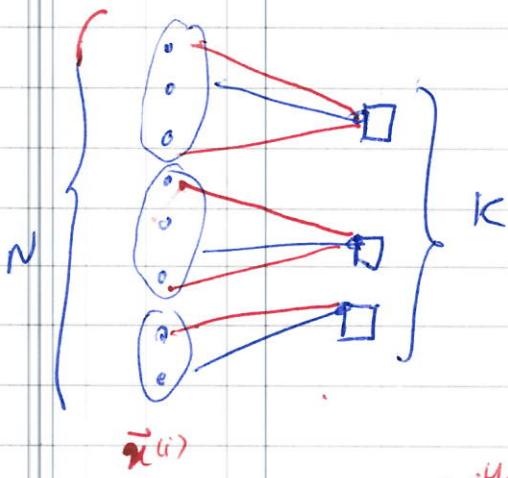
Given $\{\vec{x}^{(i)}\}_{i=1}^N$ our N samples

and a dissimilarity ("distance")

$$D(\vec{x}^{(i)}, \vec{x}^{(j)})$$

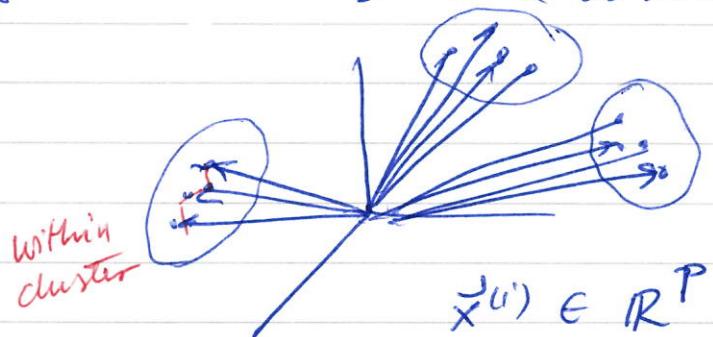
the objective is to find a partition of the N samples into K clusters such that the dissimilarity is small ~~across~~ within cluster ~~and~~ than across clusters.

In pictures: we have a mapping of the N samples to K clusters



Geometrically:

If the samples are continuous variables and D is a distance:

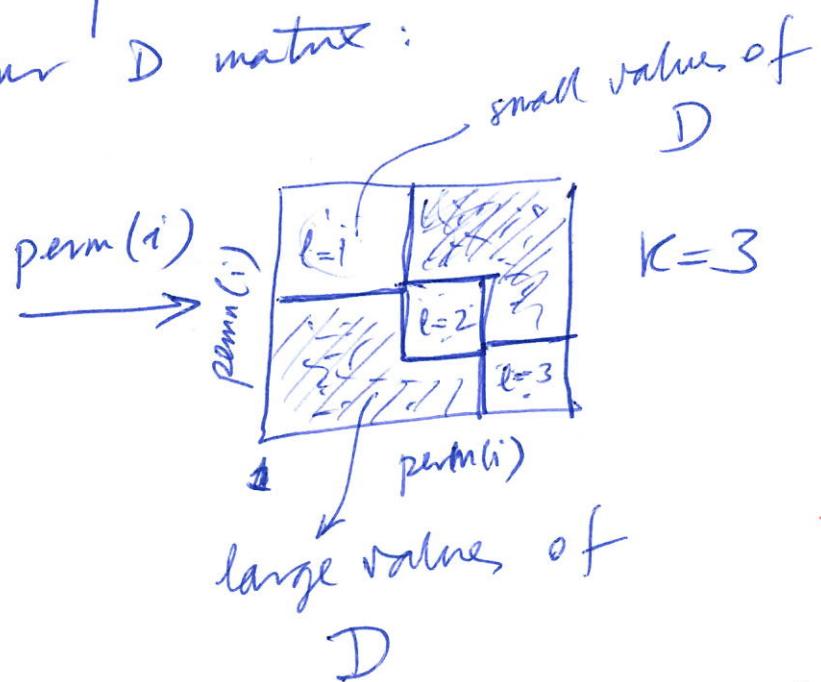


such that

$$\left. \begin{array}{l} \sum_{\text{within cluster}} D_{ij} \ll \\ \sum_{\text{across clusters}} D_{ij} \gg \end{array} \right\}$$

Another look at this problem:
 we have a $D_{N \times N}$ matrix and
 we want to reorder the rows and columns
 (i.e. find the permutation) that makes
 blocks in your D matrix:

$$\begin{matrix} & i=1 & \dots & N \\ i=1 & \vdots & \vdots & \vdots \\ N & \vdots & D_{ij} & \vdots \\ N & \vdots & \vdots & \vdots \end{matrix}$$



This will mean that for this clustering:

Within cluster dissimilarity

$$W(C) = \sum_{l=1}^K \sum_{i,j \in C_l} d(\vec{x}^{(i)}, \vec{x}^{(j)}) \frac{1}{2}$$

~~$\sum_{i,j \in C_l}$~~
is small

Total dissimilarity

Between cluster dissimilarity

$$B(C) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N d(\vec{x}^{(i)}, \vec{x}^{(j)}) - W(C)$$

is large

Finding the best H (or the best clustering) is

a combinatorial optimisation problem:

- There are usually NP-hard, all possible arrangements have to be tried to find the global optimum.
- But the number of clusterings to try ~~exploses~~ explodes:
For N samples, K clusters.

$$S(N, K) = \frac{1}{K!} \sum_{l=1}^K (-1)^{N-l} \binom{K}{l} l^N$$

$$S(10, 4) = 34,105$$

$$S(19, 4) = 10^{10}$$

⋮

Impractical to do this
enumeration.

Therefore, we need to come up with
heuristics to optimise.

K-means: What is the K-means heuristic?

We want to find an assignment between the N samples and \underline{k} clusters

First: Decide on K

Task: Find an assignment matrix:

$$H_{N \times K} = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & & & & & \\ 0 & \dots & \dots & 0 & \dots & 1 \\ c_1 & & & & & c_K \end{bmatrix}_{i=1}^{i=N}$$

This summarizes a hard assignment that is exhaustive.

$$\{C_l\}_{l=1}^K \quad (i) \quad C_l \cap C_{l'} = \emptyset$$

$l \neq l'$

$$(ii) \quad \bigcup_{l=1}^K C_l = \{\vec{x}^{(i)}\}_{i=1}^N$$

As stated above,
This is a combinatorial problem: check all
the permutations to see which one is best

K-means algorithm:

Given $\{\vec{x}^{(i)}\} \in \mathbb{R}^P$

Compute $D(\vec{x}^{(i)}, \vec{x}^{(j)}) = \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$

Distance matrix

For a given clustering: $\{C_l\}_{l=1}^K = C$

~~can we calculate~~
~~within cluster~~
~~normalised distance~~

$$W(C) = \frac{1}{2} \sum_{l=1}^K \frac{1}{|C_l|} \sum_{i,j \in C_l} \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$$

~~equivalent to finding $H_{N \times K}$~~

where
| C_l | is the cardinality of C_l

$W(C)$ can be rewritten in terms of H :

$$\text{Tr} \left[\frac{1}{2} (H^T H)^{-1} \left[\begin{array}{c|c} H^T & D_{N \times N} \\ \hline K \times N & N \times K \end{array} \right] \right] = W(C)$$

$$\text{diag}(C_l) = H^T H = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} \square & \square & \square \\ \square & \square & \square \\ \square & \square & \square \end{bmatrix} \quad k=3$$

$$|C_3| \quad \sum_{i,j \in C_3} \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$$

$$T = \frac{1}{2} \vec{1}^T D \vec{1} = \frac{1}{2} \vec{1}^T [H^T D H] \vec{1}$$

$$\vec{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}_{N \times 1}$$

$$\cancel{\vec{H} \vec{1}_{K \times 1} = \vec{1}_{N \times 1}}$$

Objective : Minimise $W(C)$

$$\min_{H} \text{Tr} \left[\frac{1}{2} (H^T H)^{-1} (H^T D H) \right]$$

$D_{N \times N} (\vec{x}^{(i)})$ is the distance matrix

Maximise $T - W(C) = \text{between cluster distance}$.

Algorithm :

- Step 0 : Initialisation Assign at random ~~the~~ every sample to a cluster.

- Step 1 : Compute the centroid of each of the K clusters

$$\vec{m}_l = \frac{1}{|C_l|} \sum_{i \in C_l} \vec{x}^{(i)}$$

- Step 2 : Reassign each $\vec{x}^{(i)}$ to the closest centroid.

Consider
~~Find~~
 \hat{x}_i ,
 $\hat{l}_i^{(t)}$

Evaluate

$$\hat{l}_i^{(t+1)} = \arg \min_l \|\tilde{x}^{(i)} - \tilde{m}\|^2$$

Iterate step 1 and 2
Until convergence:

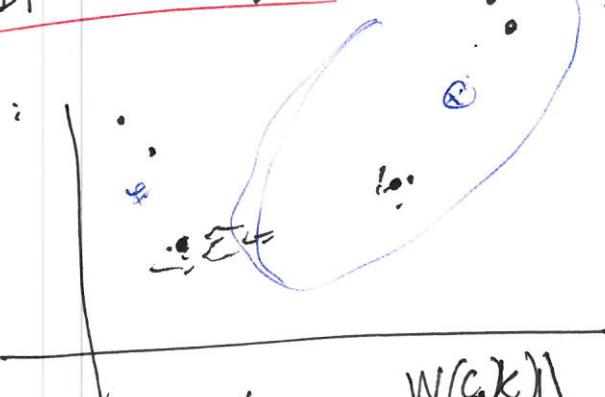
i.e., $W(C)$ does not improve much.
or assignments are not changing.

Heuristic: Gradient method

because at every step $W(C)$
is decreased.

If converges to a local optimum

(1) Outliers:

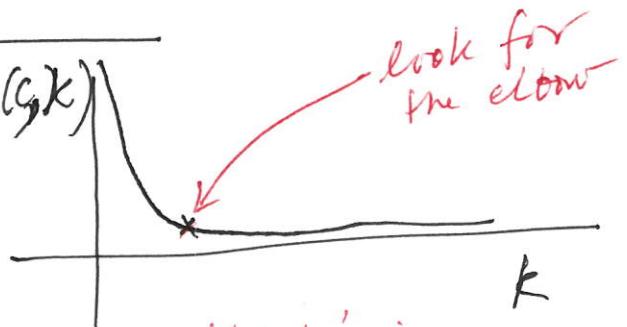


Instead:

k -medoids.

(2) Try different k 's:

How to choose k



or the 'kink' in

$$W(C_{K+1}^*, K+1) - W(C_K^*, K)$$

Comments:

K-means optimization

The \checkmark step makes sense:

(Step 1+2) Given clustering $C = \{C_l\}_{l=1}^K$

$$W = \frac{1}{2} \sum_{l=1}^K \frac{1}{|C_l|} \sum_{i:j \in C_l} \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$$

with centroids $\vec{m}_l = \frac{1}{|C_l|} \sum_{i \in C_l} \vec{x}^{(i)}$ $l = 1, \dots, K$

$$W = \frac{1}{2} \sum_{l=1}^K \frac{1}{|C_l|} \sum_{i:j \in C_l} \|(\vec{x}^{(i)} - \vec{m}_l) - (\vec{x}^{(j)} - \vec{m}_l)\|^2$$

$$= \sum_{l=1}^K \frac{1}{2} \frac{1}{|C_l|} \sum_{i:j \in C_l} \left[\|\vec{x}^{(i)} - \vec{m}_l\|^2 + \|\vec{x}^{(j)} - \vec{m}_l\|^2 - 2 (\vec{x}^{(i)} - \vec{m}_l) \cdot (\vec{x}^{(j)} - \vec{m}_l) \right]$$

$$= \sum_{l=1}^K \sum_{i \in C_l} \|\vec{x}^{(i)} - \vec{m}_l\|^2 - (\vec{x}^{(i)} - \vec{m}_l) \underbrace{\frac{1}{|C_l|} \sum_{j \in C_l} (\vec{x}^{(j)} - \vec{m}_l)}_0$$

In summary:

$$W = \frac{1}{2} \sum_{l=1}^K \frac{1}{|C_l|} \sum_{i:j \in C_l} \|\vec{x}^{(i)} - \vec{x}^{(j)}\|^2$$

$$= \sum_{l=1}^K \sum_{i \in C_l} \|\vec{x}^{(i)} - \vec{m}_l\|^2$$

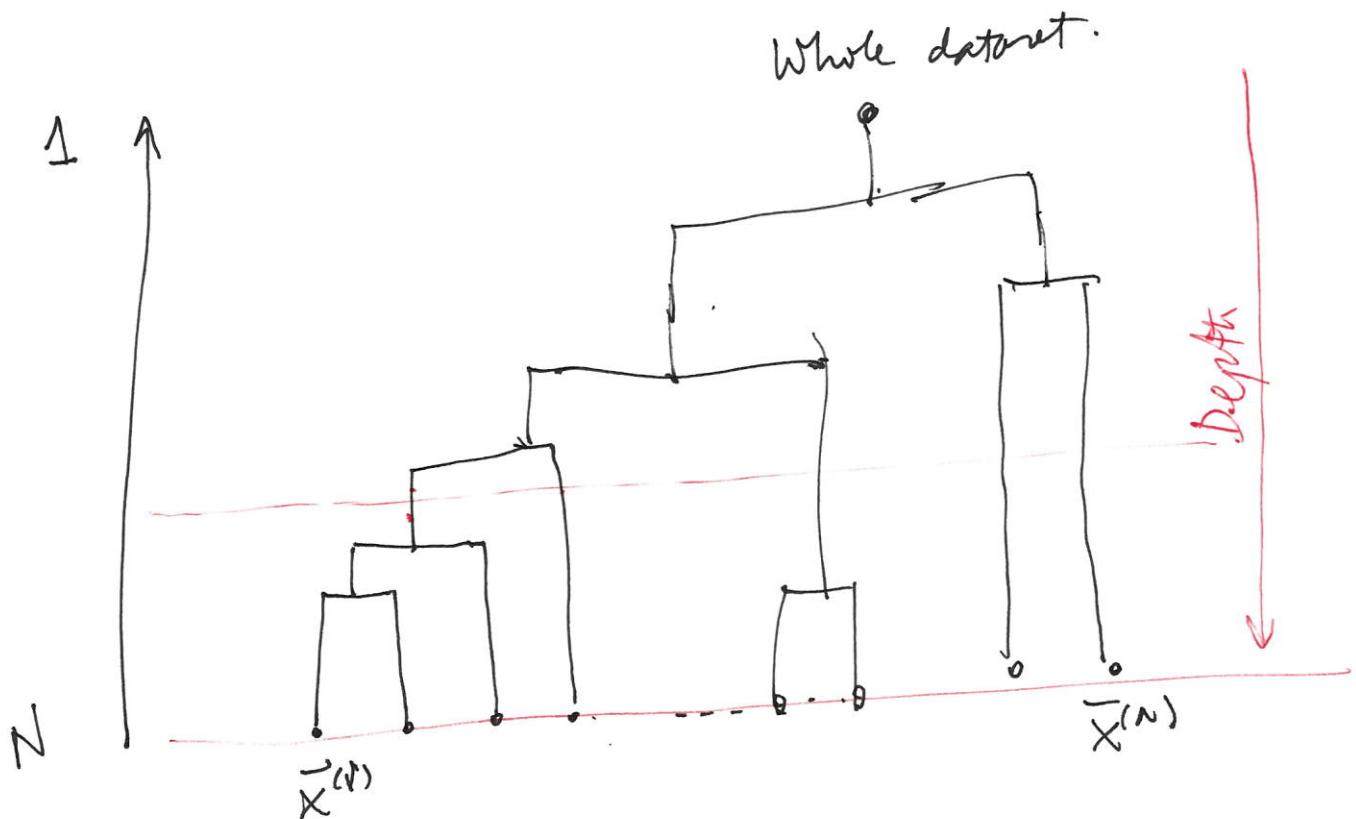
So trying to minimize the distance to the centroids is our objective.

(2) Hierarchical clustering

$$\{\vec{x}^{(i)}\}_{i=1, \dots, N}$$

$$D_{N \times N} = (D_{ij}) \quad D_{ij} = d(\vec{x}^{(i)}, \vec{x}^{(j)})$$

Represents the whole data set as a binary tree ('dendrogram') where leaves are samples and the root is the whole dataset.



- Hierarchical structure imposed on the data because of the requirement of binary splits.
- Monotonicity relating height and dissimilarity within cluster.

Agglomerative schemes:

Reduce the number of clusters 1 by 1 from N to 1.

(1) Simple linkage (SL) :

Two clusters : $d_{SL}(G, H) = \min_{\substack{i \in G \\ j \in H}} D_{ij}$

Nearest neighbour

(2) Complete linkage (CL) :

Farthest neighbour : $d_C(G, H) = \max_{\substack{i \in G \\ j \in H}} D_{ij}$

(3) Group average:

$$d_{GA}(G, H) = \frac{1}{N_G N_H} \sum_{\substack{i \in G \\ j \in H}} d_{ij}$$

Disadv:

Recursive k-means with $k=2$ gives a binary tree also but:

(1) monotonicity is not preserved

(2) it depends on the initialisations at every split.

So it can be quite variable.

Better schemes exist but, in general, agglomerative algorithms are used more extensively.

Spectral methods

Principal component analysis (PCA)

$$\vec{x}^{(i)} \quad i=1, \dots, N \quad \vec{x}^{(i)} \in \mathbb{R}^P$$

Find a description of the data set that captures as much information as possible in reduced dimensions.

Original dimension: $P \longrightarrow$ Reduced dimension $M < P$

$$\vec{x}^{(i)} = \sum_{j=1}^P a_j^{(i)} \vec{\phi}_j$$

Search for an orthonormal basis: $\{\vec{\phi}_j\} \quad \vec{\phi}_j \in \mathbb{R}^P$

$$\vec{\phi}_j^T \cdot \vec{\phi}_k = \delta_{jk} \quad \begin{cases} 1, & \text{if } j=k \\ 0, & \text{otherwise} \end{cases}$$

$$\Rightarrow a_j^{(i)} = \vec{\phi}_j^T \cdot \vec{x}^{(i)} = \vec{x}^{(i)T} \cdot \vec{\phi}_j$$

Minimisation of \mathcal{L} under constraints:

$$\mathcal{L} = \sum_{j=M+1}^P \vec{\phi}_j^\top C_x \vec{\phi}_j + \sum_{j=M+1}^P \gamma_j (1 - \vec{\phi}_j^\top \vec{\phi}_j)$$

$$\frac{\nabla \mathcal{L}}{\vec{\phi}_j} = \frac{\partial \mathcal{L}}{\partial \vec{\phi}_j} = 2 C_x \vec{\phi}_j - 2 \gamma_j \vec{\phi}_j$$

$$\left. \frac{\partial \mathcal{L}}{\partial \vec{\phi}_j} \right|_{\vec{\phi}_j^*} = 0 \quad C_x \vec{\phi}_j^* = \gamma_j \vec{\phi}_j^*$$

Solution: $\{b_j^*, \vec{\phi}_j^*\}$
↑
eigenvectors of C_x

$$\text{MSE} = \sum_{j=M+1}^P \underbrace{\vec{\phi}_j^{*\top} C_x \vec{\phi}_j^*}_{\vec{\phi}_j^* F} = \sum_{j=M+1}^P \vec{\phi}_j^{*\top} \gamma_j \vec{\phi}_j^* \\ = \sum_{j=M+1}^P \gamma_j$$

Sum of the eigenvalues that have been discarded.

$$MSE = \sum_{j=M+1}^P \frac{1}{N} \sum_{i=1}^N \left(\vec{x}^{(i)T} \cdot \vec{\phi}_j - \underbrace{\frac{1}{N} \sum_{i=1}^N \vec{x}^{(i)T} \cdot \vec{\phi}_j}_{b_j} \right)^2$$

$$= \sum_{j=M+1}^P \frac{1}{N} \sum_{i=1}^N \left[\left(\vec{x}^{(i)T} - \frac{1}{N} \sum_{i=1}^N \vec{x}^{(i)T} \right) \cdot \vec{\phi}_j \right]^2$$

$$= \sum_{j=M+1}^P \frac{1}{N} \left[\vec{\phi}_j^T \cdot \left(\vec{x}^{(i)} - \frac{1}{N} \sum_{i=1}^N \vec{x}^{(i)} \right) \cdot \left(\vec{x}^{(i)} - \frac{1}{N} \sum_{i=1}^N \vec{x}^{(i)} \right)^T \vec{\phi}_j \right]$$

$$= \underset{i}{\mathbb{E}} \left[(\vec{x} - \mathbb{E}(\vec{x})) \cdot (\vec{x} - \mathbb{E}(\vec{x}))^T \right]$$

Covariance matrix

$$(C_x)_{p \times p}$$

$$MSE = \sum_{j=M+1}^P \vec{\phi}_j^T C_x \vec{\phi}_j$$

(1xP) (PxP) (Px1)

Data

Approximate $\tilde{x}^{(i)}$ by:

$$\tilde{x}_M^{(i)} = \sum_{j=1}^M a_j^{(i)} \vec{\phi}_j + \sum_{j=M+1}^P b_j \vec{\phi}_j$$

Find $\{b_j, \vec{\phi}_j\}$ ^{that} apply to all samples.

For each sample:

$$\Delta \tilde{x}^{(i)} = \tilde{x}^{(i)} - \tilde{x}_M^{(i)} = \sum_{j=M+1}^P [a_j^{(i)} - b_j] \vec{\phi}_j$$

$i = 1, \dots, N$

Error on
the dataset:

$$MSE = \frac{1}{N} \sum_{i=1}^N \|\Delta \tilde{x}^{(i)}\|^2 = \frac{1}{N} \sum_{i=1}^N \left[\sum_{j=M+1}^P (a_j^{(i)} - b_j) (\vec{\phi}_j^\top \cdot \vec{\phi}_K) \right]$$

$$= \frac{1}{N} \sum_{i=1}^N \sum_{j=M+1}^P (a_j^{(i)} - b_j)^2 \quad (\text{by orthogonality})$$

Minimize MSE:

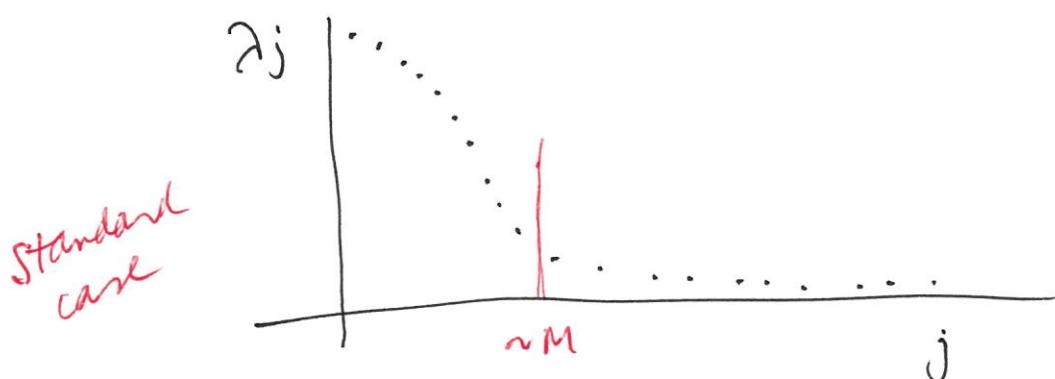
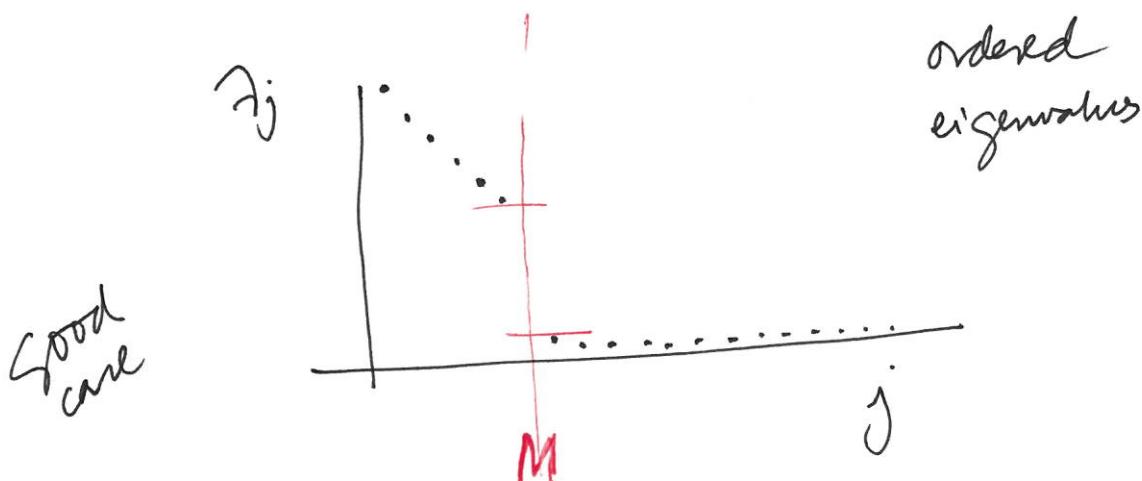
$$(1) \quad \frac{\partial MSE}{\partial b_j} = \frac{1}{N} \sum_{i=1}^N (-2)(a_j^{(i)} - b_j)$$

$$\left. \frac{\partial MSE}{\partial b_j} \right|_{b_j^*} = 0$$

$$b_j^* = \frac{1}{N} \sum_{i=1}^N a_j^{(i)}$$

Find eigendecomposition

Obtain C_x from data:



Tolerance: ϵ

$$M(\epsilon) \mid \sum_{j=M+1}^P \lambda_j < \epsilon$$

Principal components are the eigenvectors
of C_x

$$\hat{\vec{x}}^{(i)} = \sum_{j=1}^M (\vec{x}^{(i)\top} \cdot \vec{\phi}_j) \vec{\phi}_j$$

and the $\vec{\phi}_j$ are the
eigenvectors of the
covariance matrix

Mathematical basis for PCA :

Singular value decomposition (SVD)

Eckart & Young 39.

