Learning from Observations

ENCS3340

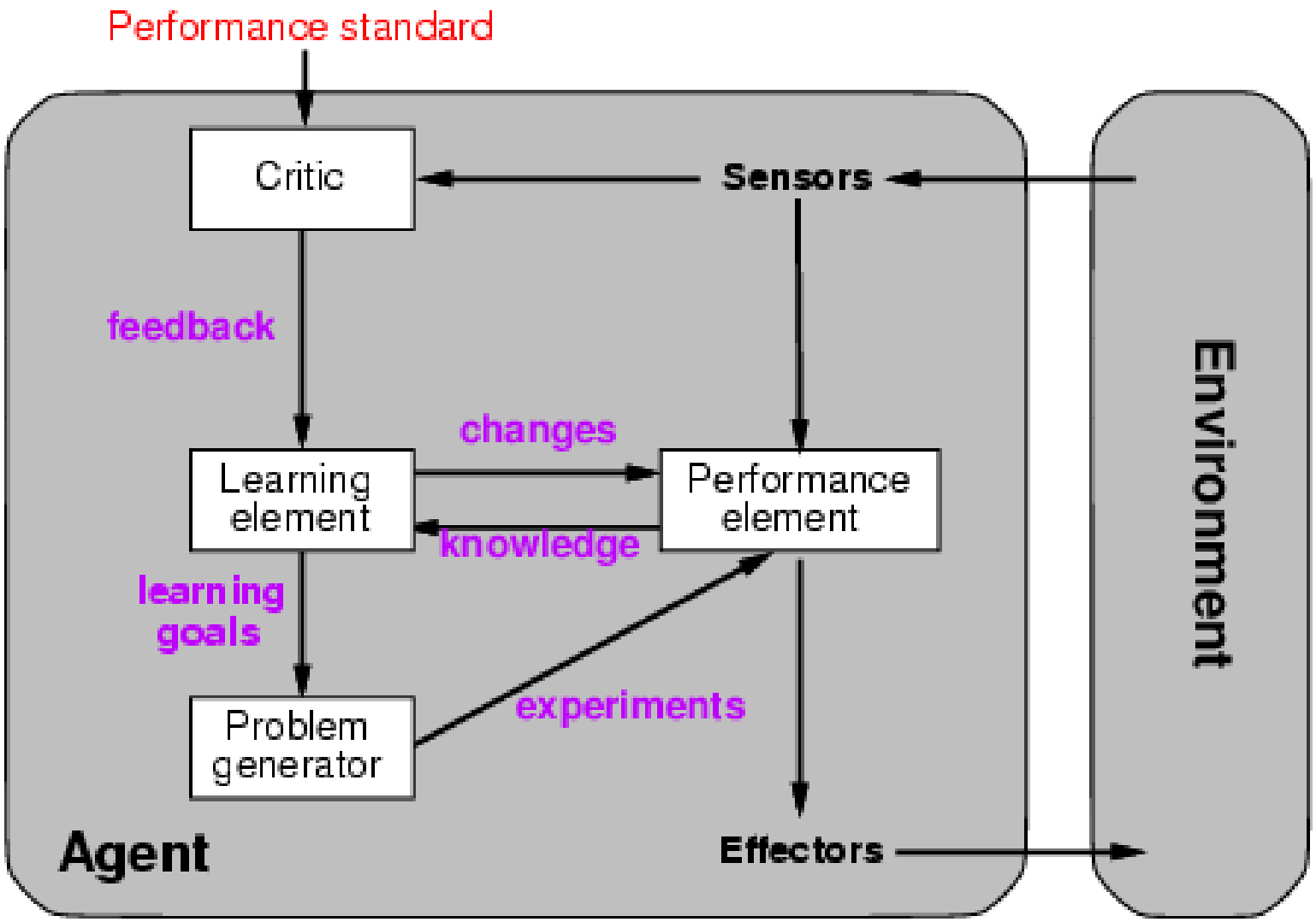
Outline

- Learning agents
- Inductive learning
- Decision tree learning
- Learning Performance measurements
- Naïve Bays Learning
- Artificial Neural Networks ANNs
- K-Means Clustering
- KNN (K-Nearest Neighbors)