If B has rank M

$$\text{then } \|A - B\| \geq \|A - A_m\|$$

$$A_m = \sum_{j=1}^M \sigma_j \vec{u}_j \vec{v}_j^T$$

where \vec{u}_j are the right eigenvectors of A
 \vec{v}_j are the left eigenvectors of A
 σ_j are the singular values of A

These definitions
follow from
the SVD

SVD

Analogue to
Diagonalization
of
square
matrices

$$AV = V\Lambda$$

$A_{n \times n}$

$$V = (\vec{v}_1 \dots \vec{v}_n)$$

$$\Lambda = \text{diag}(\sigma_i)$$

$$A = V\Lambda V^{-1}$$

but for rectangular matrices

Analogue for rectangular matrices:

Given $A_{m \times n} \in \mathbb{R}^{m \times n}$

$$A_{m \times n} V_{n \times n} = U_{m \times m} \Sigma_{m \times n}$$

$$V_{n \times n} = (\vec{v}_1 \dots \vec{v}_n)$$

$$U_{m \times m} = (\vec{u}_1 \dots \vec{u}_m)$$

$$\Sigma_{m \times n} = \left(\begin{array}{cc|c} \sigma_1 & 0 & 0 \\ 0 & \ddots & \vdots \\ 0 & 0 & \sigma_r \end{array} \right) \quad \text{with } r = \min(m, n)$$

$$\min(m, n) \geq r$$

Mathematical basis for PCA:

Singular value decomposition (SVD)

Eckart & Young (39.)

If B has rank M

$$\text{then } \|A - B\| \geq \|A - A_m\|$$

$$A_m = \sum_{j=1}^M \sigma_j \vec{u}_j \vec{v}_j^T$$

where \vec{u}_j are the right eigenvectors of A
 \vec{v}_j are the left eigenvectors of A
 σ_j are the singular values of A

These definitions
follow from
the SVD

SVD

Analogue to
Diagonalization
of
square
matrices

$$AV = V\Lambda$$

$A_{n \times n}$

$$V = (\vec{v}_1 \dots \vec{v}_n)$$

$$\Lambda = \text{diag}(\sigma_i)$$

$$A = V\Lambda V^{-1}$$

but for rectangular matrices.

SVD

Analogue for rectangular matrices:

Given $A_{m \times n} \in \mathbb{R}^{m \times n}$

$$A_{m \times n} V_{n \times n} = U_{m \times m} \Sigma_{m \times n}$$

$$V_{n \times n} = (\vec{v}_1 \dots \vec{v}_n)$$

$$U_{m \times m} = (\vec{u}_1 \dots \vec{u}_m)$$

$$\Sigma_{m \times n} = \begin{pmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_r \\ \hline & & 0 \end{pmatrix}$$

Singular values.

$$\min(m, n) \geq r$$

r is the rank of A (at most $\min(m, n)$)
but could be smaller.

Reduced form of the SVD:

$$A_{m \times n} U_{m \times m} \Sigma_{m \times n} V_{n \times n}^T = \sum_{j=1}^r \sigma_j \vec{u}_j \vec{v}_j^T = (U_r)_{m \times r} \Sigma_r (V_r^T)_{r \times n}$$

Truncations of U and V taking the first r vectors \vec{u}_j and \vec{v}_j .

Definition of SVD

What are these U and V ?

$$\boxed{AV = U\Sigma \quad VV^T = I \quad UU^T = I}$$
$$A = U\Sigma V^T$$

$$(A^T A)_{n \times n} = V \Sigma^T \underbrace{U^T}_U \Sigma V^T = V (\Sigma \Sigma^T) V^T$$

$$(AA^T)_{m \times m} = U \Sigma \underbrace{V^T}_V \Sigma^T U^T = U (\Sigma \Sigma^T) U^T$$

$\left\{ \begin{array}{l} V_{n \times n} \text{ contains the eigenvectors of } (A^T A) \\ U_{m \times m} \text{ contains the eigenvectors of } (AA^T) \\ \Sigma^T \text{ and } \underline{\Sigma \Sigma^T} \text{ contain the non-zero eigenvalues of } A^T A \text{ and } AA^T \end{array} \right.$

Singular values

$\sigma_i = \sqrt{\text{eigenvectors of } (A^T A) \text{ and } (AA^T)}$

Left and right singular vectors come in pairs with the same σ_k^2

$$\left\{ \begin{array}{l} (A^T A) \vec{v}_k = \sigma_k^2 \vec{v}_k \\ (AA^T) \vec{u}_k = \sigma_k^2 \vec{u}_k \end{array} \right.$$

with the same σ_k^2

It follows from the equality that:

$$\vec{u}_k = \frac{A \vec{v}_k}{\sigma_k}$$

Check: $(AA^T)\vec{u}_k = A A^T \frac{A \vec{v}_k}{\sigma_k} = \sigma_k^2 \frac{A \vec{v}_k}{\sigma_k} = \sigma_k^2 \vec{u}_k$ ✓

Eckart - Young: of rank at most k

"Given $A_{m \times n}$ find $B_{m \times n}$ that is close to A "
i.e., $\|A - B\|$ small.
 \uparrow Matrix norm.

Matrix Norms:

- Induced norms: ℓ_2 norm = Spectral norm

$$\|A\|_2 = \max_{\vec{x}} \frac{\|A\vec{x}\|}{\|\vec{x}\|} = \sigma_1$$

largest singular value of A

- Element-wise norms:

e.g. Frobenius norm:

$$\|A\|_F^2 = \sum_{ij} |a_{ij}|^2 = \text{Tr}(A^T A) =$$

$$= \text{Tr}(U \Sigma^2 U^T) = \text{Tr}(\Sigma^2) = \sum_{j=1}^r \sigma_j^2$$

Equivalent to
the vector norm of

$$\|\text{vec}(A)\|$$

These are vector norms

(1) $E-Y$ for spectral norm:

Statement :

$$\text{If } \text{rank}(B) \leq k \text{ then } \|A - B\| = \max_{\vec{x}} \frac{\|(A - B)\vec{x}\|}{\|\vec{x}\|}$$

$$\geq \frac{\|(A - A_k)\vec{x}\|}{\|\vec{x}\|}$$

$$\text{where } A_k = U_k \Sigma_k V_k$$

Proof :

$$\textcircled{a} \| (A - A_k) \vec{x} \| = \sigma_{k+1} \quad \text{since } A_k = U_k \Sigma_k V_k$$

where these are from above

$$\exists \vec{x} \neq 0_v / B\vec{x} = 0_v \text{ and } \vec{x} = \sum_{j=1}^{k+1} c_j v_j$$

given that

$$\begin{cases} \dim [\ker B] \geq n-k \\ \Rightarrow \{v_j\}_{j=1}^{k+1} \cap \ker B \neq \emptyset \end{cases}$$

Then we have : $\| (A - B) \vec{x} \|^2 = \| A \vec{x} \|^2 = \sum_{j=1}^{k+1} c_j^2 \sigma_j^2$

$$= \left(\sum_{j=1}^{k+1} c_j^2 \right) \sigma_{k+1}^2 = \|\vec{x}\|^2 \sigma_{k+1}^2$$

~~**~~ $\| (A - B) \vec{x} \|^2 \geq \|\vec{x}\|^2 \sigma_{k+1}^2$ ✓

where the $\{\sigma_1, \sigma_2, \dots, \sigma_k, \sigma_{k+1}, \dots, \sigma_r\}$ are ordered in decreasing order.

Hence : From ~~a~~ and ~~**~~ $\left[\frac{\| (A - B) \vec{x} \|^2}{\|\vec{x}\|^2} \geq \sigma_{k+1}^2 = \| A - A_k \|^2 \right]$ ✓

Computative math:

(2) $E - Y$ for Frobenius norm:

Find B that minimizes

$$\min_B \|\mathbf{A} - \mathbf{B}\|_F^2$$

① Any B of rank k or less can be written as:

$$\mathbf{B} = \mathbf{C} \mathbf{R}$$

$m \times k$ $k \times n$

where \mathbf{B} of rank k or less

where

$$\mathbf{C}^T \mathbf{C} = \mathbf{D}$$

$$\mathbf{R} \mathbf{R}^T = \mathbf{I}$$

This is from SVD
(for example)

Consider E and minimize over C, R :

$$E = \|\mathbf{A} - \mathbf{C} \mathbf{R}\|_F^2$$

Matrix calculus

$$\frac{\partial E}{\partial \mathbf{C}} = 2(\mathbf{C} \mathbf{R} - \mathbf{A}) \mathbf{R}^T \quad ①$$

$$\frac{\partial E}{\partial \mathbf{R}} = 2(\mathbf{R}^T \mathbf{C}^T - \mathbf{A}^T) \mathbf{C} \quad ②$$

$$\cancel{(\mathbf{A}^T \mathbf{A}) \mathbf{R}^T} \quad \mathbf{C} \mathbf{R} \mathbf{R}^T = \mathbf{A} \mathbf{R}^T \quad ①$$

$$\text{At the minimum} \Rightarrow \mathbf{R}^T \mathbf{C}^T \mathbf{C} = \mathbf{A}^T \mathbf{C} \quad ②$$

SVD
gives best
 $\mathbf{C} \mathbf{R}$

$$(\mathbf{A}^T \mathbf{A}) \mathbf{R}^T = \mathbf{R}^T \mathbf{D}$$

$$\mathbf{R}^T = \mathbf{V}_K$$

$$(\mathbf{A} \mathbf{A}^T) \mathbf{C} = \mathbf{C} \mathbf{D}$$

$$\mathbf{C} = \mathbf{U}_K.$$

Combining
① & ②
we get

And the error is then:

$$E = \|\mathbf{A} - \mathbf{C} \mathbf{R}\|_F^2 = \sum_{j=k+1}^r \sigma_j^2$$

Connection with PCA

$$y^T_{N \times P} = \begin{pmatrix} x_1^{(1)} & \dots & x_P^{(1)} \\ \vdots & & \vdots \\ x_1^{(N)} & \dots & x_P^{(N)} \end{pmatrix} = \begin{pmatrix} \bar{y}_1^T \\ \vdots \\ \bar{y}_N^T \end{pmatrix}$$

$$\bar{y}_i \in \mathbb{R}^P \quad i = 1, \dots, N$$

$$\vec{1}_{N \times 1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \text{ vector of ones.}$$

Let: $\vec{1}^T_{1 \times N} y^T_{N \times P} = \left(\sum_{i=1}^N x_1^{(i)} \dots \sum_{i=1}^N x_P^{(i)} \right)_{1 \times P}$

$$\frac{1}{N} \vec{1}^T (\vec{1}^T y^T) = \begin{pmatrix} \langle x_1 \rangle_i & \dots & \langle x_p \rangle_i \\ \vdots & & \vdots \\ \langle x_1 \rangle_i & \dots & \langle x_p \rangle_i \end{pmatrix}_{N \times P}$$

Then define:

$$\tilde{y}^T_{N \times P} = y^T - \frac{1}{N} \vec{1} \vec{1}^T y^T = \underbrace{\left(I - \frac{1}{N} \vec{1} \vec{1}^T \right)}_{\text{Centering matrix}} y^T$$

It then follows
that

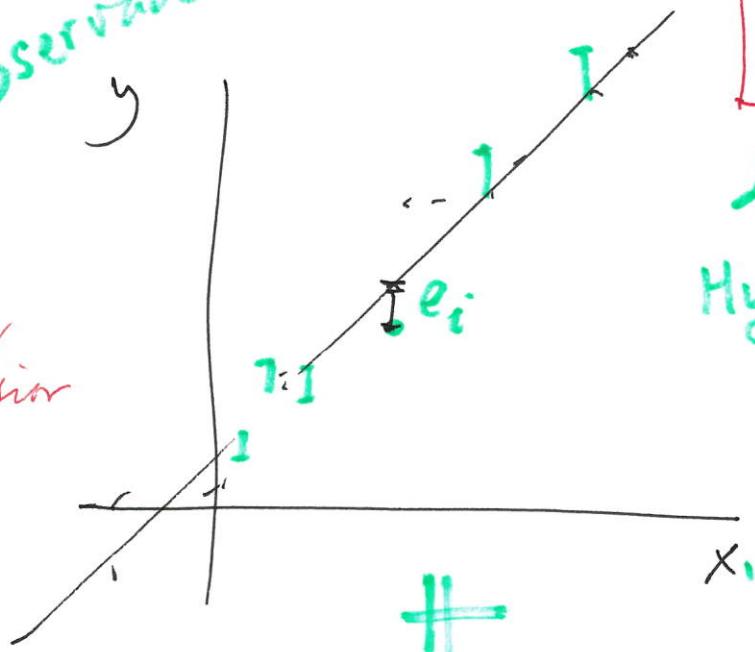
$$(\tilde{y} \tilde{y}^T)_{P \times P} = C_x \text{ sample covariance matrix}$$

Using the SVD of \tilde{y} which is
the centered data matrix.

So we conclude that PCA just considers the SVD of
the centered data matrix \tilde{y} , which is equivalent
to obtaining the eigenvectors of the covariance matrix
 $C_x = (\tilde{y} \tilde{y}^T)$

Observable:

(A)
Linear regression

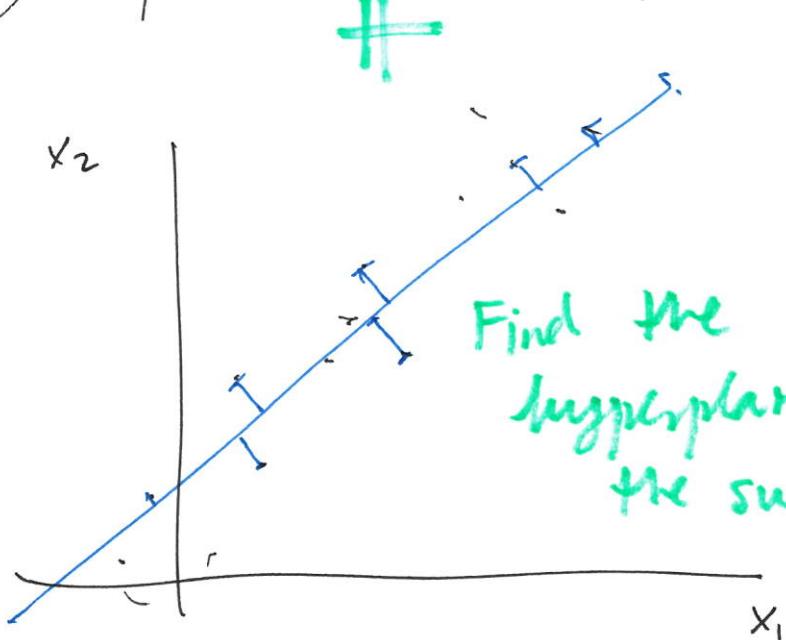


Geometric picture of
PCA/SVD

Linear regression :

Hyperplane that minimize
the sum of squared
error

(B)
PCA



Find the

hyperplane such that
the sum of distances
that are normal
to the plane
are minimized.

The hyperplanes we obtain are not the same for linear regression and PCA.

PCA through SVD is close to total least squares or proper orthogonal decompositions.

$N \gg p$

Rank $r = p$

Remember:

SVD and PCA

$$X_{N \times p} = Y_{N \times p}^T = \begin{pmatrix} \vec{y}_1^T \\ \vdots \\ \vec{y}_N^T \end{pmatrix}$$

$$\vec{y}_i \in \mathbb{R}^p$$

$$\tilde{Y}_{N \times p}^T = \left(I - \frac{1}{N} \mathbf{1}\mathbf{1}^T \right) Y^T$$

centered
data
matrix

$$\sum_{j=1}^p \sigma_j \vec{u}_j \vec{v}_j^T = \tilde{Y}_{N \times p}^T = U_{N \times p} \sum_{p \times p} V_{p \times p}^T$$

SVD

SVD provides
a new basis
 $\{\vec{V}_j\}_{j=1}^p$
orthonormal

$$U = (\vec{u}_1, \dots, \vec{u}_p)_{N \times p}$$

$$V = (\vec{v}_1, \dots, \vec{v}_p)_{p \times p}$$

our new
basis

$\downarrow E-Y$

For a
given k :

$$\hat{Y}_{N \times p}^T = \left[U_k \Sigma_k \right]_{N \times k} V_{k \times p}^T$$

closest
approximation
of
rank (k)

$$\left\{ \begin{array}{l} (Y_{PCA}) = \left[U_k \Sigma_k \right]_{N \times k} \\ \text{coordinates of the} \end{array} \right\}$$

$$V_k = (\vec{v}_1, \dots, \vec{v}_k) \quad \vec{v}_i \in \mathbb{R}^p$$

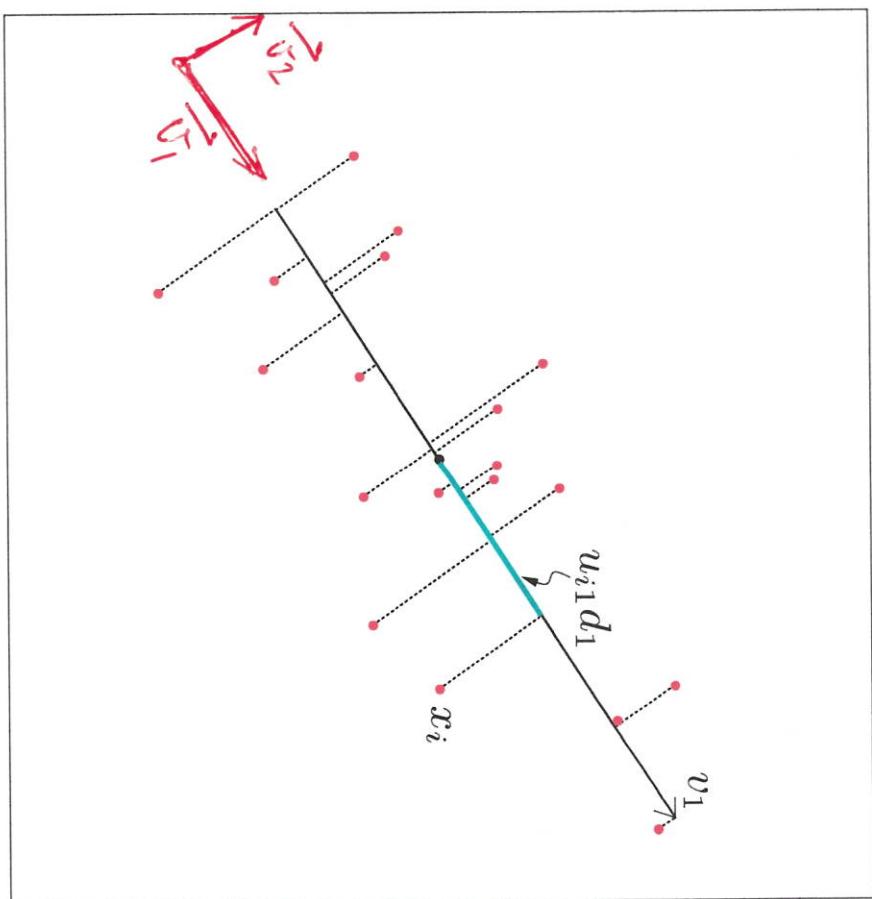
$$\text{where } \Sigma_k = \text{diag}(\sigma_1, \dots, \sigma_k)$$

$$\sigma_1 > \sigma_2 > \dots > \sigma_k$$

ordered
singular
values.

Examples overleaf →

Illustration of PCA and SVD



$$\vec{y}_i \in \mathbb{R}^2$$

$$\begin{aligned}\vec{y}_i &= \sum_{i=1}^{i=N} \alpha_i^{(i)} \vec{v}_1 + \alpha_2^{(i)} \vec{v}_2 \\ &= \sigma_1 u_{i1} \vec{v}_1 + \sigma_2 u_{i2} \vec{v}_2 \\ &= (\vec{v}_1^\top \vec{y}_i) \vec{v}_1 + (\vec{v}_2^\top \vec{y}_i) \vec{v}_2\end{aligned}$$

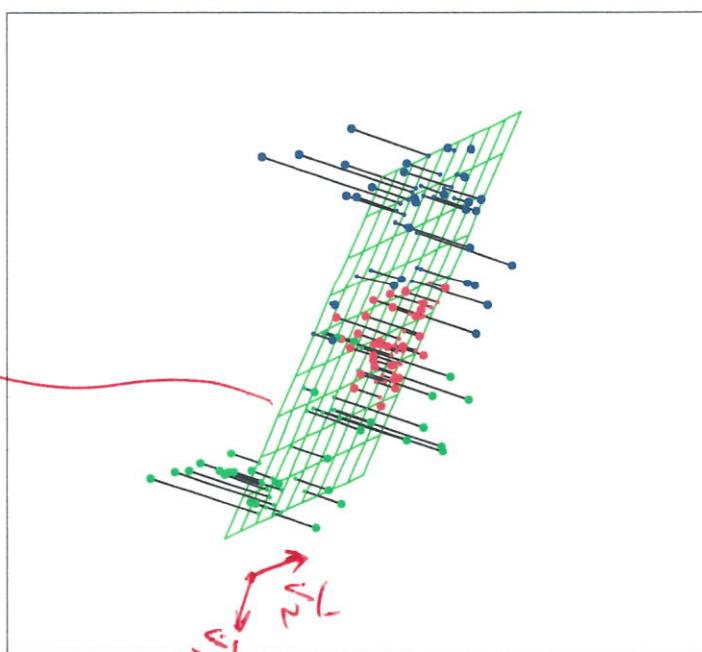
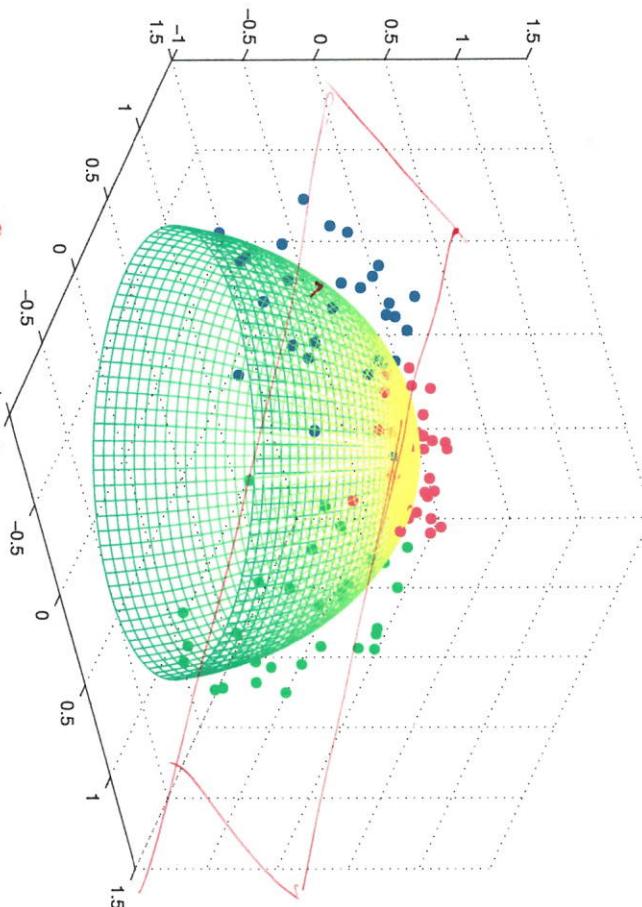
If we approximate by first component, we minimize the sum of perpendicular errors.

New Basis

\vec{v}_1 captures maximum variability

Illustration of PCA and SVD

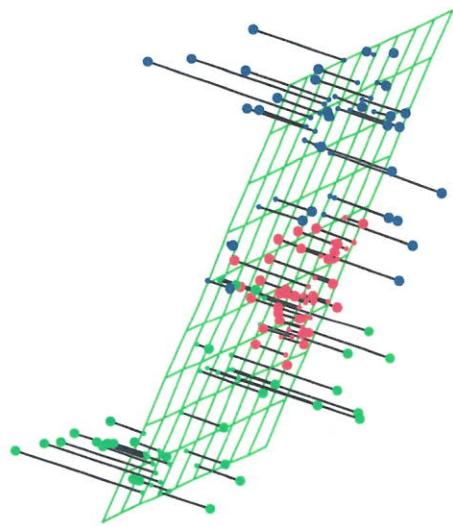
$$\vec{y}_i \in \mathbb{R}^3$$



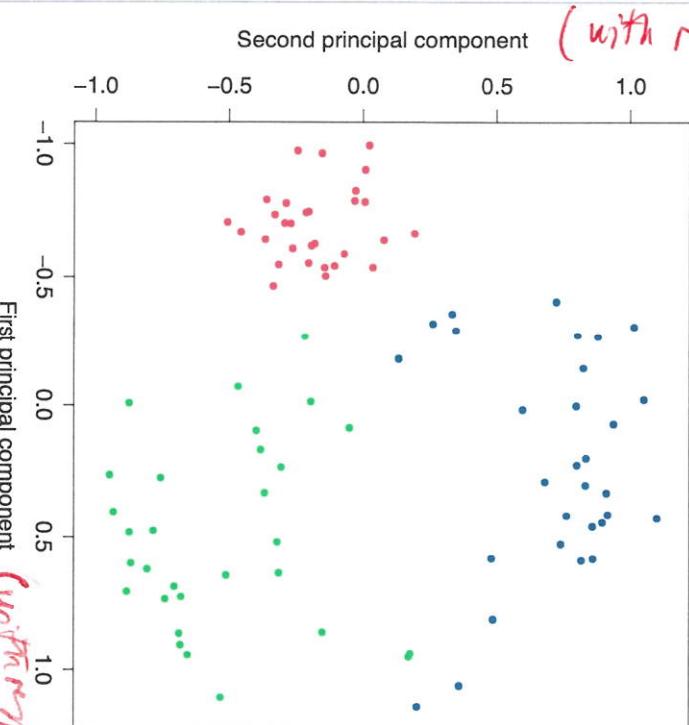
✓ Best approximation given
by hyperplane (\vec{v}_1, \vec{v}_2)

Illustration of PCA and SVD

$p=3$



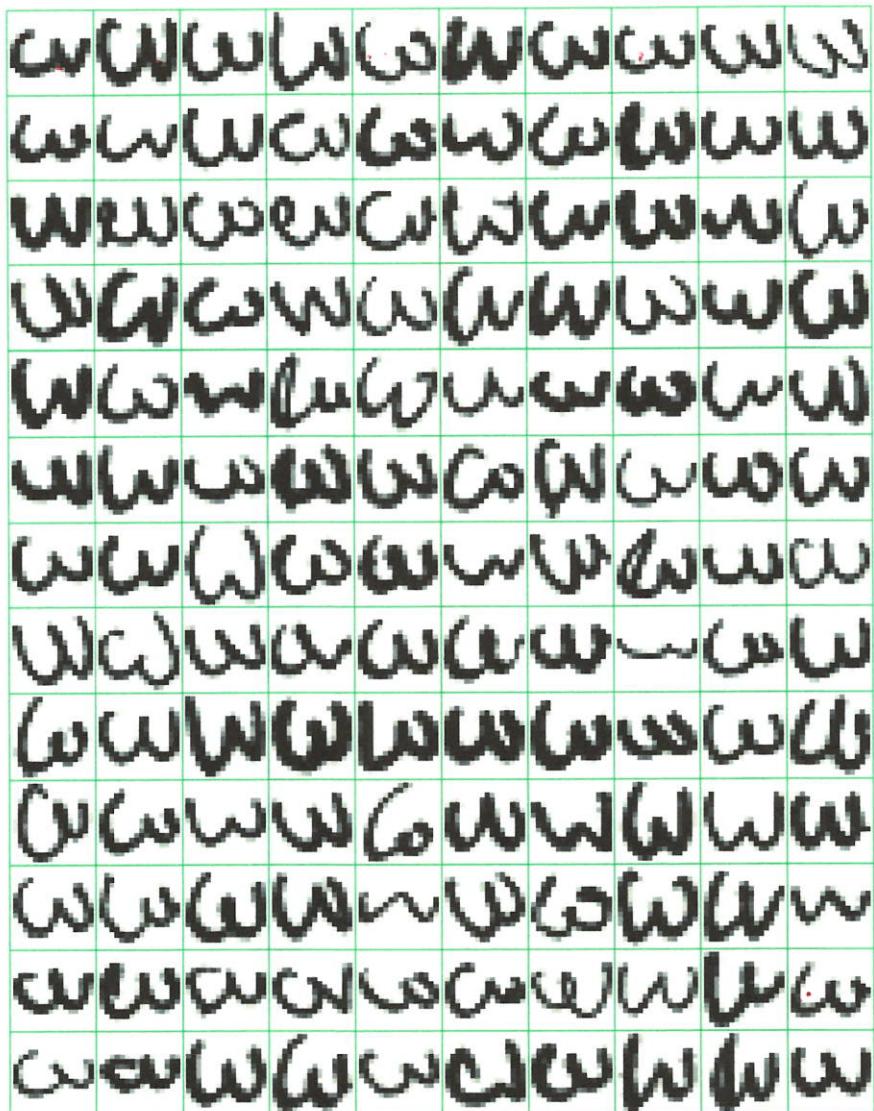
(with respect to \tilde{v}_2)



(\tilde{y}_{PCA}) $_{N \times 2}$ = coordinates in
(\tilde{v}_1, \tilde{v}_2)

$\tilde{y}_{N \times p}^T$

Illustration of PCA and SVD



Sample:
 $\{\vec{y}_i\}_{i=1}^{130}$

$N = 130$ images
 Vector:
 $\vec{y}_i \in \mathbb{R}^{16 \times 16}$

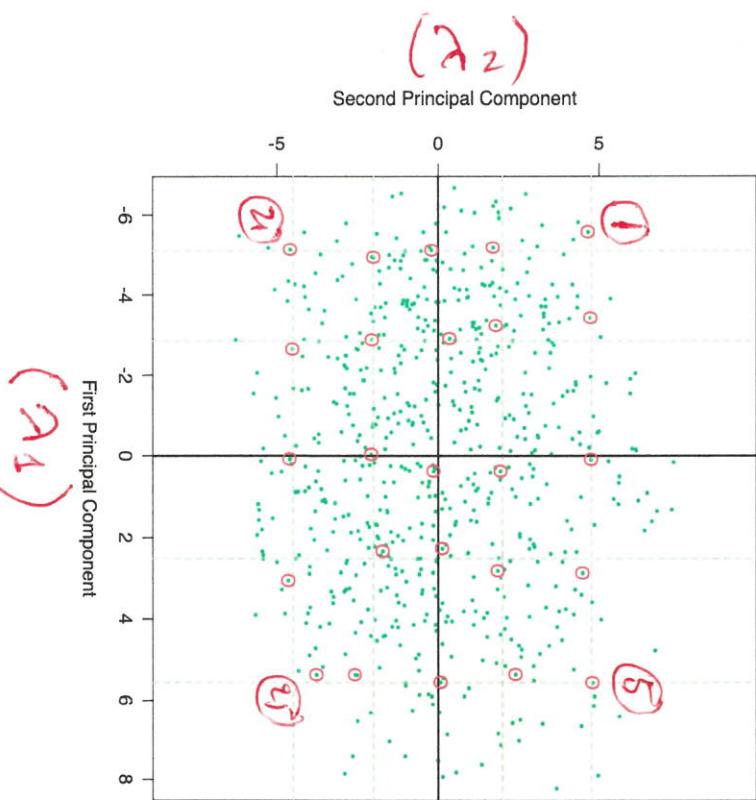
Each \vec{y}_i contains
 the grayscale of
 the images with
 256 pixels.

Illustration of PCA and SVD

Vectors expressed in terms of the first two v_i .

$$\hat{f}(\lambda) = \bar{x} + \lambda_1 \bar{v}_1 + \lambda_2 \bar{v}_2 \\ = \boxed{3} + \lambda_1 \cdot \boxed{4} + \lambda_2 \cdot \boxed{5}.$$

Illustration of PCA and SVD



The 3 becomes
less thick

PC_1
(The bottom part of the
3 becomes more curved)

First extension:

1. PCA is about linear projections that minimize a quadratic. How do we extend?

① "Nonlinearity": Kernel PCA

$$(\tilde{y}^T \tilde{y})_{ij} = (\vec{\tilde{y}_i}^T \vec{\tilde{y}_j})$$

↓
PCA was about eigenvalues of $\tilde{y}^T \tilde{y}$ and $\tilde{y} \tilde{y}^T$

↓
which are "Kernel" matrices.

If we assume a kernel function

$$k_{ij} = k(\tilde{y}_i, \tilde{y}_j)$$

↑
positive definite, etc. (see SVMs)

Then

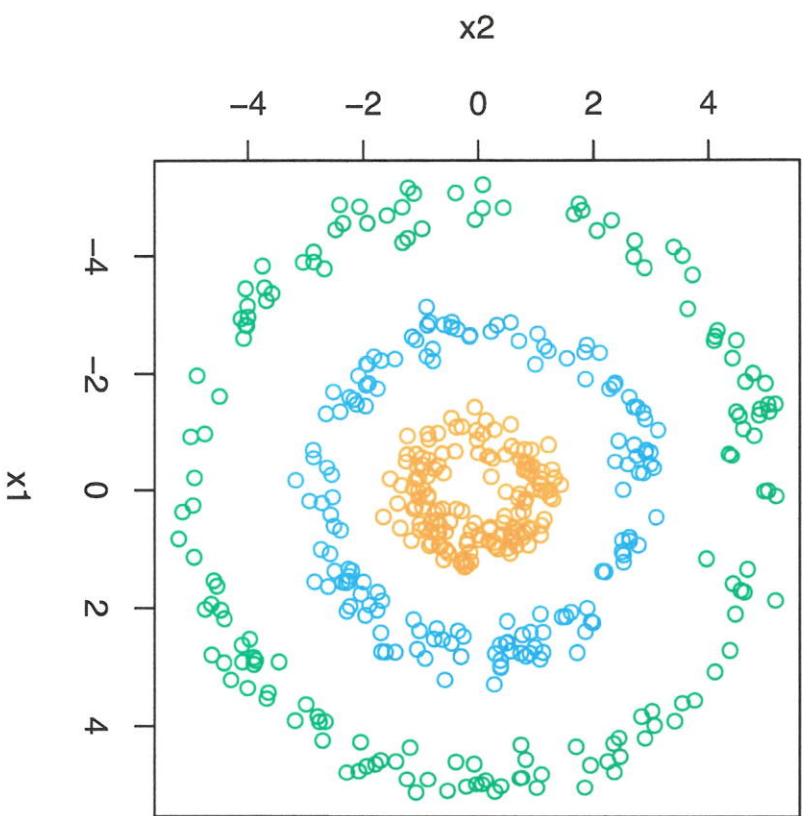
Eigen decomposition of K (kernel matrix)

$$\text{e.g. } K(\vec{x}_i, \vec{x}_j) = e^{-\frac{\|\vec{x}_i - \vec{x}_j\|^2}{c}}$$

Radial Kernel

→ Example overleaf

We can do spectral analysis of the kernel (which is nonlinear in the original coordinates)

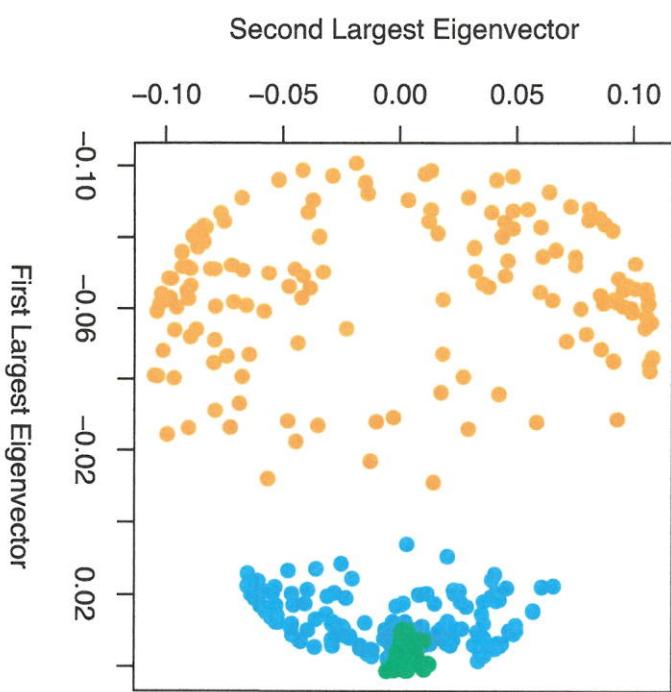


Representing these data in terms of standard PCA does not lead to a good mapping.

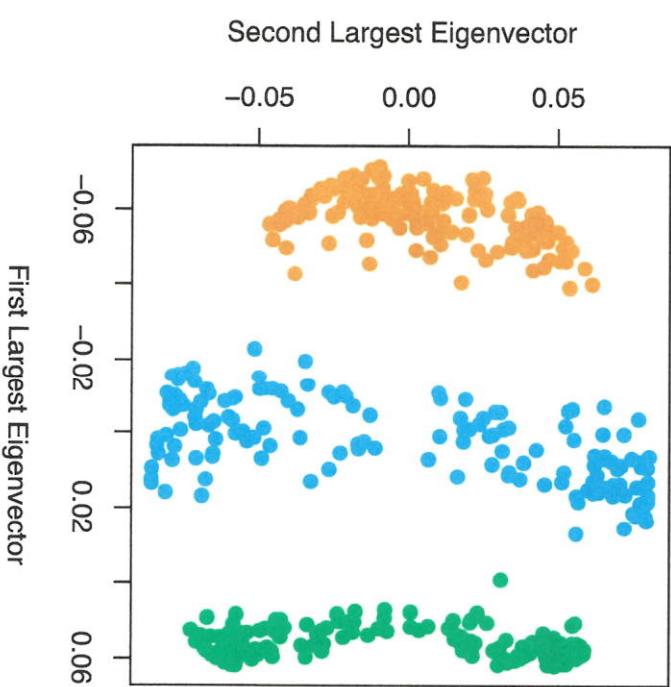
Kernel PCA

Increasing the hyperparameter c

Radial Kernel ($c=2$)



Radial Kernel ($c=10$)



Second extension: Interpretability through sparsity.

Sparse PCA

Our description from standard PCA was of the form:

$$(\mathbf{y}_{\text{PCA}})_{N \times K} = (\mathbf{U}_K \mathbf{\Sigma}_K)_{N \times K}$$

↓
coordinates in terms of

$$\mathbf{V}_K = (\tilde{v}_1 \dots \tilde{v}_K)_{P \times K}$$

The basis \mathbf{V} is non sparse in terms of our original descriptors

i.e. \mathbf{V} is not a sparse matrix.

Original PCA:

This is an alternative description of PCA

$$\max_{\tilde{v}} \tilde{v}^T (\tilde{Y} \tilde{Y}^T) \tilde{v} \quad \tilde{v} \in \mathbb{R}^P$$

such that $\tilde{v}^T \tilde{v} = 1$

Sparse

by adding extra constraint:

$$\sum_{j=1}^P |v_j| \leq t$$

Similar to LASSO

$$\{\tilde{v}_{j,L}\}_{j=1}^P$$

SVD LASSO basis

Find a basis that tries to minimize the quadratic error under sparsity constraints.

If we find a description in terms of a few vectors that are sparse, then the description is closer to identifying the important combinations of descriptors.

$$\hat{y} = \hat{f}(\lambda) = \sum_{j=1}^k \lambda_j \vec{v}_{j,L}$$

$k \ll$ from LASSO.

Then the matrix V will be sparser and the \vec{v}_j will contain more zeros.

$$\vec{v}_{j,L} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad \begin{array}{l} \xrightarrow{\hspace{1cm}} \text{coordinate for } x_1 \\ \xrightarrow{\hspace{1cm}} \text{coordinate for } x_2 \end{array}$$

Third extension: Interpretability through non-negativity

Non-negative matrix factorisation (NNM)

Standard.

PCA / SVD

$$\hat{y}^T = (U_K \Sigma_K) V_{K \times p}^T$$

↓ the same matrix

$$X = (U_K \Sigma_K) V_{K \times p}^T$$

$\underbrace{N \times K}_{\text{NMF}}$ $\underbrace{K \times p}_{\text{NMF}}$

Generalise to a factorisation:

$$X = W_{N \times K} H_{K \times p}^T$$

where
we require positivity

$$w_{ij} \geq 0 \quad \forall i, j$$

$$h_{ij} \geq 0 \quad \forall i, j$$

It can be shown to be equivalent to:

$$\max_{W, H} L(W, H) = \sum_{i=1}^N \sum_{j=1}^K [x_{ij} \log (WH)_{ij} - (WH)_{ij}]$$

$x_{ij} \sim \text{Poi}$ with mean $(WH)_{ij}$

What does ~~NMF~~ give?

See over

for an example with images

Non negative matrix factorisation

$$\{\vec{y}_i\}_{i=1}^N \quad \vec{y}_i \in \mathbb{R}^P$$

Each \vec{y}_i is an N-dimensional vector

The sample is a 381-dimensional image 19×19

$N=2429$

$$p=381=19 \times 19$$

$k=49$

$$\text{H}^{(49 \times 381)}$$

pixels of the feature maps

$$VQ = k\text{-means}$$

$$\text{Original} \xrightarrow{(49 \times 19)} \text{W}^{(49 \times 49)} \times \text{H}^{(49 \times 381)} = \text{all positive}$$

- Setup:
- Each \vec{y}_i is an image 19×19 pixels, which is vectorized into a 381-dimensional vector.
 - Number of images in sample $N=2429$

The 49 faces closest to "original"

$$\text{PCA} \xrightarrow{(49 \times 19)} \text{W}^{(49 \times 49)} \times \text{H}^{(49 \times 381)} = \text{all positive}$$

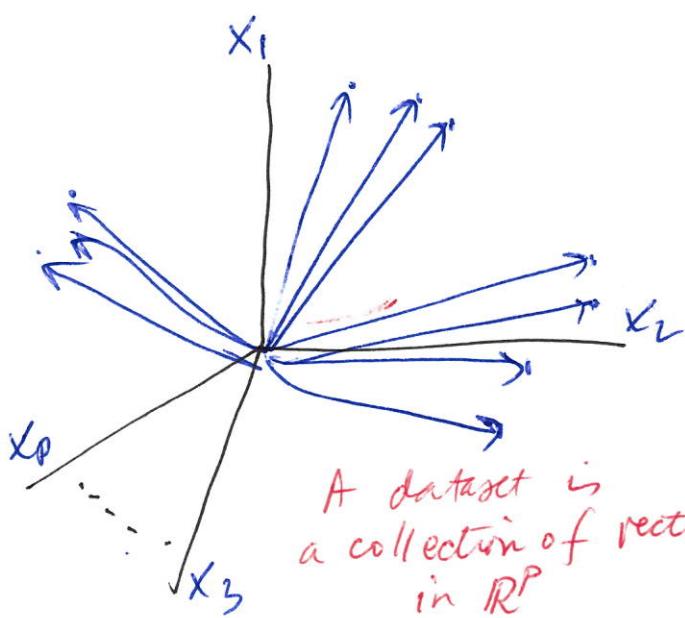
This whole set of squares is the (V_k) pixel matrix

Each of the squares is a $\vec{y}_i \in \mathbb{R}^{381}$



$$\xrightarrow{(49 \times 19)} \text{W}^{(49 \times 49)} \times \text{H}^{(49 \times 381)} = \text{all positive}$$

Graph-based learning



$$\{\vec{y}_i\}_{i=1}^N \quad i=1, \dots, N$$

$$\vec{y}_i \in \mathbb{R}^p$$

$$\vec{y}_i = \begin{pmatrix} x_1^{(i)} \\ \vdots \\ x_p^{(i)} \end{pmatrix}$$

$$y^T_{N \times p} = X_{N \times p} = \begin{pmatrix} \vec{y}_1^T \\ \vdots \\ \vec{y}_N^T \end{pmatrix}$$

Vectors in \mathbb{R}^p :

→ Geometry of the dataset
in relation to { clustering & reduced dimensionality }

key ingredient:

Similarity or dissimilarity
(in some cases a distance)

between samples.

An $N \times N$ matrix summarizing
the relationships (pairwise)
between samples

Examples :

dissimilarities $D_{N \times N}$

↑
similarities $S_{N \times N}$

e.g.

$$D_{ij} = \|\vec{y}_i - \vec{y}_j\|^2 \quad (\text{distance})$$

"dissimilarity"

$$S_{ij} = \frac{\vec{y}_i \cdot \vec{y}_j}{\|\vec{y}_i\| \|\vec{y}_j\|} \quad (\text{cosine similarity})$$

$$S_{ij} = g(\vec{y}_i, \vec{y}_j) = \frac{\text{cov}(\vec{y}_i, \vec{y}_j)}{\sqrt{\text{cov}(\vec{y}_i, \vec{y}_i)} \sqrt{\text{cov}(\vec{y}_j, \vec{y}_j)}}$$

$$D_{ij} = e^{-\frac{\|\vec{y}_i - \vec{y}_j\|^2}{c}} \quad (\text{statistical similarity})$$

(a kernel matrix)

In most
of our
unsupervised
learning methods:

Pairwise similarities between samples
are often important determinants
of the dataset.

In other
problems:

∴ In some cases the problem is
directly defined by the
pairwise similarities (we don't
even know what \vec{y}_i are, just S_{ij})

From

$S_{N \times N}$

to

Graph representation
of the dataset
that follows from
this information.

Some definitions: Combinatorial object on
Graph: Two sets : nodes and edges.

$G(V, E)$

$V = \text{set of vertices (nodes)}$ $i=1, \dots, N$

$E = \text{set of edges (links)}$

(i, j) [Pair of edges]

In our case:

The set of nodes is the set of samples:

$$\left\{ \vec{y}_i \right\}_{i=1}^N \longleftrightarrow \left\{ i \right\}_{i=1}^N \quad \begin{matrix} \text{and the} \\ \text{edges} \\ \text{represent} \\ S_{ij} \end{matrix}$$

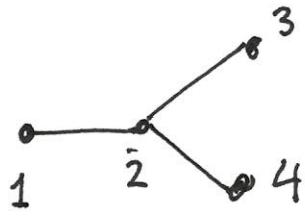
Example:

$G(V, E)$

$V = \{1, 2, 3, 4\}$

$N = 4$

$$E = \left\{ \begin{array}{l} (1, 2) \\ (2, 4) \\ (2, 3) \end{array} \right\}$$



Undirected.

Equivalent representation:

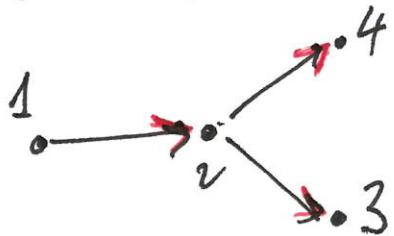
Adjacency matrix: $A_{N \times N}$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A_{ij} = \begin{cases} 0 & \text{if } i=j \text{ (connected)} \\ 1 & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

If undirected, $A = A^T$

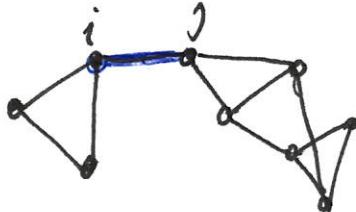
Directed graph.



$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A \neq A^T$$

Connected graph: every pair of nodes is connected by a path (weak, for undirected)



connected

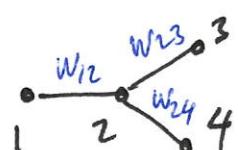
①



Two disconnected components.