Learning

- Learning is essential for unknown environments,
  - i.e., when designer lacks omniscience
- Learning is useful as a system construction method,
  - i.e., expose the agent to reality rather than trying to write it down
- Learning modifies the agent's decision mechanisms to improve performance

Learning agents



Learning element

- Design of a learning element is affected by
  - Which components of the performance element are to be learned
  - What feedback is available to learn these components
  - What representation is used for the components
- Type of feedback:
  - Supervised learning: correct answers for each example
  - Unsupervised learning: correct answers not given
  - Reinforcement learning: occasional rewards

ML: Where Used?

- Health:
  - Disease diagnosis:
  - Suicide trends
  - Extracting knowledge form report
  - Recommending stuff to patients
- Finance/Economy:
  - Predicting share prices
  - Credit approval decisions
- Law:
  - Extracting knowledge form report
  - Predicting case outcomes

ML: Where Used?

- Publishing:
  - Predict successful publications/Novels.
  - Detect Plagiarism: determining author of Docs.
  - Document Classification
- Politics:
  - Voter trends and voter influence
  - Selecting potentiall winning candidates
- Security:
  - Detecting security threats
  - Identifying potential intruders based on style

Inductive learning

- Simplest form: learn a function from examples

f is the target function

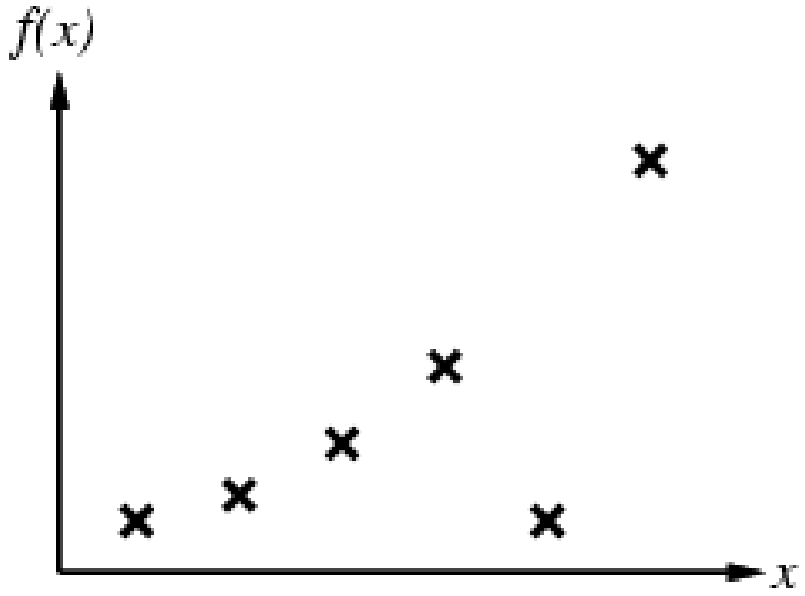
An example is a pair (*x*, *f(x)*)

Problem: find a hypothesis *h*
such that $h \approx f$
given a training set of examples (table of pair (*x*, *f(x)*))

(This is a highly simplified model of real learning:
  – Ignores prior knowledge
  – Assumes examples are given and are consistent (not conflicting)

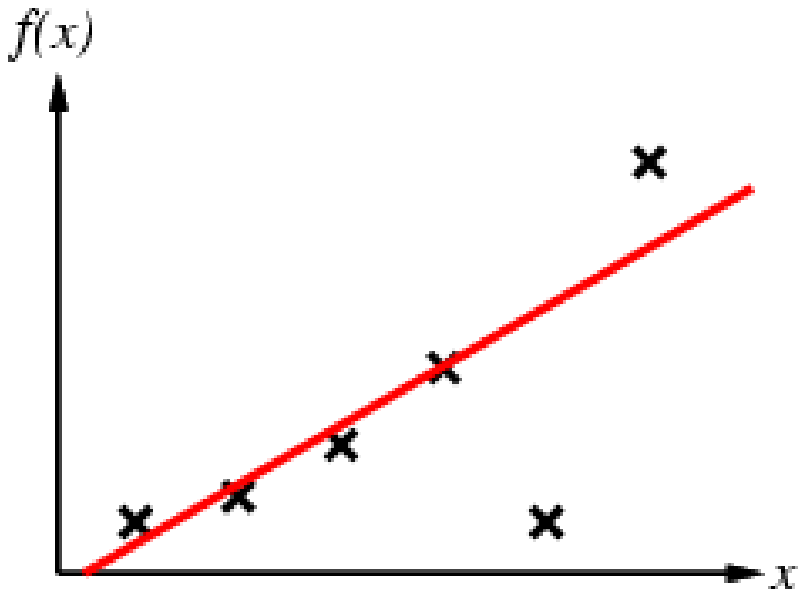
Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on **all** examples)
- Too strict: all→ most/many (error tolerance)
- E.g., curve fitting:
-



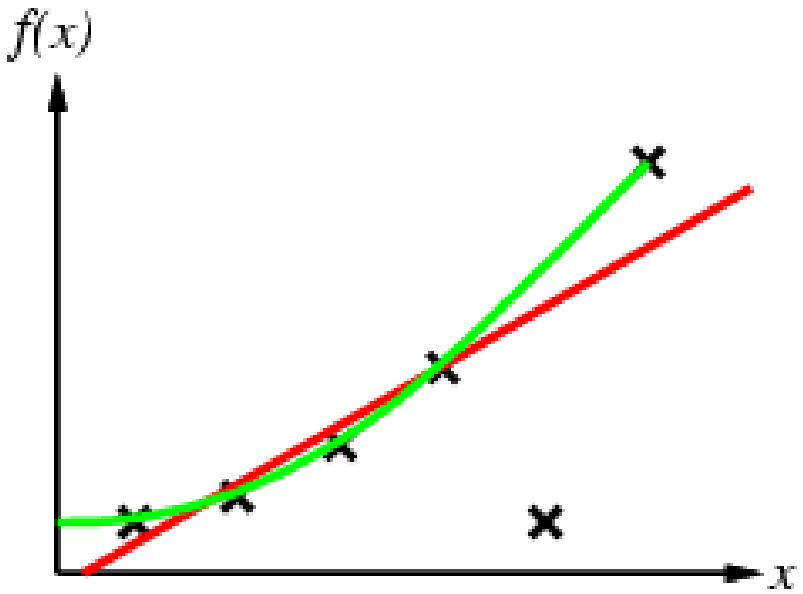
Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
-
- E.g., curve fitting:
-



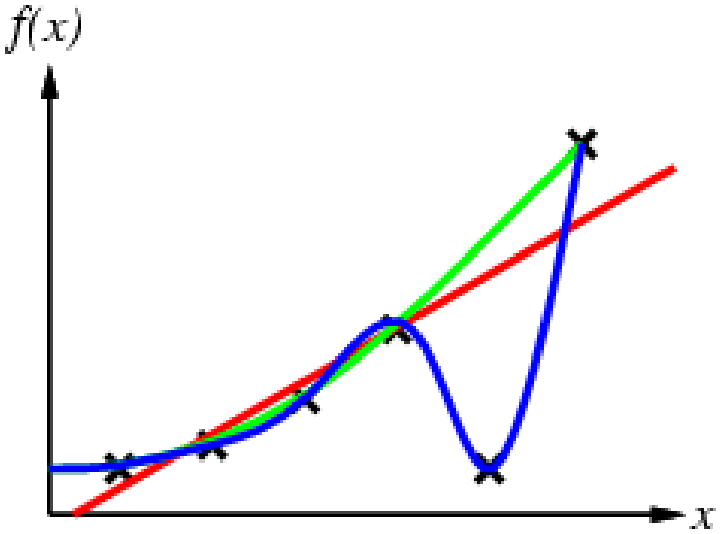
Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
-
- E.g., curve fitting:
-