See next page for
① ②

Weighted graphs:



$$W = \begin{bmatrix} 0 & w_{12} & 0 & 0 \\ w_{12} & 0 & w_{23} & w_{24} \\ 0 & w_{23} & 0 & 0 \\ 0 & w_{24} & 0 & 0 \end{bmatrix}$$

Components in the graph:

If A is connected, then

$A_{N \times N}$ has full rank

and cannot be rearranged in
block-diagonal form (irreducible)

If we have more than one
disconnected connected component
then A can be written in
block diagonal form:

①

$$A = \begin{bmatrix} \text{diag} & 0 \\ 0 & \text{diag} \end{bmatrix}$$

After
edge
is gone

②

$$A = \begin{bmatrix} \text{diag} & 0 \\ 0 & \text{diag} \end{bmatrix}$$

First
connection

: Clustering problem is related to
making the adjacency matrix
as block-diagonal as possible
by relabelling of the nodes.

[Remember that nodes are samples]

Second connection : distances between samples

Distances on graphs.

- Minimal distance on the graph (geodesic):
geodesic Distance between nodes i and j in an undirected graph:

$$\min_{\{i,j\}} \# \text{edges in } \{i,j\}$$

where $\{i,j\}$ denotes the set of paths between i and j in the graph.

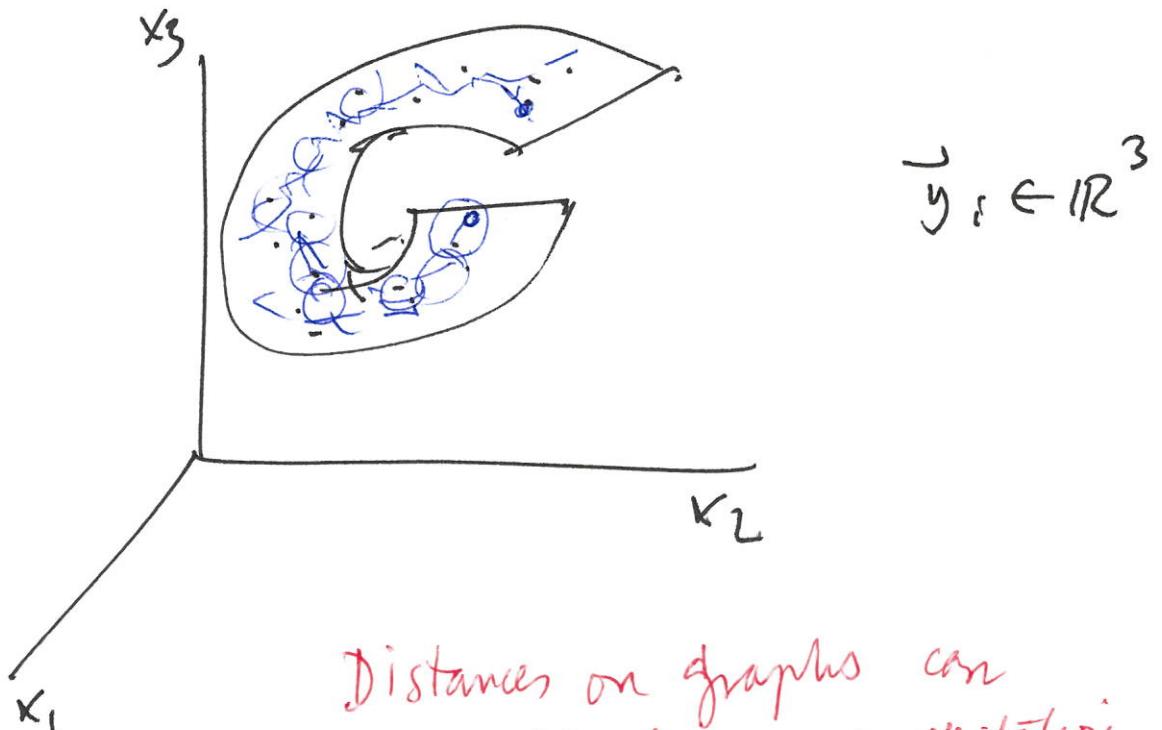
[Breadth first search algorithms]
(BFS)

- Average distance between i,j :

$$\langle \# \text{edges} \rangle_{\{i,j\}}$$

where $\langle \cdot \rangle_{\{i,j\}}$ denotes the average over the set of paths between (i,j)

If graph is weighted, the length of a path is the sum of the weights of the edges on the path



Distances on graphs can provide better representations of the intrinsic geometry of the dataset.

Quick recap:

Graph-based learning

Two situations:

$$\textcircled{1} \quad \left\{ \vec{y}_i \right\}_{i=1}^N, \quad \vec{y}_i \in \mathbb{R}^P$$

A dataset as a cloud of points
in \mathbb{R}^P {vectors}



$$D_{N \times N} \text{ or } S_{N \times N}$$

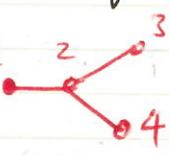
summarizes pair-wise interactions
between samples.

- $\textcircled{2}$ Our dataset is directly given by
pairwise relationships between samples
or observations (entries)

Both in $\left\{ \begin{array}{l} \textcircled{1} \quad \text{Data} \rightarrow D_{N \times N} \text{ or } S_{N \times N} \\ \text{or} \\ \textcircled{2} \quad \text{we only have } D_{N \times N} \end{array} \right\}$

We can represent the dataset as a graph
where the nodes (or vertices) are the
samples and the links (or edges) are
the pair-wise relationships:

Definition: Graph: $G = \{V, E\}$ $V = \{\vec{v}_i\}_{i=1}^N$

e.g.  $V = \{1, 2, 3, 4\}$ $E = \{(1, 2), (2, 3), (2, 4)\}$

Graph can be

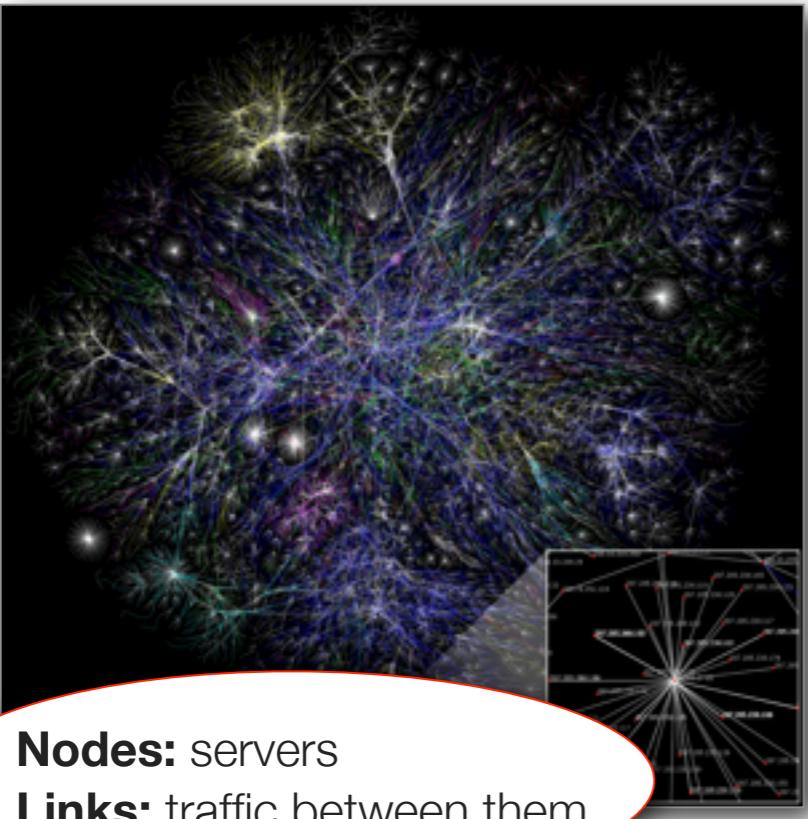
[undirected / directed]
[unweighted / weighted]

Why graphs?

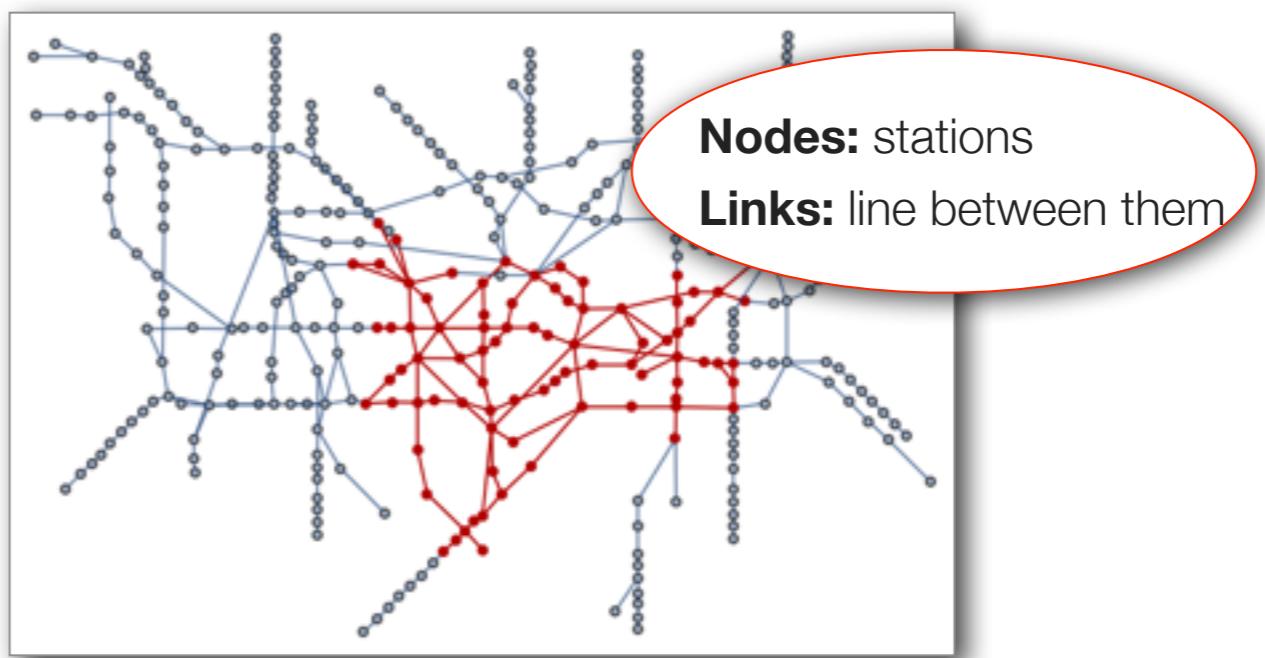
- ▶ In our Machine Learning lectures we looked at problems where we have $i=1,2,\dots,N$ observations.
- ▶ But we assumed (implicitly or explicitly) that these N samples were independent from each other - i.e. that they do not interact with one another.
- ▶ Sometimes there are many interactions, and in fact, such interactions are a **crucial feature of the data**.
- ▶ Many interactions in the real world:
 - ▶ social media (follow, re-tweet), friendships, family
 - ▶ proteins in cells bind to one another
 - ▶ transport (shared commute, same to/from flights)
 - ▶ etc, etc...

A few networks

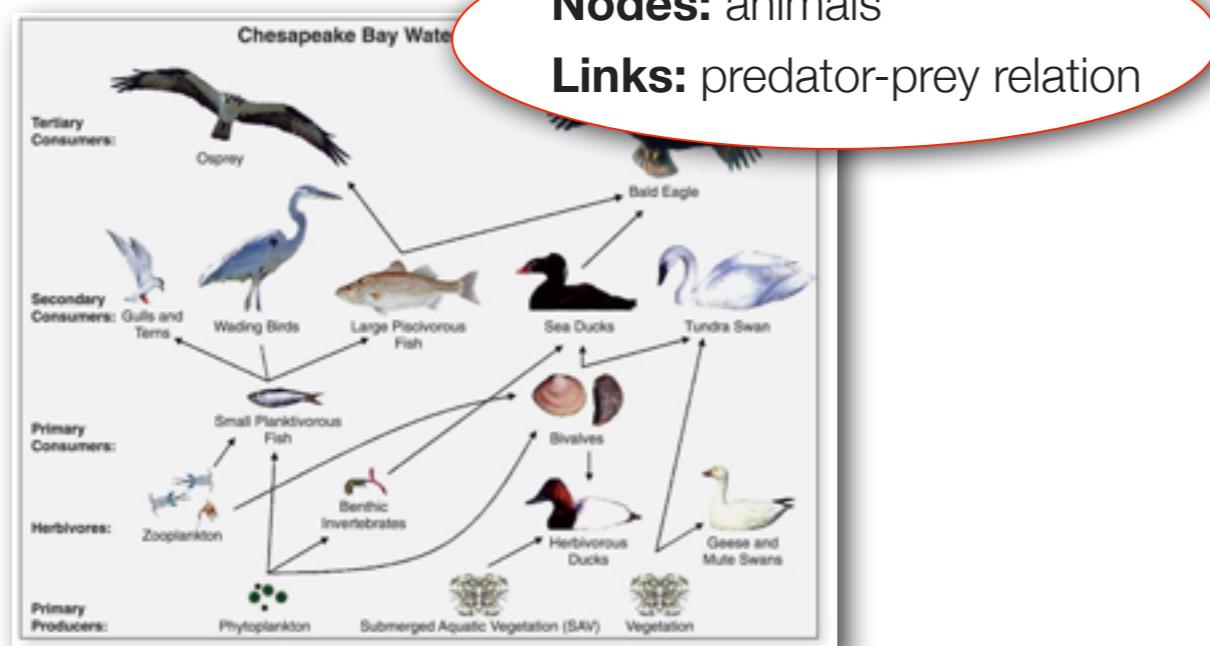
The internet



London Underground

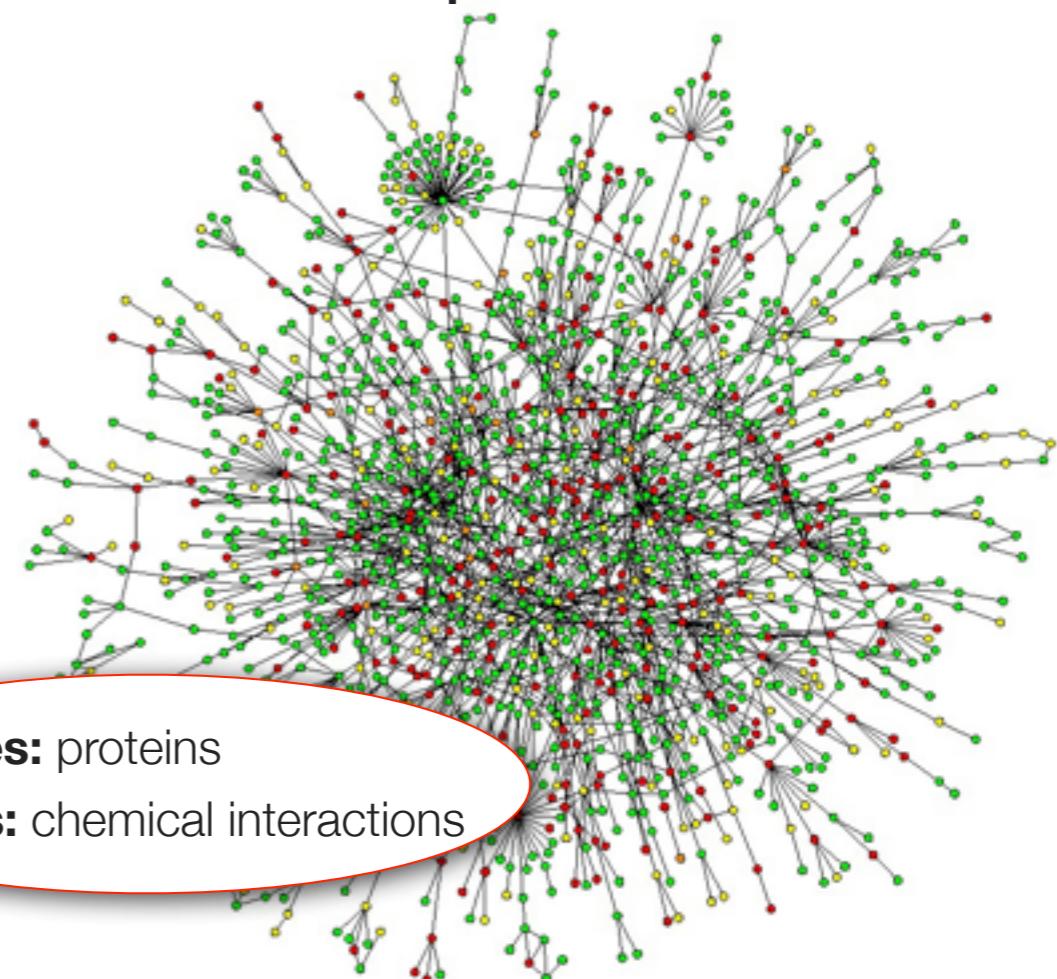


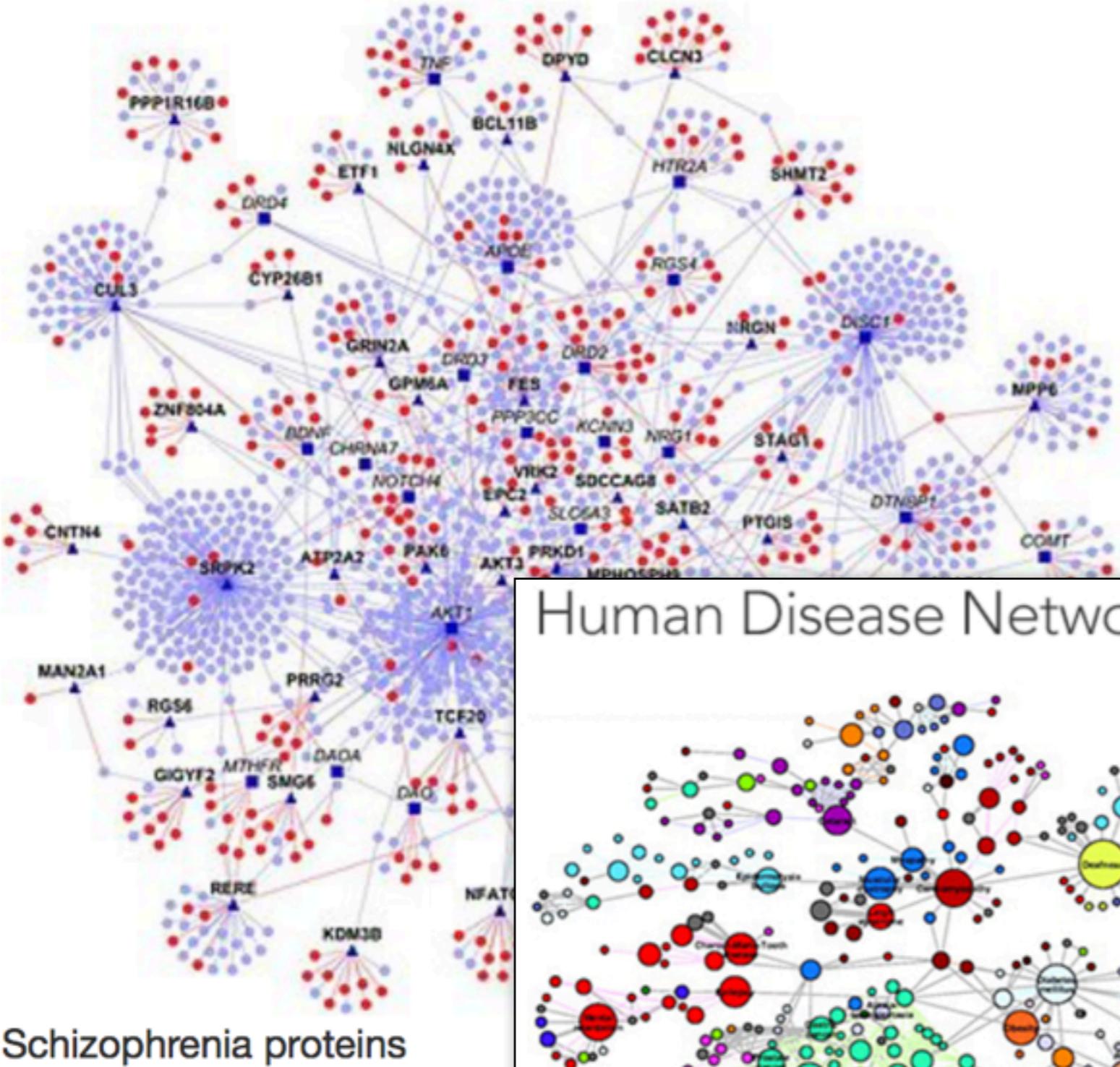
Food webs



Nodes: proteins
Links: chemical interactions

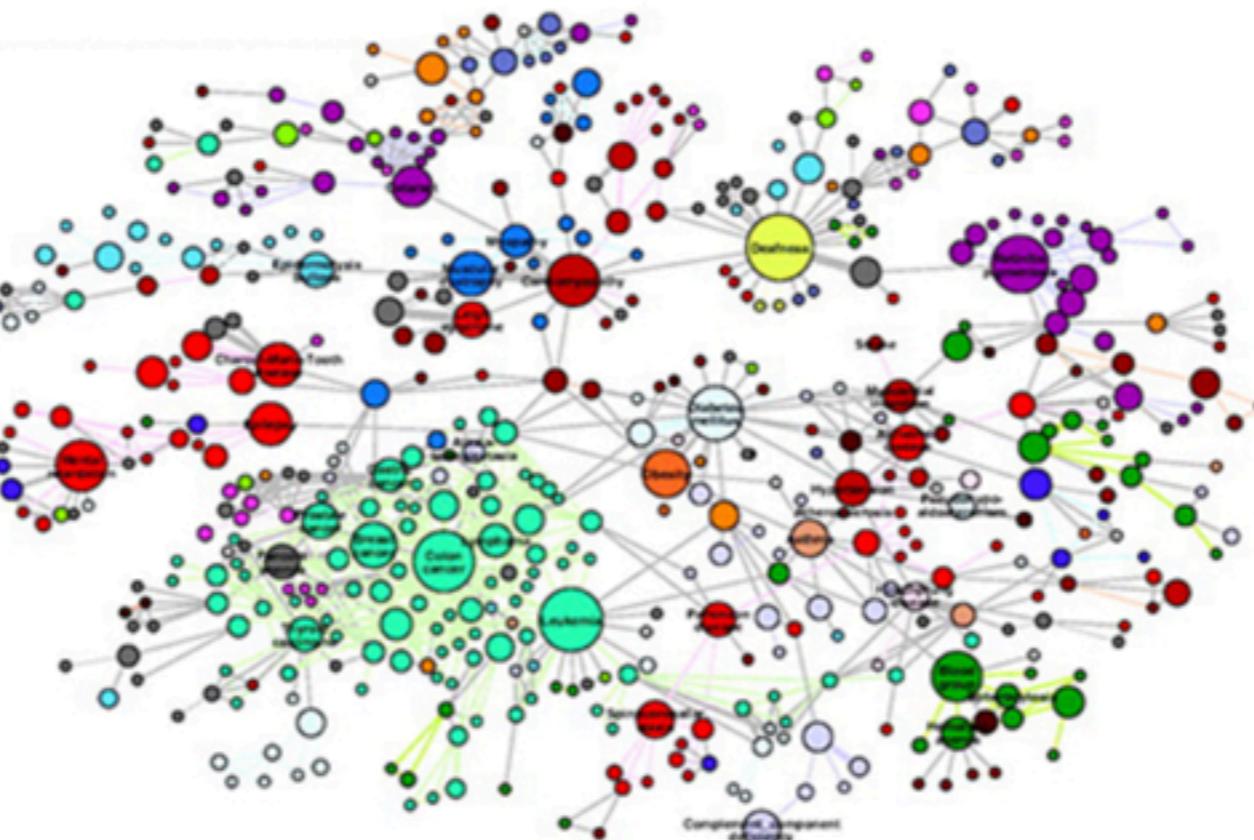
S cerevisiae protein interactions





Schizophrenia proteins

Human Disease Network



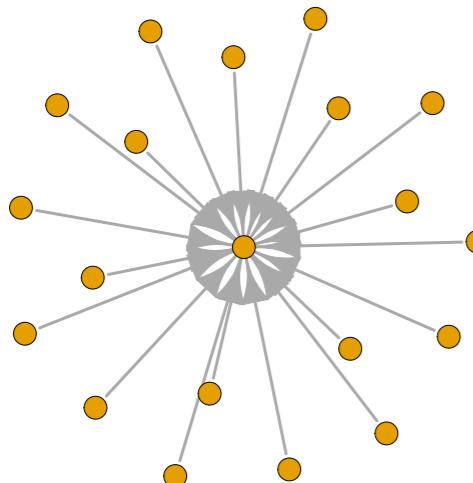
Disorder Class

- Bone
- Cancer
- Cardiovascular
- Connective tissue
- Dermatological
- Developmental
- Ear, Nose, Throat
- Endocrine
- Gastrointestinal
- Hematological
- Immunological
- Metabolic
- Muscular
- Neurological
- Nutritional
- Ophthalmological
- Psychiatric
- Renal
- Respiratory
- Skeletal
- multiple
- Unclassified

Network science

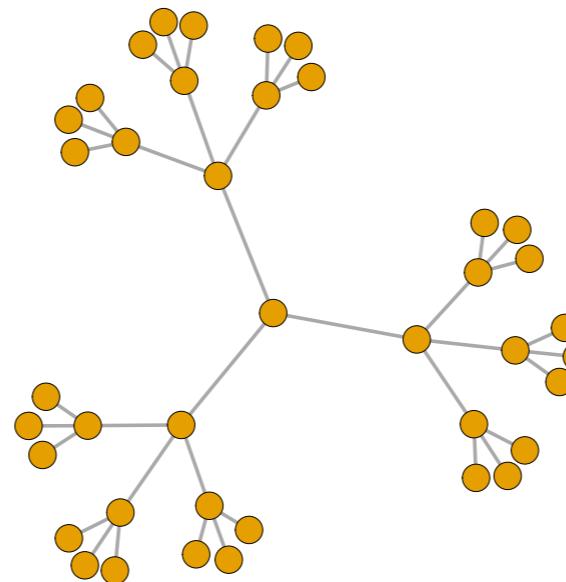
- ▶ Network science provides methods to study such complex interactions.
- ▶ Especially useful for ***large*** networks, i.e. those with many components (players) and complex connectivity.