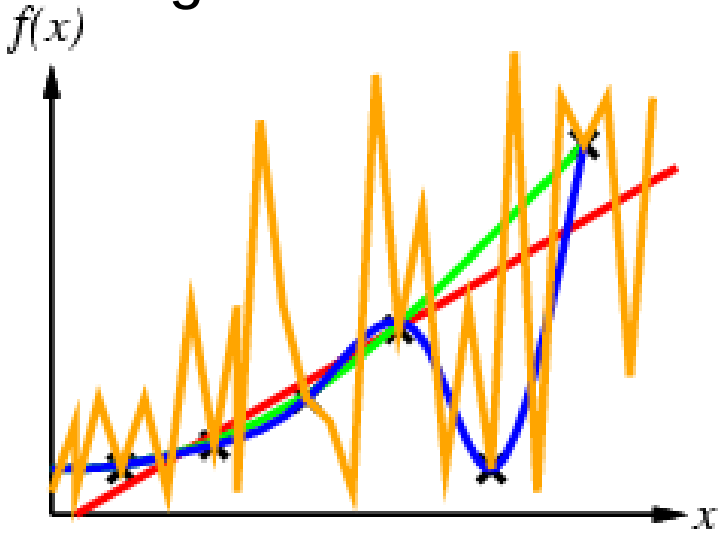
Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
-
- E.g., curve fitting:
-



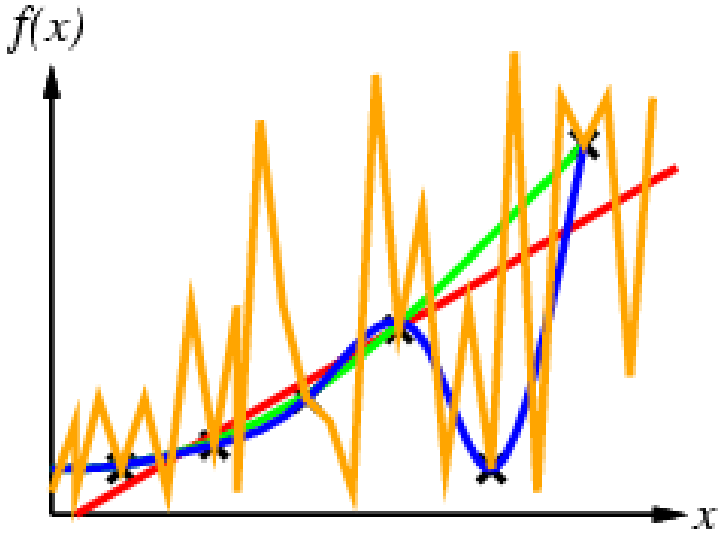
Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
- E.g., curve fitting:



Inductive learning method

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)
- E.g., curve fitting:



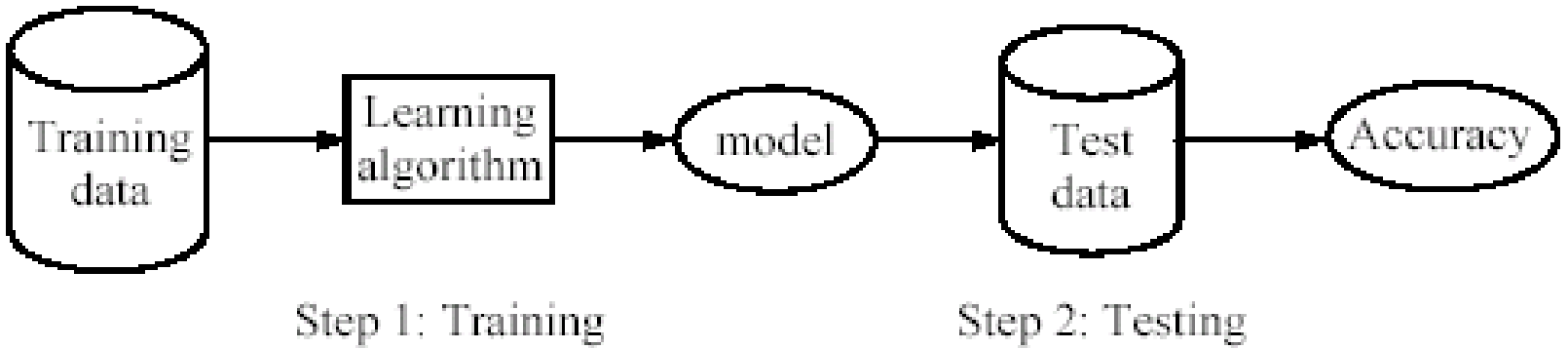
- Occam's razor: prefer the simplest hypothesis consistent with data (KIS: Keep It Simple)

Supervised learning process: two steps

Learning (training): Learn a model using the training data

Testing: Test the model using unseen test data to assess the model accuracy

$$Accuracy = \frac{\text{Number of correct classifications}}{\text{Total number of test cases}},$$



What do we mean by learning?

- Given
  - a **data set** *D*,
  - a task *T*, and
  - a performance measure ***M***,

  a computer system is said to **learn** from *D* to perform the task *T* if after learning the system's performance on *T* improves as measured by *M*.
- In other words, the learned model helps the system to perform *T* better as compared to no learning.

Learning decision trees

Problem: **To Wait or not to Wait**: decide whether to wait for a table at a restaurant, based on the following attributes:

1. Alternate: is there an alternative restaurant nearby?
2. Bar: is there a comfortable bar area to wait in?
3. Fri/Sat: is today Friday or Saturday?
4. Hungry: are we hungry?
5. Patrons: number of people in the restaurant (None, Some, Full)
6. Price: price range ($, $$, $$$)
7. Raining: is it raining outside?
8. Reservation: have we made a reservation?
9. Type: kind of restaurant (French, Italian, Thai, Burger)
10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

Attribute-based Representations

- Examples described by attribute values (Boolean, discrete, continuous)
- E.g., situations where I will/won't wait for a table: T wait, F don't wait