Small networks are intuitive



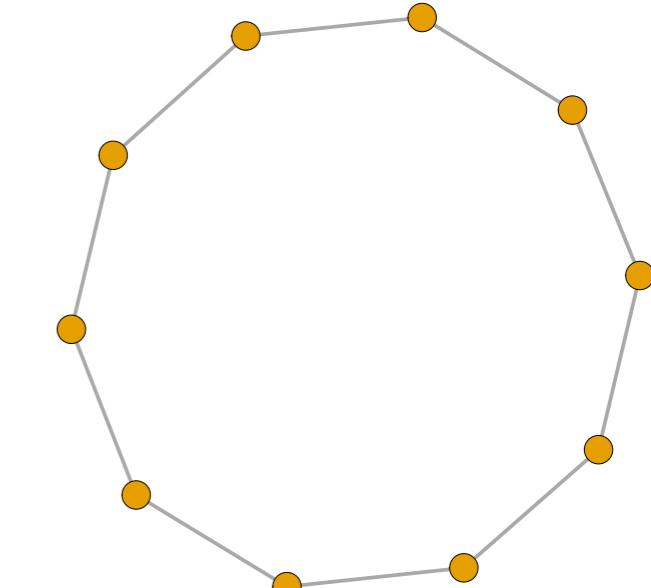
star

one hub



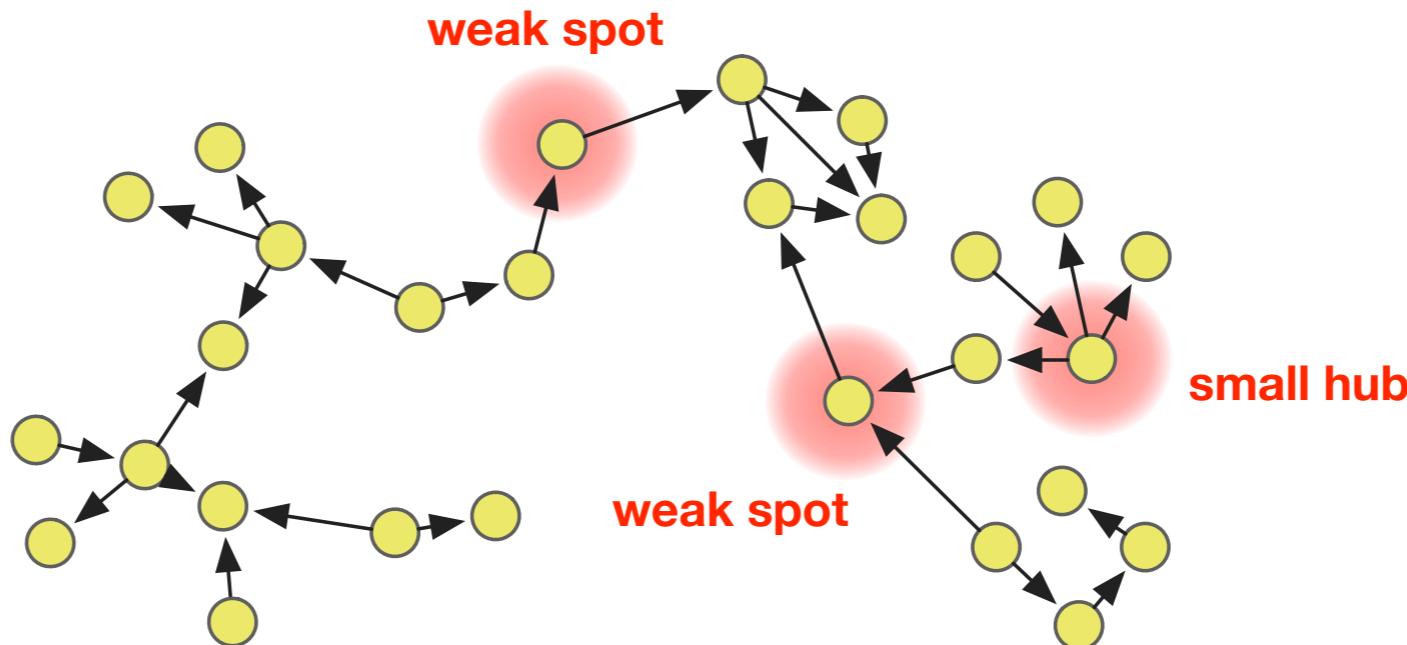
tree

several small hubs

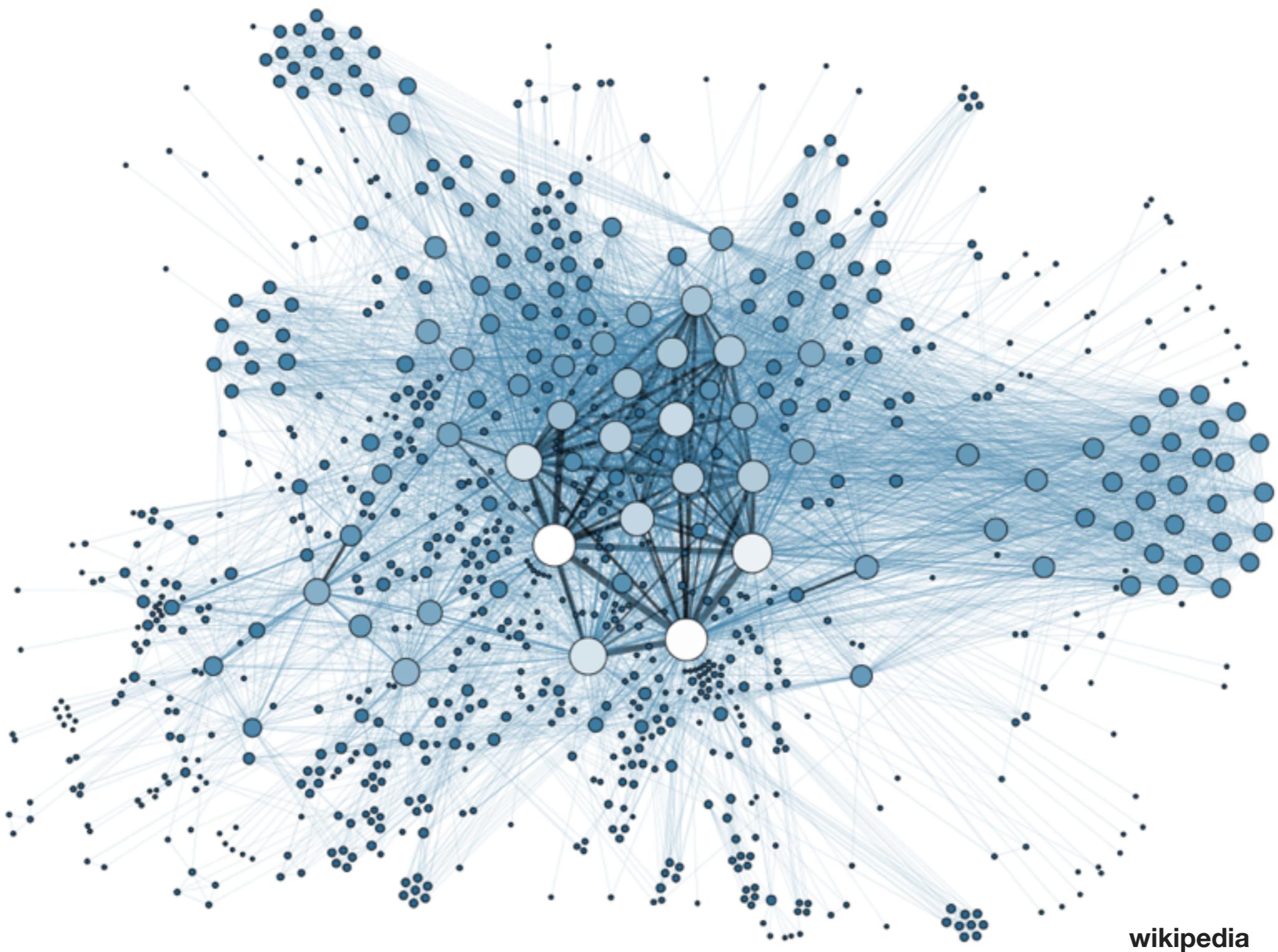


ring

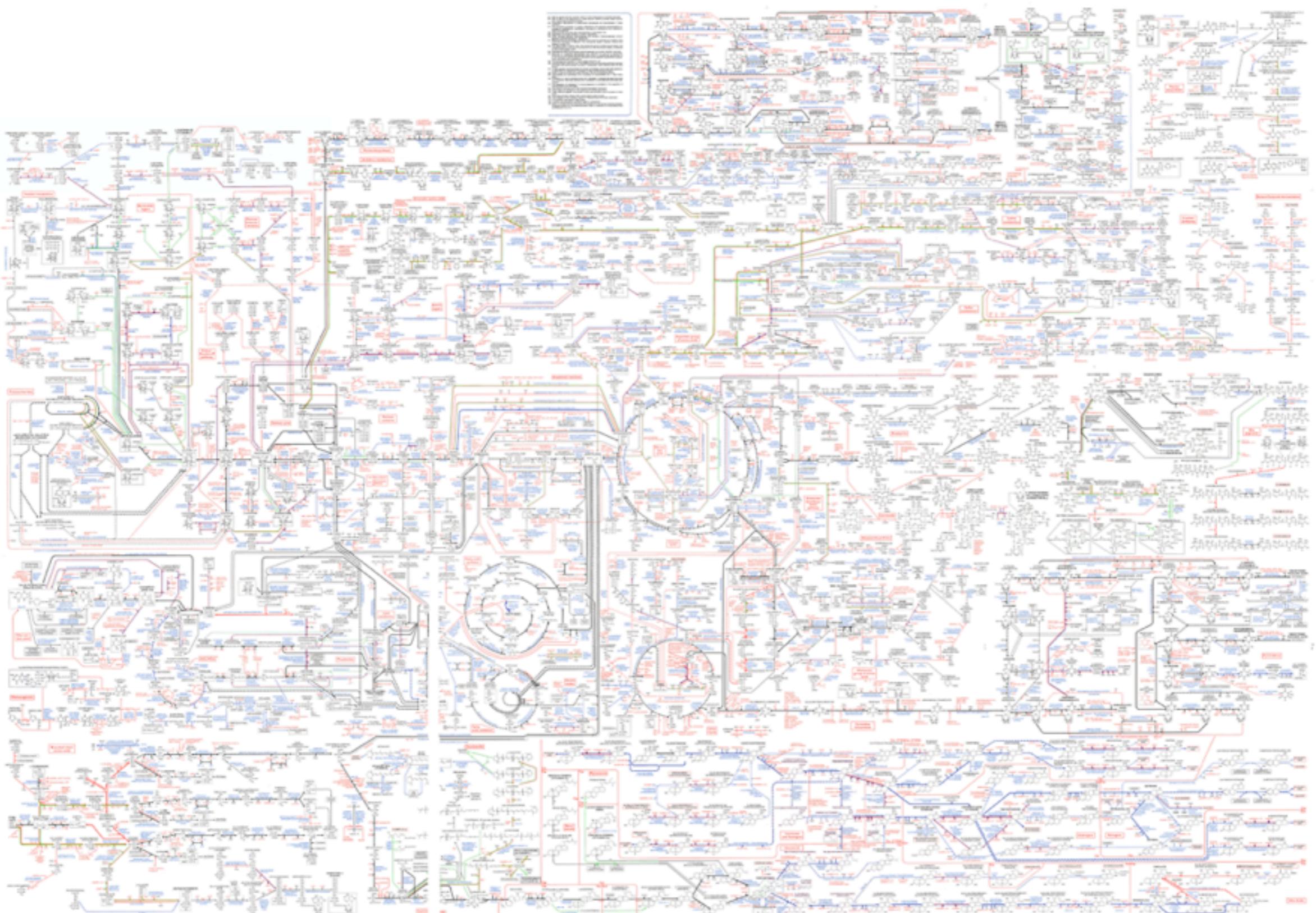
only two links per node



But large ones are not (at all)

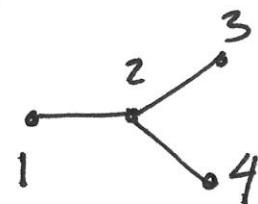


But large ones are not (at all)



Human metabolism

Example:



Undirected

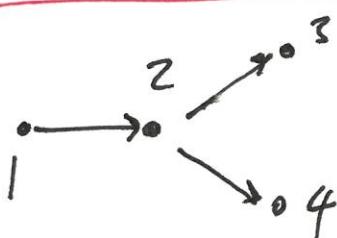
$N=4$

$$A_{N \times N} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A\vec{1} = \begin{bmatrix} 1 \\ 3 \\ 1 \\ 1 \end{bmatrix} = \vec{d} \quad \begin{array}{l} \text{degree of} \\ \text{node 1} \end{array}$$

\downarrow degree vector

Directed



$$A = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$A \neq A^T$$

$$A\vec{1} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \vec{d}_{in} \quad \text{in-degree}$$

$$A^T\vec{1} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 0 \end{bmatrix} = \vec{d}_{out} \quad \text{out-degree}$$

(1) Graph construction from data

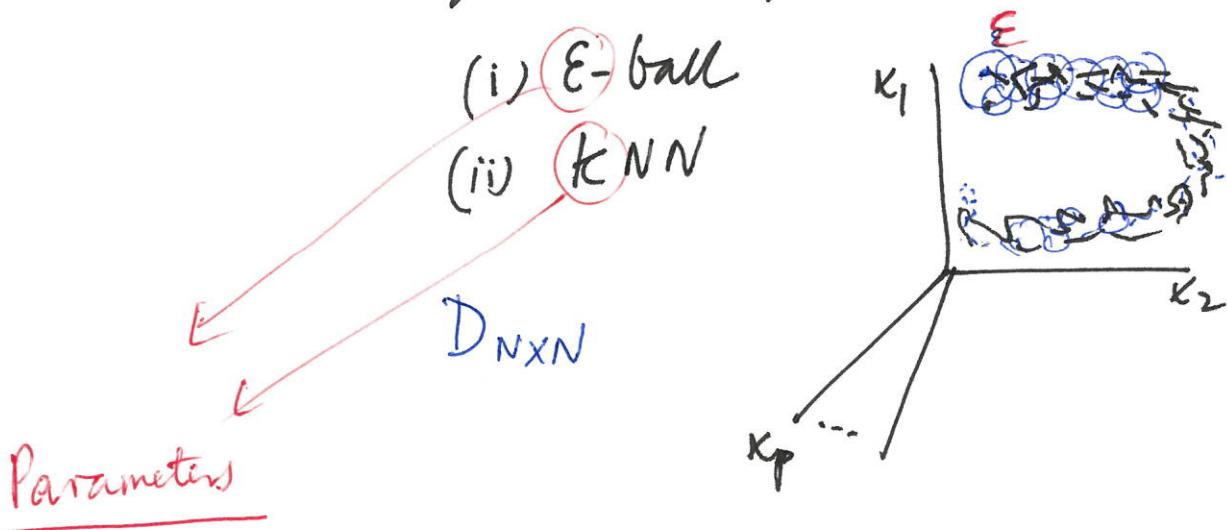
We do not have a graph

but just $\{y_i\}_{i=1}^N$

$$\rightarrow D_{N \times N}, S'_{N \times N}$$

Adjacency of weighted complete graphs.

- Thresholding $S'_{N \times N}$
is not the best strategy
(usually precision matrix is important)
- Geometric graphs :



The "other" matrix (besides A):

Laplacian matrix

Extension of the Laplacian operator:

① Recall the PDE:

$$u = u(x, t), \frac{\partial u}{\partial t} = \underbrace{\nabla^2 u}_{\text{Laplacian}}$$

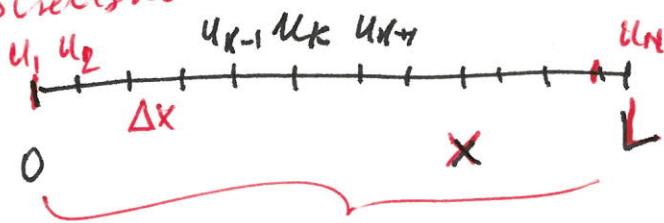
{ Heat equation
or
Diffusion equation

In this case:

$$\nabla^2 u = \frac{d}{dx} \left(\frac{du}{dx} \right)$$

$x \in [0, L], t \in \mathbb{R}^+$

Discretisation:



$$\vec{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix}_{N \times 1}$$

$$\nabla^2 u = \frac{d}{dx} \left(\frac{du}{dx} \right) \approx \Delta (\Delta u) = \Delta (u_{k+1} - u_k)$$

$$= (u_{k+2} - u_{k+1}) - (u_{k+1} - u_k)$$

$$= u_{k+2} - 2u_{k+1} + u_k$$

Can be rewritten
in terms of a
matrix →

Equation becomes:

$$\frac{d\vec{u}}{dt} = \begin{pmatrix} -1 & 1 & & & & \\ & -2 & 1 & & & \\ & & 1 & -2 & 1 & \\ & & & 1 & -2 & 1 \\ 0 & \dots & 1 & -2 & 1 & 0 & \dots & 0 \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -1 & \\ 0 & & & & & & & \\ & & & & & & & \end{pmatrix}_{N \times N} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{k+1} \\ u_{k+2} \\ \vdots \\ u_N \end{pmatrix}$$

Laplacian matrix $\equiv -L$

Graph:



$$-L = A - D$$

where $D = \text{diag}(\vec{d}) = \text{diag}(A^T)$

Definition:

Combinatorial graph Laplacian.

$$L = D - A$$

$$\frac{d\vec{u}}{dt} = -L\vec{u}$$

Heat equation on
the graph.

Solution:

$$\vec{u}(t) = e^{-Lt} \vec{u}(0)$$

Note that: ① $L \vec{1} = (D - A) \vec{1} = \vec{d} - \vec{d} = 0$.

i.e., $\vec{1}_{N \times 1}$ is an eigenvector of L with eigenvalue zero.

② So $\vec{1}$ is a stationary point of the heat equation ✓

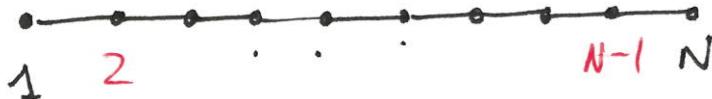
...

Equation becomes:

$$\frac{d\vec{u}}{dt} = \begin{pmatrix} -1 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & 1 & -2 & 1 & \\ 0 & \dots & 1 & -2 & 1 & 0 & \dots & 0 \\ & & & 1 & -2 & 1 & & \\ 0 & & & & 1 & -1 & & \\ & & & & & & & \end{pmatrix}_{N \times N} \begin{pmatrix} u_1 \\ \vdots \\ u_{k-1} \\ u_k \\ u_{k+1} \\ u_{k+2} \\ \vdots \\ u_N \end{pmatrix}$$

Laplacian matrix $\equiv -L$

Graph:



$$-L = A - D$$

where $D = \text{diag}(\vec{d}) = \text{diag}(A^T)$

Definition:

Combinatorial graph Laplacian.

$$L = D - A$$

$$\frac{d\vec{u}}{dt} = -L\vec{u}$$

Heat equation on the graph.

Solution:

$$\vec{u}(t) = e^{-Lt} \vec{u}(0)$$

Note that: ① $L \vec{1} = (D - A) \vec{1} = \vec{d} - \vec{d} = 0$.

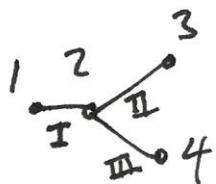
i.e., $\vec{1}_{N \times 1}$ is an eigenvector of L with eigenvalue zero.

② So $\vec{1}$ is a stationary point of the heat equation ✓

②

$$L = D - A = \underset{N \times N}{B^T} \underset{N \times E}{B} \underset{E \times N}{\dots}$$

B is the incidence matrix



$$B_{E \times N} = \begin{bmatrix} \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{array}{l} \text{I} \\ \text{II} \\ \text{III} \end{array}$$

$$B^T B = \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} = L$$

$$\vec{u}^T L \vec{u} \quad \vec{u} \in \mathbb{R}^N$$

||

$$\vec{u}^T B^T B \vec{u} = \|B\vec{u}\|^2 \geq 0$$

Positive semi definite matrix.

$$\vec{u}(t) = e^{-tL} \vec{u}(0)$$

$$L \vec{v}_i = \lambda_i \vec{v}_i$$

$$L V = V \Lambda \quad \Lambda = \text{diag}(\lambda_i)$$

$$V = (\vec{v}_1 \dots \vec{v}_N)$$

$$V V^T = I$$

$$L = V \Lambda V^T$$

$$e^{-tL} = \sum_{k=0}^{\infty} \frac{1}{k!} (-tL)^k$$

$$L^k = V \Lambda \underbrace{V^T V}_{I} V \Lambda V^T = V \Lambda^k V^T$$

$$e^{-tL} = V \left[\sum_{k=0}^{\infty} \frac{(-t)^k}{k!} \lambda^k \right] V^T$$

$\left(\begin{array}{cccc} \sum_{k=0}^{\infty} \frac{(-t)^k}{k!} \lambda_1^k & & & 0 \\ & \ddots & & \\ 0 & \ddots & \ddots & \sum_{k=0}^{\infty} \frac{(-t)^k}{k!} \lambda_N^k \\ & & & \end{array} \right)$

$$\left(\begin{array}{cccc} e^{-\lambda_1 t} & & & \\ & \ddots & & \\ & & \ddots & e^{-\lambda_N t} \\ & & & \end{array} \right)$$

$$\text{diag}(e^{-\lambda_i t})$$

$$\begin{aligned}
 e^{-tL} &= V \text{diag}(e^{-\lambda_i t}) V^T \\
 &= \sum_{i=1}^N e^{-\lambda_i t} \vec{v}_i \vec{v}_i^T
 \end{aligned}$$

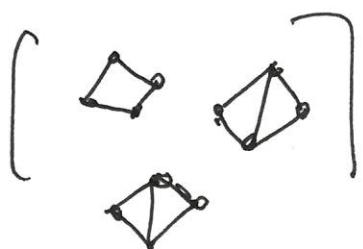
$$\begin{aligned}
 \vec{u}(t) &= e^{-tL} \vec{u}(0) \\
 &= \sum_{i=1}^N e^{-\lambda_i t} \vec{v}_i \left[\vec{v}_i^\top \vec{u}(0) \right] \\
 &= \underbrace{\left(\vec{1}^\top \cdot \vec{u}(0) \right)}_{\text{Fiedler eigenvector}} \vec{1} + \sum_{i=2}^N e^{-\lambda_i t} \vec{v}_i \left(\vec{v}_i^\top \cdot \vec{u}(0) \right)
 \end{aligned}$$

As $t \rightarrow \infty$

$$\vec{u}(t) - N \langle \vec{u}(0) \rangle \vec{1} \approx e^{-\lambda_2 t} \vec{v}_2 \left[\vec{v}_2^\top \cdot \vec{u}(0) \right]$$

Spectral connectivity Fiedler eigenvector

③ Number of zero eigenvalues of L is equal to the number of disconnected components.



$$L = \begin{bmatrix} L_1 & & 0 \\ & L_2 & \\ 0 & & L_3 \end{bmatrix} = L_1 \oplus L_2 \oplus L_3$$

$n_1 + n_2 + n_3 = N$
 $n_1 \quad n_2 \quad n_3$

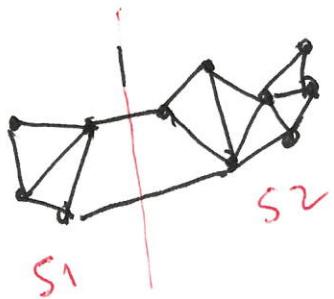
$$L \vec{1} = 0$$

$N \times N$

$$L_i \vec{1} = 0$$

$n_i \times 1$

Spectral clustering



Split the graph into K subgraphs with minimal cost but balanced cuts.

$$K=2$$

cost :

$$C = \frac{1}{2} \sum_{\substack{i \in S_1 \\ j \in S_2}} A_{ij}$$

$$s_i = \begin{cases} +1 & \text{if } i \in S_1 \\ -1 & \text{if } i \in S_2 \end{cases}$$

$$t_{ij} = \frac{1}{2}(1 - s_i s_j) = \begin{cases} 1 & i, j \text{ are in different clusters} \\ 0 & i, j \text{ are in the same cluster} \end{cases}$$

$$C = \frac{1}{2} \sum_{i,j} t_{ij} A_{ij} = \frac{1}{4} \left(\underbrace{\sum_{i,j} A_{ij}}_{\downarrow} - \sum_{i,j} A_{ij} s_i s_j \right)$$

Note that: $\sum_{i,j} A_{ij} = \sum_i d_i = \sum_i d_i s_i^2 = \sum_{i,j} d_i s_i s_j \delta_{ij}$

$$C = \frac{1}{4} \sum_{i,j} \left(\underbrace{d_i (\delta_{ij} - A_{ij})}_{L_{ij}} \right) s_i s_j$$

$$\min_{\vec{s}} C = \frac{1}{4} \vec{s}^T L \vec{s} \quad \vec{s} \in \mathbb{Z}^N$$

$s_i = \pm 1$

under these constraints:

$$\begin{cases} \textcircled{1} & \vec{s}^T \vec{s} = N = n_1 + n_2 \\ \textcircled{2} & \vec{s}^T \vec{1} = n_1 - n_2 \end{cases}$$

Relaxation : Solve the above

for $\vec{s} \in \mathbb{R}^N$ under ① & ②

$$\nabla_{\vec{s}} \left[\vec{s}^T L \vec{s} + \lambda (N - \vec{s}^T \vec{s}) + 2\mu (n_1 - n_2 - \vec{s}^T \vec{1}) \right]$$

$$\nabla_{\vec{s}} [\dots] \Big|_{\vec{s}^*} = 0 \quad \text{At } \vec{s}^* :$$

$$L \vec{s}^* = \lambda \vec{s}^* + \mu \vec{1}$$

$$L \vec{1} = 0$$

$$\begin{matrix} \vec{1}^T L \vec{s}^* \\ \parallel \\ 0 \end{matrix} \Rightarrow \underbrace{\vec{1}^T \vec{s}^*}_{(n_1 - n_2)} + \mu \underbrace{\vec{1}^T \vec{1}}_N$$

$$\frac{\mu}{2} = -\frac{(n_1 - n_2)}{N}$$

$$L \vec{s}^* = \lambda \left(\vec{s}^* + \frac{\mu}{2} \vec{1} \right)$$

$$\begin{matrix} \parallel \\ L \left(\vec{s}^* + \frac{\mu}{2} \vec{1} \right) \\ \vec{v} \end{matrix}$$

$$\vec{v} = \vec{s}^* + \frac{\mu}{2} \vec{1}$$

$$L \vec{v} = \lambda \vec{v}$$

eigenvalue of L
 eigenvector of daphnia

$$C = \frac{1}{4} \vec{s}^* L \vec{s}^* = \frac{1}{4} \vec{v}^T L \vec{v} = \frac{1}{4} \lambda_2 \vec{v}^T \vec{v}$$

\vec{v}_2 associated with λ_2

Plugging in you get

$$\underline{C_{\min} = \lambda_2 \frac{n_1 n_2}{N}}$$

This vector \vec{v}_2 is called the Fiedler vector and indicates an optimal bipartition.

The quality of the bipartition is given by the eigenvalue λ_2 , the algebraic connectivity of the graph.

If $\lambda_2 \ll \cancel{\text{positive number}}$
then there exists a good bipartition
of the graph given by \vec{v}_2 .

Spectral partitioning:

Given $L = V \Lambda V^T$

Find a clustering into k groups.

① Take:

$$V_k = \begin{bmatrix} \vec{v}_1 & \dots & \vec{v}_k \end{bmatrix}_{N \times k} \xrightarrow{\text{row}} \vec{w}_i \text{ (row)}$$

$\vec{w}_i \in \mathbb{R}^{k \times 1}$

②

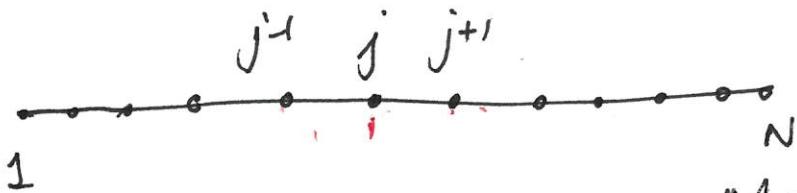
Take the rows of $V_k(i)$ as
descriptors of each node.

③

Carry out k -means on the vectors

$$\{\vec{w}_i\}_{i=1}^N \quad \vec{w}_i \in \mathbb{R}^k$$

Connection of graphs with random walks.



Consider discretization
of a 1D domain

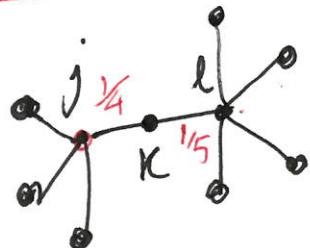
Discrete-time Markov chain

Defined in terms of: $(\vec{P}_t)_{N \times 1}$ = each coordinate containing the probability of the random walker at node i and time t .

$$\vec{1}^T \vec{P}_t = 1. \quad (\text{Normalized})$$

Random walk
on:

$$P_{t+1}^{(j)} = \frac{1}{2} (P_t^{(j-1)} + P_t^{(j+1)})$$



$$P_{t+1}^{(k)} = \frac{1}{5} P_t^{(l)} + \frac{1}{4} P_t^{(j)}$$

This applies generally to any graph:

For a given graph with adjacency matrix A we have a Discrete-time Markov chain:

$$\vec{P}_{t+1} = \underbrace{(AD^{-1})}_{M} \vec{P}_t = M \vec{P}_t$$

Transition matrix

Solution: $\vec{P}_t = M^t \vec{P}_0$

Associated continuous-time process:

$$\frac{d\vec{P}(t)}{dt} = - \underbrace{(I - AD^{-1})}_{L_{RW}} \vec{P}(t)$$

$$L_{RW} D = L = D - A$$

$$\underline{L_{RW} = L D^{-1}}$$

And the isospectral (with L_{RW})

Symmetrized normalized Laplacian:

$$\tilde{L} = \tilde{D}^{-\frac{1}{2}} L_{RW} \tilde{D}^{\frac{1}{2}} = \tilde{D}^{-\frac{1}{2}} L \tilde{D}^{-\frac{1}{2}}$$

$\tilde{L} = \tilde{L}^T$ ✓

See
'spectral,
clustering' section
above.

→ Similar algorithms based on
eigenvectors of \tilde{L} .

(Ng & Jordan)

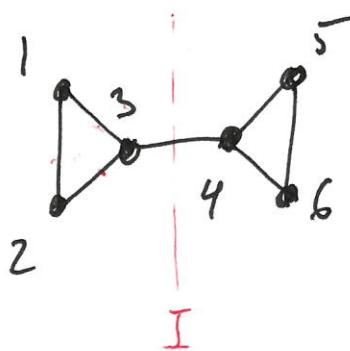
Note: They normalize by row too!

② Alternative to spectral clustering:
Modularity. (Newman)

Find a partition of the graph that
will have maximally block diagonal
structure in the adjacency matrix,
and has more 'blocks' than
expected at random.

Idea behind modularity: Count edges within blocks and between blocks

Example:



$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

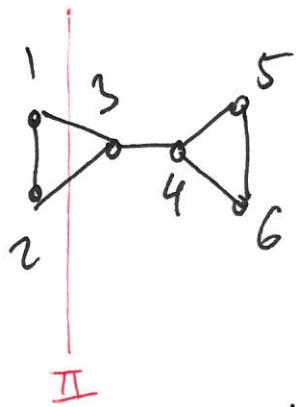
$$H_I = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}_{N \times K}$$

Into \underline{c} groups

$$\underline{c=2}$$

$$\frac{1}{2} \begin{pmatrix} H^T & A & H \\ I & I & c \times c \end{pmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} A \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} =$$

$$= \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 2 & 0 \\ 2 & 1 \\ 1 & 2 \\ 0 & 2 \\ 0 & 2 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 6 & 1 \\ 1 & 6 \end{bmatrix} = \begin{bmatrix} 3 & \frac{1}{2} \\ \frac{1}{2} & 3 \end{bmatrix}$$



Same A

$$H_{\text{II}} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

$$\frac{1}{2} (H_{\text{II}}^T A H_{\text{II}}) = \begin{bmatrix} 1 & 1 \\ 1 & 4 \end{bmatrix}$$

Maximize $\underset{H}{\text{Tr}} \left[\frac{1}{2} (H^T A H) \right]$

Second ingredient : Null model.

Configuration model.

$$\begin{array}{c} r \quad i \quad j \quad s \\ \diagdown \quad \diagup \\ l \quad \quad \quad \end{array} \quad \bar{d} = \begin{pmatrix} d_1 \\ \vdots \\ d_N \end{pmatrix} \quad 2E = \sum_{i=1}^N d_i$$

$$R_{ij} = \frac{d_i \cdot d_j}{2E}$$

Discount the edges that are expected to be present at random.

$$R = \frac{1}{2E} \vec{d} \vec{d}^T$$

(Expected edges
if we
rewire at random)

where $\vec{d} = A\vec{1}$

Construct the modularity matrix: Z

$$\frac{1}{2E} \text{Tr} \left[H^T \left[A - \frac{1}{2E} \vec{d} \vec{d}^T \right] H \right] = Q$$

and try to make it as modular as possible:
Modularity optimization:

$$\underset{H}{\text{Max}} \quad Q$$

How is Q optimised:

① Similarly to spectral clustering, it can be shown that we can use the leading eigenvector of Z to maximise Q .

Relaxation
to
 $\vec{s} \in \mathbb{R}^N$, etc

Then we can effect bipartitions in a recurrent manner until Q does not increase.

stopping criterion: $\Delta Q < 0$ in the iteration.

② In reality modularity is optimised through a greedy agglomerative algorithm that performs better than the relaxation on the eigenvectors of \mathbb{Z} .

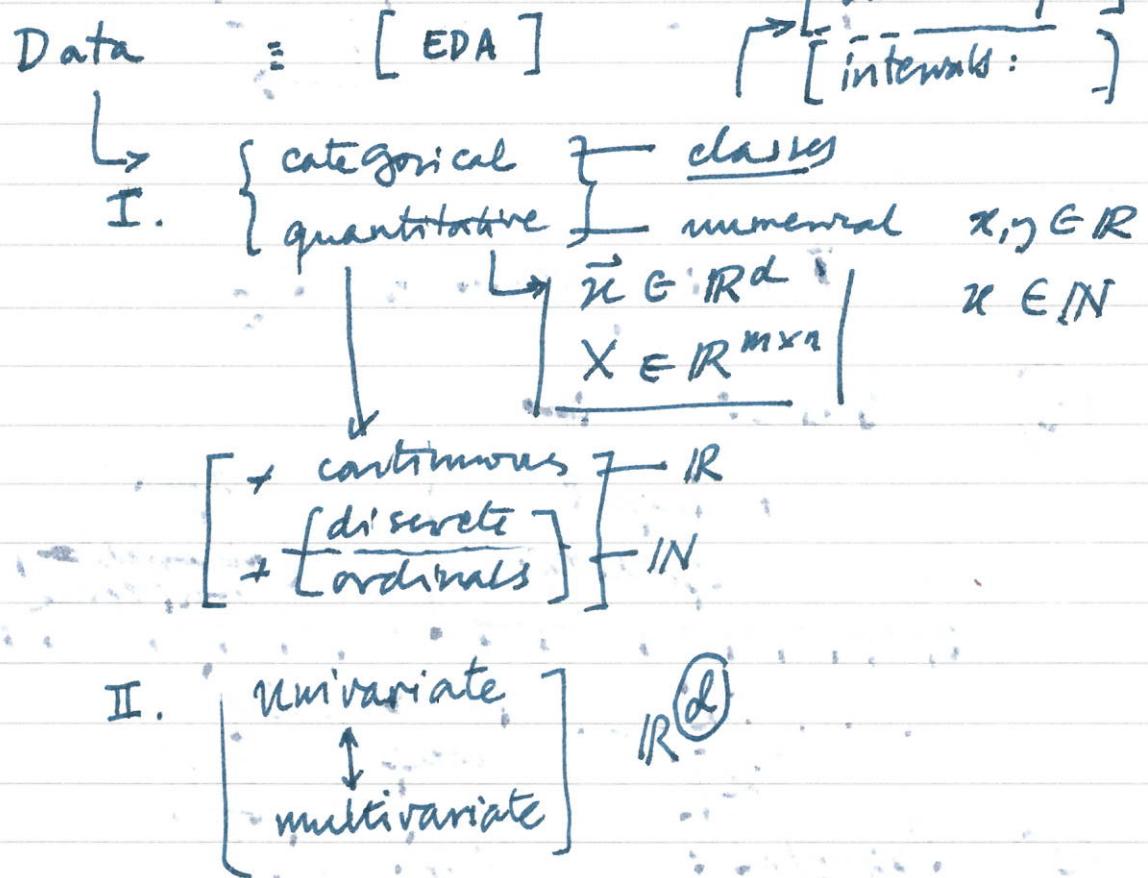
spectral

This method is called Louvain optimisation and has become an industry standard. It was proposed by Blondel et al in 2008.

6 Oct 2019

E/N

Lecture 2



III.

Predictor \Rightarrow Outcome

dependency: [independent] \Rightarrow [dependent]

[control]
 [input] \Rightarrow [observable]
 [output]

Both can share:
 randomness
 Int

$$x + \varepsilon_{in} \rightsquigarrow f(x + \varepsilon_{in}) + \delta_{out}$$

Learning: from input \longrightarrow output

Brief summary for supervised learning

given data \rightarrow EDA (clean-up)

\downarrow
decide on variables

+
task

\rightarrow Declare predictor space ; outcome space
 X y

\rightarrow $\left[\begin{array}{l} N \text{ noisy samples to be used for learning.} \\ \text{to learn.} \end{array} \right] \left(\begin{array}{l} (x_i, y_i) \\ x_i \in X \\ y_i \in Y \end{array} \right) \quad i=1, \dots, N$

In Supervised learning ~~means that~~
this is the training set.

keep some samples for validation.

For a given task with training set ~~set~~
 $\{(x_i, y_i)\}_{i=1}^N$

\Rightarrow Find function

$$f : X \rightarrow Y$$

and define

loss function $L(f(x), y)$
such that

(i) in-sample loss (err) is minimized

$$\mathbb{E} [L(f(\{x_i\}), \{y_i\})] \ll$$

and $\{(x_i, y_i)\}_{i \in \text{training}}$

(ii) expected out-of-sample loss is also small

$$\mathbb{E} [L(f(\{x_k\}), \{y_k\})] \ll$$

$\{(x_k, y_k)\}_{k \in \text{test}}$
(validation)

we expect then that
 \downarrow

$f(x_{\text{unknown}})$ will be a good predictor for y_{unknown}

Linear regression

Data:

Inputs
(predictors)

X

Output
(outcome)

y

Quantitative

Samples:

$i = 1, \dots, N$

samples
(observations)

$$x_1^{(1)} \ x_2^{(1)} \ \dots \ x_p^{(1)} \quad | \quad y^{(1)}$$

$$x_1^{(2)} \ x_2^{(2)} \ \dots \ x_p^{(2)} \quad | \quad y^{(2)}$$

⋮

$$x_1^{(N)} \ x_2^{(N)} \ \dots \ x_p^{(N)} \quad | \quad y^{(N)}$$

$$\text{Inputs: } \{\vec{x}^{(i)}\}_{i=1}^N \quad \text{output: } \{y^{(i)}\}_{i=1}^N$$

Assume a linear relationship: $f(\vec{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_p x_p$

$$\begin{cases} \vec{x} \in \mathbb{R}^p \\ y \in \mathbb{R} \end{cases}$$

$$\hat{y} = f_{LR}(\vec{x}; \vec{\theta})$$

$$\vec{\theta} = (\beta_0, \dots, \beta_p)$$

↑ Hypo/Parameters.

Defines the model

Need to find $\vec{\theta}$



$$f: X \rightarrow Y$$

$$f: \mathbb{R}^p \rightarrow \mathbb{R}$$

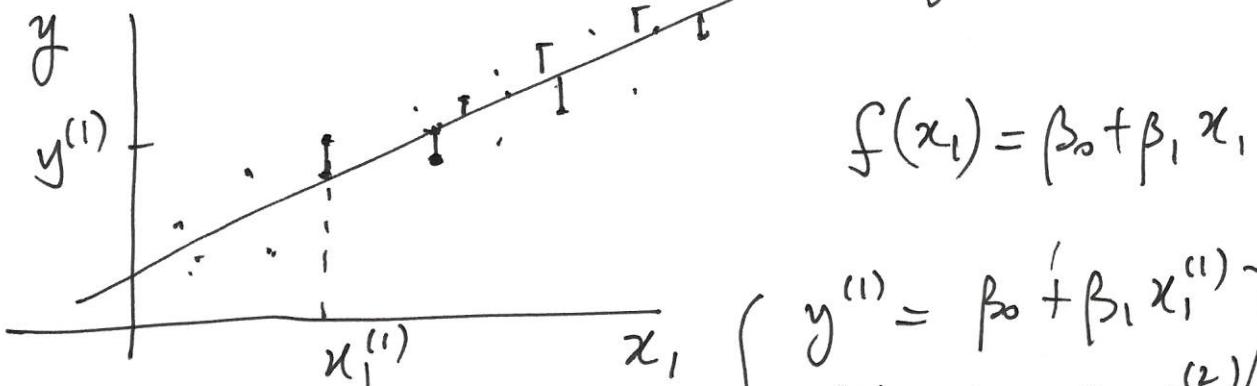
Simplest case (so that we can draw):

Assume

$$\underline{p=1}$$

$$\underline{N \gg 2}$$

$$f(x_i)(x_i^{(i)}, y^{(i)}) \quad i=1, \dots, N$$



$$f(x_i) = \beta_0 + \beta_1 x_i$$

Mean Square error :

MSE

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N \left[y^{(i)} - f_{LR}(x_i; \beta_0, \beta_1) \right]^2 = \frac{1}{N} \sum_{i=1}^N e^{(i)}^2$$

$$L(y, f_{LR}(x_i))$$

$$\text{Define: } \vec{y}^\top = (y^{(1)}, \dots, y^{(N)}) \in \mathbb{R}^N$$

$$\vec{x}_1^\top = (x_1^{(1)}, \dots, x_1^{(N)}) \in \mathbb{R}^N$$

$$\vec{1}^\top = (1, \dots, 1) \in \mathbb{R}^N$$

$$f_{LR}(\vec{x}) = \beta_0 \vec{1} + \beta_1 \vec{x}_1 = \begin{pmatrix} \vec{1} & \vec{x}_1 \end{pmatrix}_{N \times 2} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}_{2 \times 1}$$

$$f_{LR}(\vec{x}) = \vec{X} \cdot \vec{\beta} \quad (\vec{\beta} = \vec{\theta}) \quad \text{Reminder}$$

$$\vec{e} = \vec{y} - \vec{X}\vec{\beta}$$

$$L[f_{LR}(\vec{x}_i), y] = \frac{1}{N} \vec{e}_i^T \vec{e}_i = \frac{\|\vec{e}\|^2}{N}$$

Mimimize L in the space of parameter
 $\vec{\beta}$

$$L(\vec{\beta}) = L(\beta_0, \beta_1) = \frac{1}{N} [(\vec{y} - \vec{X}\vec{\beta})^T (\vec{y} - \vec{X}\vec{\beta})]$$

$$(1) \quad \left. \frac{dL}{d\vec{\beta}} \right|_{\vec{\beta}^*} = \nabla_{\vec{\beta}} L \Big|_{\vec{\beta}^*} = 0$$

$$(2) \text{ Check that } H(\vec{\beta}^*) = \left(\frac{\partial^2 L}{\partial \beta_i \partial \beta_j} \right) \cancel{\text{is positive definite}}$$

$$H > 0$$

$$L(\vec{\beta}) = \frac{1}{N} [\vec{y}^T \vec{y} - \vec{y}^T X \vec{\beta} - \vec{\beta}^T X^T \vec{y} + \vec{\beta}^T X^T X \vec{\beta}]$$

$$\nabla_{\vec{\beta}} L (\vec{y}^T X \vec{\beta})$$

Aside

$$\nabla_{\vec{\beta}} (\vec{\alpha}^T \cdot \vec{\beta}) = \nabla_{\vec{\beta}} (\alpha_1 \beta_0 + \alpha_2 \beta_1) = \begin{pmatrix} \alpha_1 \\ \alpha_2 \end{pmatrix}$$

$$\nabla_{\vec{\beta}} (\vec{\beta}^T \cdot \vec{\alpha}) = \vec{\alpha}$$

$$\nabla_{\vec{\beta}} (\vec{\beta}^T A \vec{\beta}) = A \vec{\beta} + A^T \vec{\beta} = (A + A^T) \vec{\beta}$$

$$\begin{aligned} \nabla_{\vec{\beta}} L(\vec{\beta}) &= \frac{1}{N} [-X^T \vec{y} - X^T \vec{y} + (X^T X + (X^T X)^T) \vec{\beta}] \\ &= -\frac{2}{N} [X^T \vec{y} - X^T X \vec{\beta}] \end{aligned}$$

Normal equations.

$$\nabla_{\beta} L \Big|_{\beta^*} = 0 \quad \underbrace{x^T y = (x^T x) \beta^*}$$

$$X = \begin{bmatrix} \vec{1}; \vec{x}_1 \end{bmatrix}_{N \times 2} \text{ invertible}$$

$$\beta^* = (X^T X)^{-1} X^T \vec{y} \quad \underbrace{\qquad \qquad \qquad}_{}$$

Centralities in graphs

(1) Degree : $\vec{C}_d = \frac{\vec{d}}{2E} = \frac{\vec{A}\vec{1}}{2E}$

(2) Betweenness centrality :



(3) Closeness :

$$\vec{C}_c(i) = \frac{1}{\frac{1}{N} \sum_j d_{ij}}$$

(4) Eigenvector centrality :

$$\vec{C}_E \quad \vec{C}_E(i) = \alpha \sum_{j \neq i} A_{ij} \vec{C}_E(j)$$

$$A \vec{C}_E = \lambda_1 \vec{C}_E$$

(5) Page Rank .

$$\vec{c}_{PR\ t+1} = \alpha (A\bar{D}^{-1})^T \vec{c}_{PR\ t} + (1-\alpha) \frac{\vec{1}}{N}$$

$$\alpha < 1$$

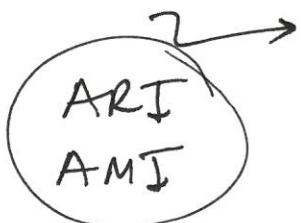
$$\underline{\alpha = 0.85}$$

PageRank is the stationary eigenvector
of this process:

$$\vec{c}_{PR}^* = \alpha (A\bar{D}^{-1})^T \vec{c}_{PR}^* + (1-\alpha) \frac{\vec{1}}{N}$$

→ Comparing clusterings :

- Extension of measures we considered when looking at confusion matrices to multiclass clusterings with unequal numbers of clusters.



Comparing clusterings: ARI

The contingency table [\[edit\]](#)

Given a set S of n elements, and two groupings or partitions (e.g. clusterings) of these elements, namely $X = \{X_1, X_2, \dots, X_r\}$ and $Y = \{Y_1, Y_2, \dots, Y_s\}$, the overlap between X and Y can be summarized in a contingency table $[n_{ij}]$ where each entry n_{ij} denotes the number of objects in common between X_i and Y_j : $n_{ij} = |X_i \cap Y_j|$.

$X \setminus Y$	Y_1	Y_2	\dots	Y_s	Sums
X_1	n_{11}	n_{12}	\dots	n_{1s}	a_1
X_2	n_{21}	n_{22}	\dots	n_{2s}	a_2
\vdots	\vdots	\vdots	\ddots	\vdots	\vdots
X_r	n_{r1}	n_{r2}	\dots	n_{rs}	a_r
Sums	b_1	b_2	\dots	b_s	

Definition [\[edit\]](#)

The original Adjusted Rand Index using the Permutation Model is

$$\widetilde{ARI} = \frac{\sum_{ij} \binom{n_{ij}}{2} - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}}{\frac{1}{2} [\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}] - [\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}] / \binom{n}{2}}$$

where n_{ij}, a_i, b_j are values from the contingency table.

Comparing clusterings: AMI

$$MI(U, V) = \sum_{i=1}^R \sum_{j=1}^C P(i, j) \log \frac{P(i, j)}{P(i)P'(j)}$$

where $P(i, j)$ denotes the probability that a point belongs to both the cluster U_i in U and cluster V_j in V .

$$P(i, j) = \frac{|U_i \cap V_j|}{N}$$

Adjustment for chance

$$AMI(U, V) = \frac{MI(U, V) - E\{MI(U, V)\}}{\max\{H(U), H(V)\} - E\{MI(U, V)\}}.$$

where

$$H(U) = - \sum_{i=1}^R P(i) \log P(i)$$

$$E\{MI(U, V)\} = \sum_{i=1}^R \sum_{j=1}^C \sum_{n_{ij}=(a_i+b_j-N)^+}^{\min(a_i, b_j)} \frac{n_{ij}}{N} \log \left(\frac{N \cdot n_{ij}}{a_i b_j} \right) \times \\ \frac{a_i! b_j! (N-a_i)! (N-b_j)!}{N! n_{ij}! (a_i - n_{ij})! (b_j - n_{ij})! (N - a_i - b_j + n_{ij})!}$$

$$\nabla_{\vec{\beta}} L \Big|_{\vec{\beta}^*} = 0$$

$\vec{x}^T \vec{y} = (\vec{x}^T \vec{x}) \vec{\beta}^*$

$X = \begin{bmatrix} \vec{1} & \vec{x}_1 \end{bmatrix}_{N \times 2}$ invertible

$$\vec{\beta}^* = \begin{bmatrix} (X^T X)^{-1}_{2 \times 2} & X^T_{2 \times N} \end{bmatrix} \vec{y}_{N \times 1}$$

$$(ii) H(L)_{ij} = \left(\frac{\partial^2 L}{\partial \beta_i \partial \beta_j} \right)$$

H is positive definite.

Aside::

$$\vec{\nabla}_{\vec{\beta}} \left(\vec{f}(\vec{\beta}) \right) = \begin{pmatrix} \vec{\nabla}_{\vec{\beta}} (f_1) & \dots & \vec{\nabla}_{\vec{\beta}} (f_m) \end{pmatrix}_{(p+1) \times m}$$

$$\vec{f} = \begin{pmatrix} f_1(\vec{\beta}) \\ \vdots \\ f_m \end{pmatrix}$$

$$H(L) = \vec{\nabla}_{\vec{\beta}} (\vec{\nabla}_{\vec{\beta}} L)$$

$$L = \frac{1}{N} [(\vec{y} - X\vec{\beta})^T (\vec{y} - X\vec{\beta})]$$

$$\vec{\nabla}_{\vec{\beta}} L = -\frac{2}{N} [X^T \vec{y} - (X^T X) \vec{\beta}]$$

$$H = +\frac{2}{N} \vec{\nabla}_{\vec{\beta}} ((X^T X) \vec{\beta})$$

Aside:

$$\vec{\nabla}_{\vec{\beta}} (A\vec{\beta}) = \vec{\nabla}_{\vec{\beta}} \left[\begin{array}{c} \vec{a}_1^T \cdot \vec{\beta} \\ \vdots \\ \vec{a}_m^T \cdot \vec{\beta} \end{array} \right] = \left[\vec{\nabla}_{\vec{\beta}} (\vec{a}_1^T \cdot \vec{\beta}) \dots \vec{\nabla}_{\vec{\beta}} (\vec{a}_m^T \cdot \vec{\beta}) \right]$$

$$A = \left[\begin{array}{c} \vec{a}_1^T \\ \vdots \\ \vec{a}_m^T \end{array} \right]$$

Reminder: $\vec{\nabla}_{\vec{\beta}} (\vec{x}^T \cdot \vec{\beta}) = \vec{x}$

$$\textcircled{1} \quad \vec{\nabla}_{\vec{\beta}} (A\vec{\beta}) = \left[\vec{a}_1 \dots \vec{a}_m \right]^T = A^T$$

Now back to H:

$$H = \frac{2}{N} \vec{\nabla}_{\vec{\beta}} \left((\vec{X}^T \vec{X}) \vec{\beta} \right) = \frac{2}{N} (\vec{X}^T \vec{X}) = \frac{2}{N} (\vec{X} \vec{X}^T)$$

Def: H is positive definite iff

$$\vec{z}^T H \vec{z} > 0 \quad \forall \vec{z} \neq 0$$

Show that $\vec{X}^T \vec{X}$ is positive definite:

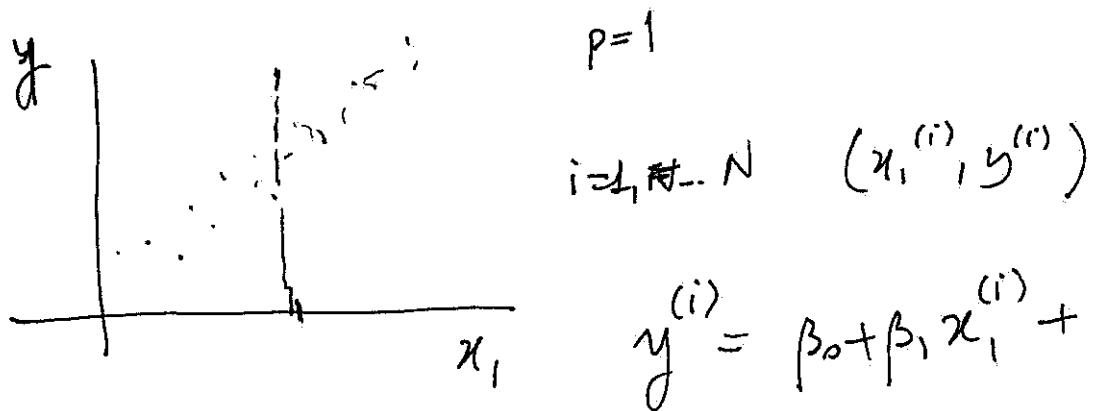
$$\vec{z}^T \vec{X}^T \vec{X} \vec{z} = (\vec{X} \vec{z})^T (\vec{X} \vec{z}) = \|\vec{X} \vec{z}\|^2 > 0$$

if $\vec{z} \neq 0$.

MSE is a convex
function in the space
of parameters



Statistical interpretation



$\epsilon^{(i)}$ are i.i.d. from
Normal distribution
with zero mean and variance σ^2

$$\epsilon \sim \mathcal{N}(0, \sigma^2)$$

$$\forall i \quad \text{Lik}(y^{(i)} | \vec{\beta}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y^{(i)} - (\beta_0 + \beta_1 x_1^{(i)}))^2}{2\sigma^2}}$$

$$\text{Lik}_{\text{tot}}(\vec{y} | \vec{\beta}) = \prod_{i=1}^N \text{Lik}(y^{(i)} | \vec{\beta})$$

$$\begin{aligned} \mathcal{L}_{\text{tot}} &= \sum_{i=1}^N \log(\text{Lik}(y^{(i)} | \vec{\beta})) = \\ &= C - \frac{1}{2\sigma^2} \sum_{i=1}^N \underbrace{(y^{(i)} - (\beta_0 + \beta_1 x_1^{(i)}))^2}_{e^{(i)}} \end{aligned}$$

$$= C - \frac{1}{2\sigma^2} \vec{e}^T \cdot \vec{e} = C - \frac{1}{2\sigma^2} N L_{\text{MSE}}$$

$$\underline{\mathcal{L}_{\text{tot}}} = C - \frac{N}{2\sigma^2} L_{\text{MSE}}$$

$$- \frac{\partial L_{\text{tot}}}{\partial \vec{\beta}} \longleftrightarrow \frac{\partial L_{\text{MSE}}}{\partial \vec{\beta}}$$

maximum likelihood

minimal loss (MSE)

Given $\{(x_i^{(i)}, y_i^{(i)})\}_{i=1}^N$
the linear model $\therefore f(x_i) = (1 x_i) \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix}$
 $= (1 x_i) \vec{\beta}$

minimises the loss

$$L = \frac{1}{N} \left[(\vec{y} - X\vec{\beta})^T (\vec{y} - X\vec{\beta}) \right]$$

$$\text{when } \vec{\beta}^* = \underbrace{(\vec{x}^T \vec{x})^{-1}}_{\vec{x}^+} \vec{x}^T \vec{y}$$

Moore-Penrose pseudoinverse
of X

$$p=1$$

$$\underset{N \times 2}{X} \vec{\beta} = \underset{N \times 1}{\vec{y}}$$

No solution!
(overdetermined)

$$\downarrow$$

$$X^T X \vec{\beta} = X^T \vec{y}$$

$$\vec{\beta} = (X^T X)^{-1} X^T \vec{y}$$

Some properties of X^+ :

$$X^+ X X^+ = X^+ \quad \longleftrightarrow \quad (\bar{A}^T A \bar{A}^{-1} = \bar{A}^{-1})$$

$$X X^+ X = X \quad \longleftrightarrow \quad (A \bar{A}^{-1} \bar{A} = A)$$

Equivalent properties for invertible A :

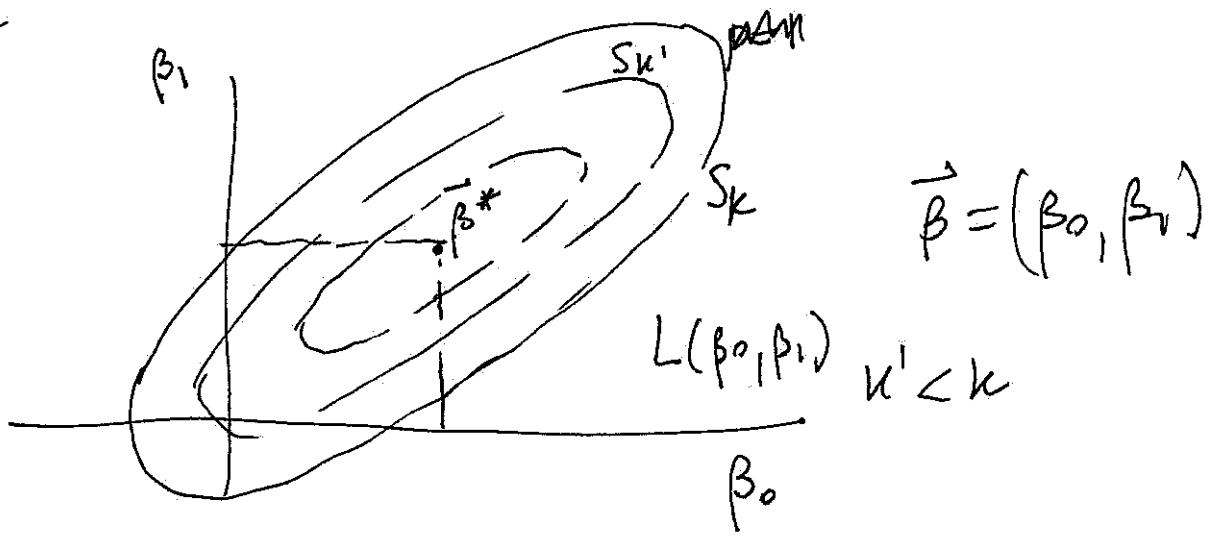
Optimisation = minimized

$$L(\vec{\beta}) = \frac{1}{N} ((\vec{y} - X \vec{\beta})^T (\vec{y} - X \vec{\beta}))$$

$$\frac{\partial L}{\partial \vec{\beta}} \Big|_{\vec{\beta}^*} = 0 \quad \textcircled{R}$$

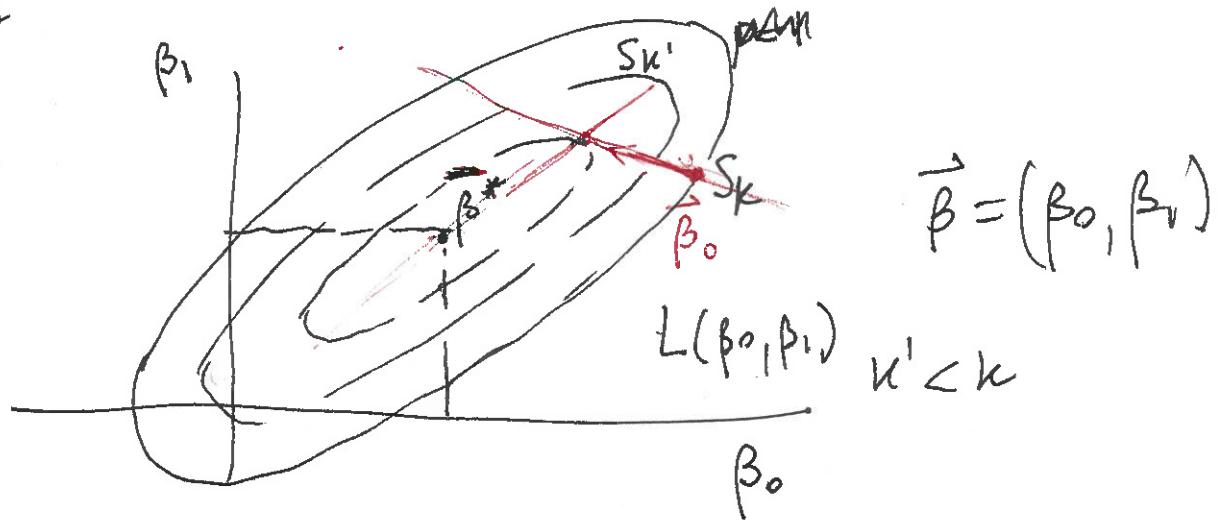
We will not be able to solve
for the normal equations. \textcircled{R}

$p=1$



$$S_k = \left\{ \vec{\beta} \mid L(\vec{\beta}) = k \right\}$$

$p=1$



$$S_k = \left\{ \vec{\beta} \mid L(\vec{\beta}) = k \right\}$$

$$L(\vec{\beta}) = \frac{1}{N} \left[(\vec{y} - X\vec{\beta})^T (\vec{y} - X\vec{\beta}) \right] \quad \textcircled{1}$$

↓
a couple of steps :

$$L(\vec{\beta}) = L(\vec{\beta}^*) + \frac{1}{2} (\vec{\beta} - \vec{\beta}^*)^T \underbrace{(\vec{\beta} - \vec{\beta}^*)}_{\frac{2}{N}(X^T X)}$$

$$\textcircled{2} \quad L(\vec{\beta}) = \frac{1}{N} \left[(\vec{y} - X\vec{\beta}^* + X\vec{\beta}^* - X\vec{\beta})^T (\vec{y} - X\vec{\beta}^* + X\vec{\beta}^* - X\vec{\beta}) \right] =$$

$$= \frac{1}{N} \left[(\vec{y} - X\vec{\beta}^*)^T (\vec{y} - X\vec{\beta}^*) + (\vec{y} - X\vec{\beta}^*)^T \times (\vec{\beta}^* - \vec{\beta}) + \right.$$

$$\left. + [X(\vec{\beta}^* - \vec{\beta})]^T (\vec{y} - X\vec{\beta}^*) + (\vec{\beta}^* - \vec{\beta})^T X^T \times (\vec{\beta}^* - \vec{\beta}) \right]$$

Normal equations: $X^T \vec{y} - X^T X \vec{\beta}^* = 0 \Rightarrow X^T [\vec{y} - X\vec{\beta}] = 0$

see next page

Convexity \Rightarrow local minimum is always a global minimum

This loss function can be minimized with gradient methods.

$\nabla_{\beta} L$ marks the direction of maximum change of L

Algorithm: Iteratively follow the direction of $-\nabla_{\beta} L$

1st order

$$\vec{\beta}_{k+1} = \vec{\beta}_k - \gamma \nabla_{\beta} L(\vec{\beta}_k)$$

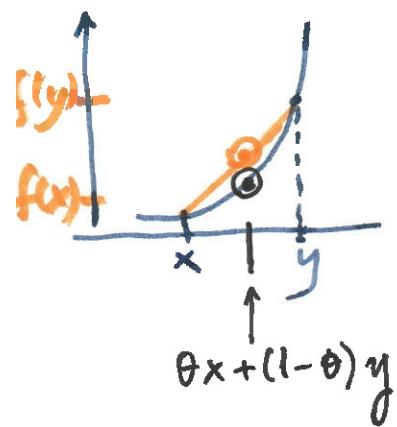
- Line search
 - Backtracking
 - Conjugate gradient
- Gradient methods.

Convexity :

$f: \mathbb{R}^n \rightarrow \mathbb{R}$ is convex if

+ $\vec{x}, \vec{y} \in \mathbb{R}^n$ and $\theta \in (0,1)$

$$\theta f(\vec{x}) + (1-\theta) f(\vec{y}) \geq f(\theta \vec{x} + (1-\theta) \vec{y})$$



If f is convex, a local minimum is always a global minimum

Let $\vec{x}_{\text{local}}, \vec{x}_{\text{global}} \in \mathbb{R}^n$, $f(\vec{x}_{\text{global}}) < f(\vec{x}_{\text{local}})$

By contradiction:

- If \vec{x}_{local} is a local minimum then:
 $f(\vec{x}_{\text{local}}) \leq f(\vec{x})$, $\|\vec{x} - \vec{x}_{\text{local}}\| < \delta$
- If f is convex,
 $\exists \theta$, $\theta \vec{x}_{\text{local}} + (1-\theta) \vec{x}_{\text{global}} = \vec{x}_\theta$
 $\|\vec{x}_\theta - \vec{x}_{\text{local}}\| < \delta$

But then : $\Rightarrow f(\vec{x}_{\text{local}}) \leq f(\vec{x}_\theta)$

$$\begin{aligned} &\leq \theta f(\vec{x}_{\text{local}}) + (1-\theta) f(\vec{x}_{\text{global}}) \\ &< \theta f(\vec{x}_{\text{local}}) + (1-\theta) f(\vec{x}_{\text{local}}) \\ &= f(\vec{x}_{\text{local}}) \end{aligned}$$

~~$f(\vec{x}_{\text{local}}) < f(\vec{x}_{\text{local}})$~~

Another way is using
Newton's method:

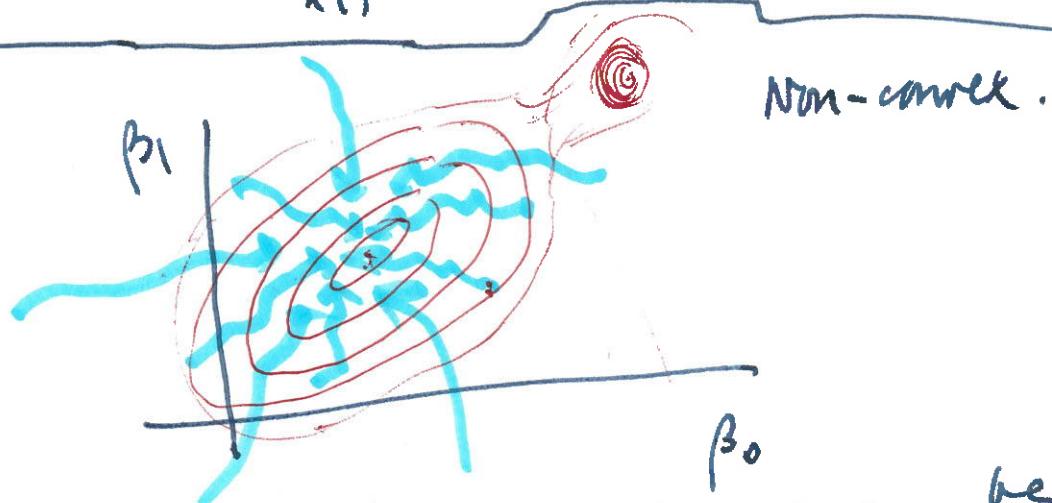
2nd order

$$L \approx L(\vec{\theta}_k) + \nabla L \Big|_{\vec{\theta}_k} (\vec{\theta} - \vec{\theta}_k) + \frac{1}{2} (\vec{\theta} - \vec{\theta}_k)^T H(\vec{\theta}_k) (\vec{\theta} - \vec{\theta}_k)$$

$$\nabla L \approx \nabla L \Big|_{\vec{\theta}_k} + H(\vec{\theta}_k) (\vec{\theta} - \vec{\theta}_k)$$

$$\nabla L(\vec{\theta}_k) + H(\vec{\theta}_k) (\vec{\theta}_{k+1} - \vec{\theta}_k) = 0$$

$$\vec{\theta}_{k+1} = \vec{\theta}_k - H(\vec{\theta}_k)^{-1} \nabla L(\vec{\theta}_k)$$



Global minimum can be
difficult to find

The same formulation applies for
 $p > 1$

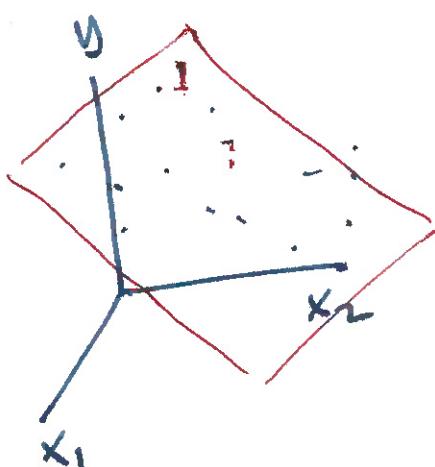
$$\left\{ \begin{array}{c} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_p^{(i)} \end{array}, \quad y^{(i)} \right\}_{i=1}^N$$

$$\beta_0 + \beta_1 x_1^{(i)} + \dots + \beta_p x_p^{(i)} = f(\vec{x}^{(i)}) \Rightarrow \vec{y}^{(i)}$$

$$\vec{\beta} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_p \end{pmatrix} \in \mathbb{R}^{(p+1) \times 1}$$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ 1 & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_p^{(n)} \end{bmatrix}_{N \times (p+1)}$$

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix}_{N \times 1}$$



$$\vec{X}\vec{\beta} = \vec{y} ? \quad \text{No solution}$$

$$\vec{e} = \vec{y} - \vec{X}\vec{\beta}$$

$$\vec{\beta}^*_{LS} = (\vec{X}^T \vec{X})^{-1} \vec{X}^T \vec{y}$$

linear regression model Least squares solution

$$\min_{\vec{\beta}} \|\vec{y} - \vec{X}\vec{\beta}\|^2$$

Bias vs Variance

Hastie, ESL, Chapter 3

$$\mathbb{E}[\vec{\beta}^*] = \mathbb{E}\left[(X^T X)^{-1} X^T \vec{y}\right] = \vec{\beta} + \mathbb{E}\left[(X^T X)^{-1} X^T \vec{\epsilon}\right] = \vec{\beta}$$
$$\vec{y} = X \vec{\beta} + \vec{\epsilon} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

Bias vs Variance . Section 3.2

Hastie, ESL, Chapter 3

$$\mathbb{E}[\vec{\beta}^*] = \mathbb{E}\left[(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{y}\right] = \vec{\beta} + \mathbb{E}\left[(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \vec{\epsilon}\right] = \vec{\beta}$$

$$\vec{y} = \mathbf{X} \vec{\beta} + \vec{\epsilon} \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

① Bias : $\|\mathbb{E}[\vec{\beta}^*] - \vec{\beta}\| = 0$

② $\mathbb{E}[(\vec{\beta} - \vec{\beta}^*)(\vec{\beta} - \vec{\beta}^*)^T] = \mathbb{E}\left[(\mathbf{X}^T \mathbf{X})^{-1} \underbrace{\mathbf{X}^T \vec{\epsilon} \vec{\epsilon}^T}_{\sigma^2 \mathbf{I}} (\mathbf{X}^T \mathbf{X})^{-1}\right]$

Variance of estimator

$$= (\mathbf{X}^T \mathbf{X})^{-1} \sigma^2$$

$$\hat{\sigma}^2 = \frac{1}{N-(p+1)} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$\mathbb{E}[\hat{\sigma}^2] = \sigma^2$$

For an estimator $\hat{\theta}^*$ of θ we have in general that:

$$\mathbb{E}[(\theta - \hat{\theta}^*)^2] = \mathbb{E}[\theta^2] + \mathbb{E}(\hat{\theta}^{*2}) - 2 \mathbb{E}[\theta \hat{\theta}^*] \stackrel{(1)}{=} \quad \theta \text{ is the true value to be estimated } (\Rightarrow \text{not a random variable})$$

$$\begin{aligned} & \stackrel{(1)}{=} \theta^2 + \underbrace{\text{var}(\hat{\theta}^*)}_{\substack{\uparrow \\ \text{variance of the estimator}}} + \underbrace{\mathbb{E}(\hat{\theta}^*)^2}_{\substack{\uparrow \\ \text{Bias}^2}} - 2 \theta \mathbb{E}(\hat{\theta}^*) \\ &= \text{var}(\hat{\theta}^*) + [\theta - \mathbb{E}(\hat{\theta}^*)]^2 \end{aligned}$$

How good is an estimator is a combination of as low bias and as low variance as possible.

Least squares is best unbiased linear estimator

Based on Gauss-Markov theorem:

Let ~~β_{LS}~~ β_{LS}^* define our LS estimator such that $\hat{f}_{LS}(\vec{x}_0) = \vec{x}_0^\top \vec{\beta}_{LS}^*$,

which is unbiased: $E[\hat{f}(\vec{x}_0)] = \vec{x}_0^\top \vec{\beta}$
where $\vec{\beta}$ is the true value,

Let \hat{f} be another linear estimator.

Gauss-Markov says:

$$\text{var}(\hat{f}_{LS}(\vec{x}_0)) \leq \text{var}(\hat{f}(\vec{x}_0))$$

As a consequence we have the following:

$$\text{MSE} = E[(\hat{f} - y)^2] = \text{var}(\hat{f}) + [y - E[\hat{f}]]^2$$

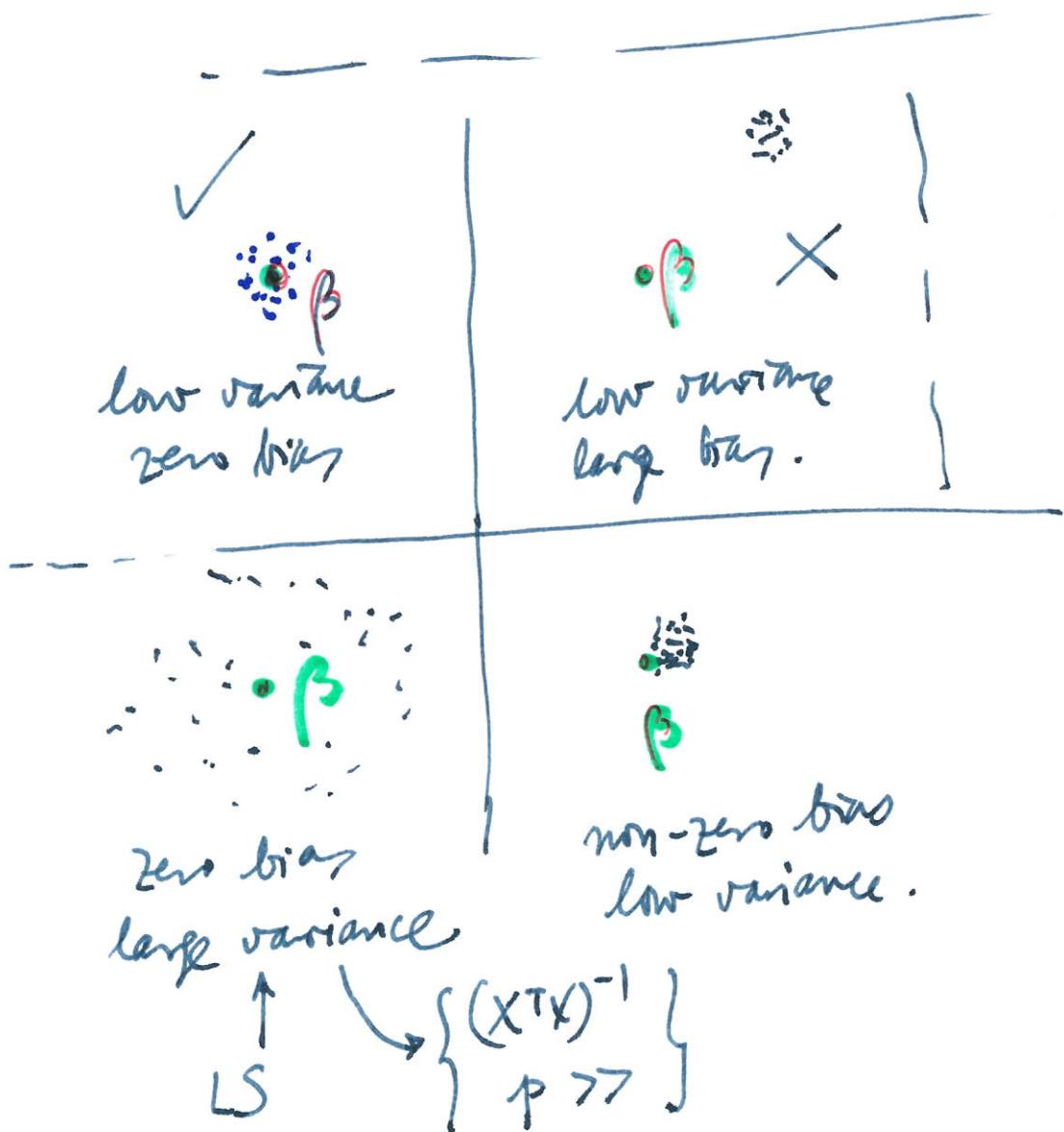
Since \Rightarrow LS has lowest MSE of all linear unbiased estimators

$$[\text{If bias}=0, \text{MSE} = \text{var}(\hat{f}), \text{etc}]$$

Quick set of pointers (sections 3.2, 3.3) ESL

↳ Gauss-Markov

If estimator is linear and unbiased,
one cannot do better than $\hat{\beta}_{LS}^*$



Motivated by reducing variance
and increasing interpretability
~~use~~ some methods attempt to
reduce the number of descriptors, p

- ① Find a subset of descriptors that are "good".

$$X = (\vec{x}_1, \dots, \vec{x}_p)$$

1.1. Find the best subset of the p parameters.

Decide the ~~size~~ size of the subset: $K =$
"Leaps and bounds"
works for up to $p \approx 50$

1.2 Sequential approaches.

Forward / Backward.

Forward: Add x_i one at a time choosing the descriptor that reduces the error maximally.

Backward: Starting from the full LS model with p parameters, chop off one by one picking the one that increases the error minimally.

Implemented through QR decomposition

Gram-Schmidt:

$$X = QR$$

$$Q^T Q = I$$

R upper triangular.

[Always project in orthogonal directions]

② Change the loss function

Remember our optimisation formulation:

Linear regression is least squares

given X, \vec{y} (our data), we find $\vec{\beta}^*$

$$\text{from } \min_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 = \min_{\vec{\beta}} L_{LS}(\vec{\beta})$$

The solution is:

$$\vec{\beta}^* = \arg \min L_{LS}(\vec{\beta}) = X^+ \vec{y} = \underbrace{(X^T X)^{-1} X^T}_{\vec{X}^T \vec{X}} \vec{y}$$

Let us introduce another estimators which will be biased but with potentially less variance.

2.1 Ridge regression:

Same as above but loss is:

$$L_{\text{ridge}}(\vec{\beta}) = \|\vec{y} - X\vec{\beta}\|^2 + \gamma \|\vec{\beta}\|^2$$

The ridge solution is obtained from:

$$\min_{\vec{\beta}} L_{\text{ridge}}(\vec{\beta}) = \min_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 + \gamma \|\vec{\beta}\|^2$$

which is equivalent to:

$$\begin{aligned} & \min_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 \\ & \text{subject to } \|\vec{\beta}\|^2 \leq t \end{aligned}$$

This solution can be obtained explicitly:

Make the size of the coefficients small \Rightarrow "a continuous version of sparsity".

Lecture 8

21 October 2019 08:59

With many thanks to Dr Giordano Scariotti

② SHRINKAGE METHODS:

LINEAR REGRESSION (LS): Given X, \vec{y} one finds $\vec{\beta}^*$

$$\min_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 = \min_{\vec{\beta}} L_{LS}(\vec{\beta}) \quad (X^\top X)^{-1} X^\top$$

THE SOLUTION IS:

$$\vec{\beta}^* = \text{ARGMIN}_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 = X^\top \vec{y}$$

$$\hat{y} = \hat{f}(x_w) = \vec{x}_w \vec{\beta}^*$$

2.1 RIDGE REGRESSION

$$L_{RIDGE}(\vec{\beta}) \|\vec{y} - X\vec{\beta}\|^2 + \lambda \|\vec{\beta}\|^2 \quad (\text{THE IDEA IS TO "ELIMINATE" some DESCRIPTORS. IN PRACTICE WE WEIGHT THEM LOW})$$

$$\min_{\vec{\beta}} L_{RIDGE}(\vec{\beta}) = \min_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 + \lambda \|\vec{\beta}\|^2 \quad ||| \text{ EQUIVALENT } \quad \text{Penalty Term}$$

$$\begin{cases} \min_{\vec{\beta}} \|\vec{y} - X\vec{\beta}\|^2 \\ \text{SUBJECT TO} \quad \|\vec{\beta}\|^2 \leq t \end{cases}$$

λ and t are inversely related

THIS PROBLEM CAN BE SOLVED EXPLICITLY:

$$L_{RIDGE}(\vec{\beta}) = \vec{y}^\top \vec{y} - \vec{\beta}^\top X^\top \vec{y} - \vec{y}^\top X \vec{\beta} + \vec{\beta}^\top (X^\top X + \lambda I) \vec{\beta}$$

$$\nabla_{\vec{\beta}} L_{RIDGE} = -2 X^\top \vec{y} + 2 (X^\top X + \lambda I) \vec{\beta}$$

$$X^\top \vec{y} = (X^\top X + \lambda I) \vec{\beta}^*$$

$$\vec{\beta}_{RIDGE}^* = (X^\top X + \lambda I)^{-1} X^\top \vec{y} \quad \text{WE CAN CHECK THAT THE HESSIAN IS POSITIVE DEFINITE}$$

$$\text{BIAS: } \vec{y} = X\vec{\beta} + \vec{\epsilon} \quad E[\vec{\epsilon}] = 0$$

$$E[\vec{\beta}_{RIDGE}^*] = E[(X^\top X + \lambda I)^{-1} (X^\top X) \vec{\beta} + (X^\top X + \lambda I)^{-1} X^\top \vec{\epsilon}] = (X^\top X + \lambda I)^{-1} (X^\top X) \vec{\beta}$$

$$(X^\top X) = V D V^\top \quad VV^\top = V^\top V = I \quad (X^\top X)^{-1} = V D^{-1} V^\top \quad \begin{array}{c} \text{D DIAGONAL} \\ \sigma(D) = \sigma(X^\top X) \end{array}$$

V contains the eigenvectors as columns

$D = \text{diag}(d_i)$ has the eigenvalues on the diagonal.

$$\text{BIAS} = E[\vec{\beta}_{\text{Ridge}}] - \vec{\beta} = (X^T X + \lambda I)^{-1} X^T \vec{\beta} = V[(D + \lambda I)^{-1} D - I] V^T \vec{\beta}$$

$$D = (D + \lambda I)^{-1} D - I \quad \leftarrow \text{EVERYTHING IS DIAGONAL}$$

$$D_{ii} = \left[\frac{d_i}{d_i + \lambda} - 1 \right] = -\frac{\lambda}{d_i + \lambda}$$

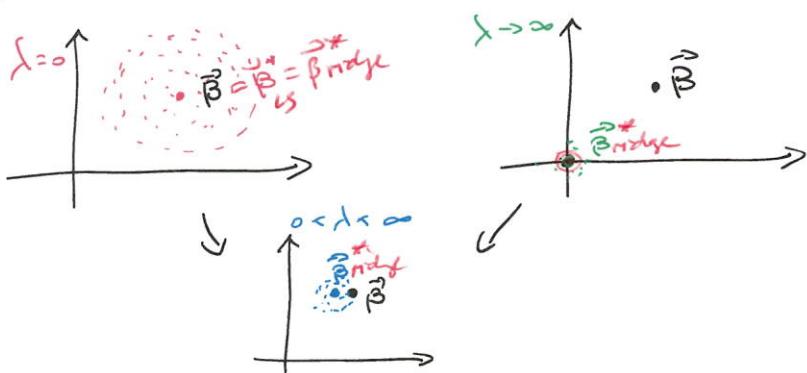
$$\Rightarrow \text{BIAS} = -\lambda V[(D + \lambda I)^{-1}] V^T \vec{\beta} = -\lambda (X^T X + \lambda I)^{-1} \vec{\beta}$$

[As expected as $\lambda \rightarrow 0$ BIAS $\rightarrow 0$ because we recover least squares
 As $\lambda \rightarrow \infty$ BIAS $\rightarrow -\vec{\beta}$ ($\approx 100\%$ error)]

Variance:

$$\begin{aligned} \text{VAR}(\vec{\beta}_{\text{Ridge}}) &= E[(\vec{\beta}_{\text{Ridge}} - E[\vec{\beta}_{\text{Ridge}}])^2] \\ &= E[(X^T X + \lambda I)^{-1} X^T \vec{\varepsilon} \vec{\varepsilon} X (X^T X + \lambda I)^{-1}] = \\ &= \sigma^2 (X^T X + \lambda I)^{-1} (X^T X) (X^T X + \lambda I)^{-1} = \\ &= \sigma^2 V \underbrace{[(D + \lambda I)^{-1} D (D + \lambda I)^{-1}]}_{\Phi} V^T \quad P_{ii} = \frac{d_i}{(d_i + \lambda)^2} \end{aligned}$$

As $\lambda \rightarrow \infty$ $P_{ii} \rightarrow 0$ (QUADRATICALLY)



2.2 LASSO (TIBSHIRANI)

$$L_{\text{Lasso}}(\vec{\beta}) = \|\vec{y} - X \vec{\beta}\|^2 + \lambda \|\vec{\beta}\|_1$$

$$\|\vec{\beta}\|_1 = \sum_{i=1}^p |\beta_i|$$

$$\text{min } L_{\text{Lasso}}(\vec{\beta}) \iff \text{min } \|\vec{y} - X \vec{\beta}\|^2 \text{ SUBJECT TO } \|\vec{\beta}\|_1 \leq t$$

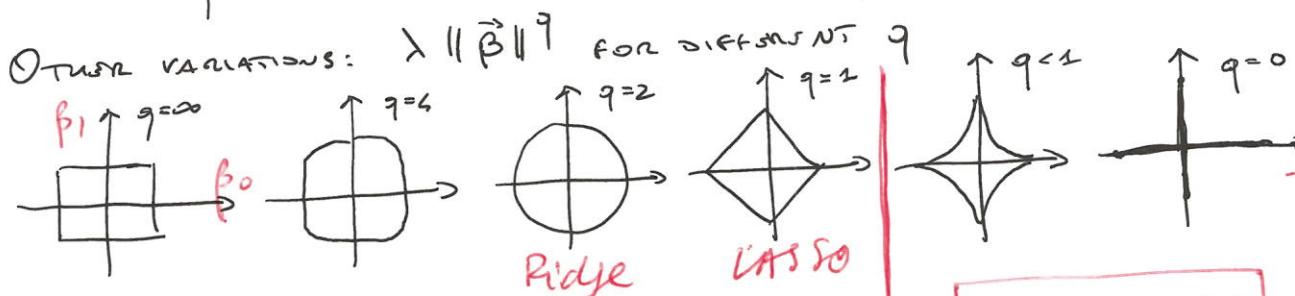
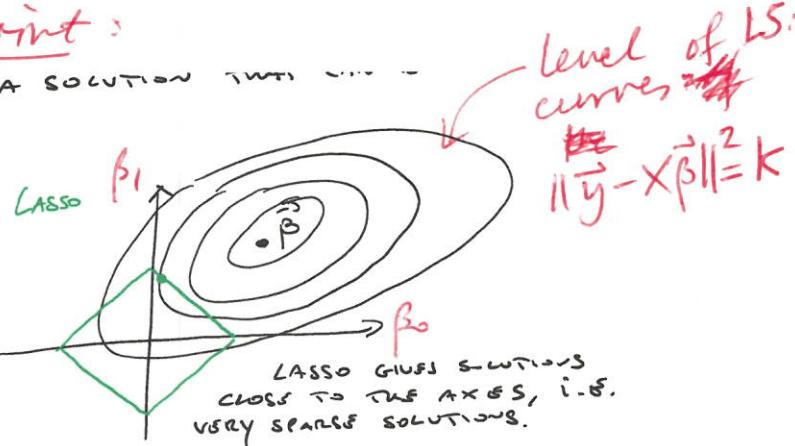
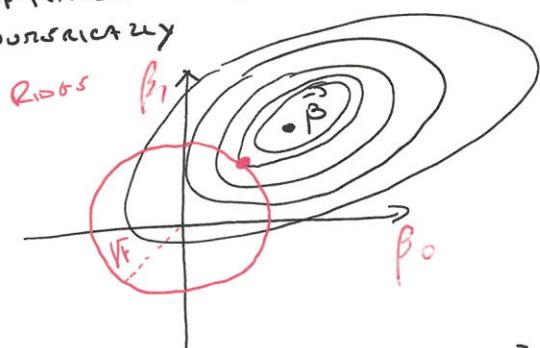
~~WE CAN'T FIND AN ANALYTICAL SOLUTION BUT THIS IS A CONVEX FUNCTION~~

No analytical solution for LASSO (contrary to Ridge)
 But it is a convex problem [optimize a convex function over a convex set]

So we can use convex optimisation techniques (quadratic programming) to optimise globally.

optimization viewpoint:

We can solve an optimisation problem that has a solution numerically



convex sets

Non-convex sets

optimization is doable

optimization is difficult

$q=0$ is the " ℓ_0 "-pseudonorm case

this whole area is called regularisation

or controlling for model complexity

where we only have solutions where some of the parameters β_i are zero = Subset selection

i.e., "Sparse" models

Difficult to optimise for sparsity

Another direction is to mix the penalty terms in objective function:

Example:

Elastic net:

$$L_{EN}(\vec{\beta}) = \|\vec{y} - X\vec{\beta}\|_2^2 + \lambda [\alpha \|\vec{\beta}\|_1 + (1-\alpha) \|\vec{\beta}\|_2]$$

Introducing non-linearity

$$\text{Linear} = LS, Ridge... = \hat{f}_{\text{lin}}(\vec{x}) = \vec{x}^T \vec{\beta}^*$$

$$\text{Non linear} = \hat{f}_{\text{nonlin}}(\vec{x}) = \underbrace{\vec{h}_m(\vec{x})}_{\text{feature}} \cdot \vec{\beta}^*$$

Example: $\{h_{m,1}, \dots, h_{m,T}\}$ from $(x_1^{d_1}, x_2^{d_2}, \dots, x_p^{d_p})$

$$\frac{(P+D)}{D}$$

Sparsity is desirable.

Others: (i) log(x_i) --

$$(ii) \sin(x_i) \cos(x_i) \quad \sin(kx_i)$$

Global vs. local models

$$\left. \begin{array}{l}
 \text{Global} \\
 \text{Local}
 \end{array} \right\} \quad \begin{array}{l}
 \text{Global nodes} \\
 \text{Local nodes}
 \end{array} \quad \left. \begin{array}{l}
 \text{Linear model:} \\
 \text{Nonlinear model:}
 \end{array} \right\} \quad i = 1, \dots, N.$$

$\hat{y} = \hat{f}_{\text{lin}}(\vec{x}^{in})$ $\vec{x}^{in} = (x_1, \dots, x_p)$
 $= \vec{x}^{in} \cdot \vec{\beta}^*$

 $\hat{y} = \hat{f}_{\text{nonlin}}(\vec{x}^{in})$
 $= h(\vec{x}^{in})^\top \cdot \vec{\beta}_h$

Alternative is to try and describe the data "locally" (a piece-wise)

We aim to not find a model f but for a model that is different depending on x^{in}

$$\left\{ \hat{f}_j(\vec{x}^{in}) \right\} \rightarrow \hat{y} = \hat{f}_j(\vec{x}^{in}) (\vec{x}^{in})$$

For continuous variable: $\vec{x}^{(i)} \in \mathbb{R}^P, y \in \mathbb{R}$
 $i=1, \dots, N$

Introduce a metric in the space of inputs:

$$\|\vec{x}^{(i)} - \vec{x}^{(j)}\|$$

Distance is a choice

Given an input \vec{x}^{in} :

compute all distances between \vec{x}^{in} and the samples.

$$\|\vec{x}^{(im)} - \vec{x}^{(i)}\| \quad \forall i=1, \dots, N$$

- ② Find the K-nearest neighbors which defines
 the ~~nearest~~ neighborhood

$$N_K(\vec{x}^{in})$$

K is a choice

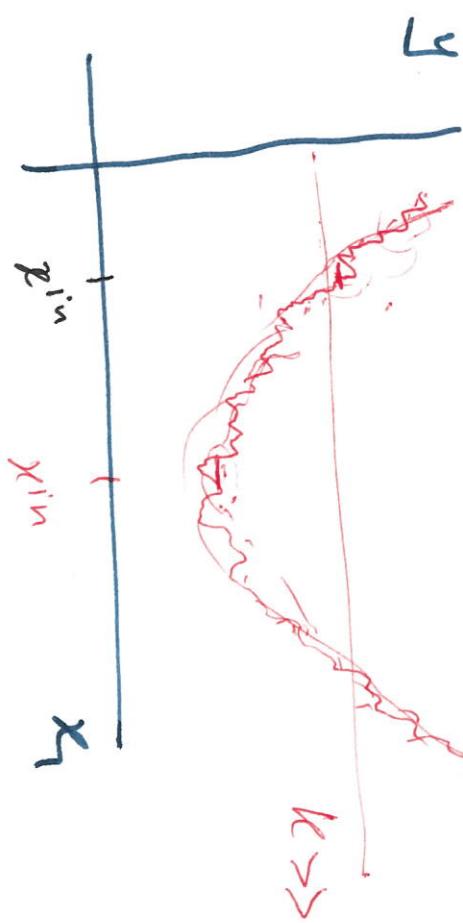
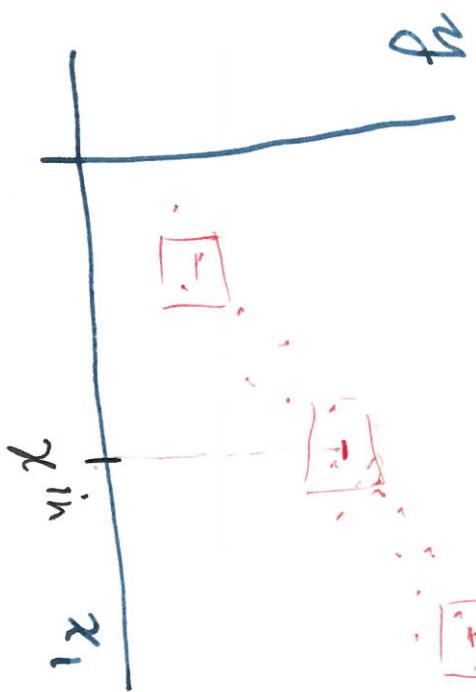
$K - NN = K$ nearest neighbors

(3)

Simplest choice for predictor:

$$\hat{f}(\vec{x}^{in}) = \frac{1}{K} \sum_{i \in K} y^{(i)}$$

K small



The outcome depend on our choices:

(1) Distance and normalization matter a lot.

$$x_1 \dots x_p$$

$1 \lesssim x_i < 10^6 \quad 10^{-10} < x_p < 10^{-5}$

(2) It works better for small, relevant P .

(3) K matters a lot.
↳ allows to scan the bias-variance trade off.