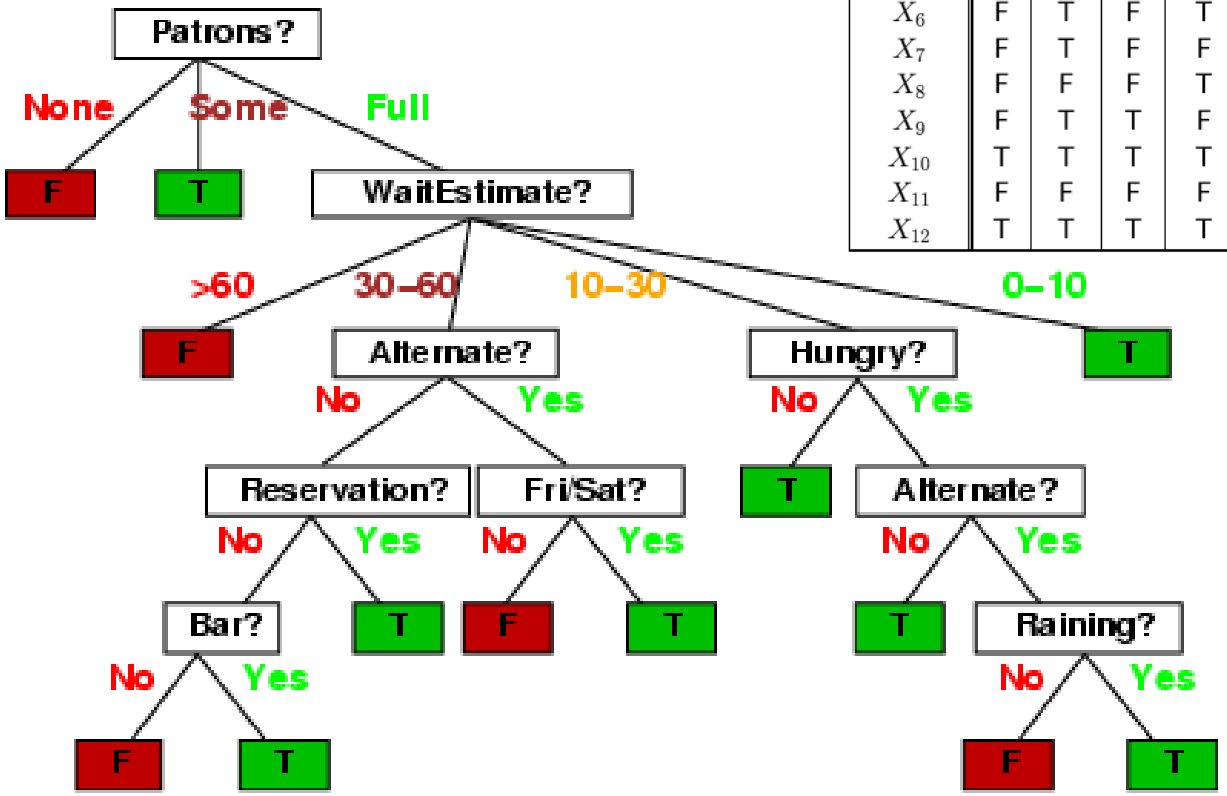
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

- We are learning Attribute **Wait**
- Classification of examples on **Wait** is positive (T) or negative (F)

Supervised Learning: Decision Trees

- **DT**: One possible representation for hypotheses
- E.g., here is the "true" tree for deciding whether to wait:

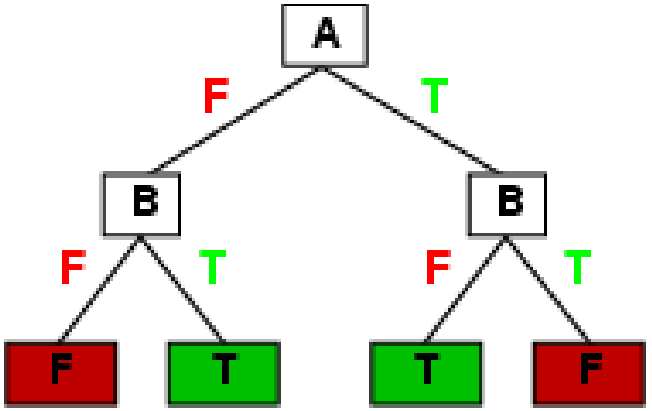
Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T



Expressiveness

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example (unless *f* nondeterministic in *x*) but it probably won't generalize to new examples
- **Generally, DT is not unique for a set of data**
- Prefer to find more compact decision trees

Hypothesis spaces

How many distinct decision trees with *n* Boolean attributes?

= number of Boolean functions (Value: True or False, 1 or 0)

= number of distinct truth tables with 2^n rows = 2^{2^n}

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

- If variables are non_boolean: say 10 possibilities each: with n=6 such attributes, number of distinct values 10^n :number of distinct truth tables with 10^n rows = 10^{10^n}

Hypothesis spaces

How many distinct decision trees with *n* Boolean attributes?

= number of Boolean functions

= number of distinct truth tables with 2^n rows = 2^{2^n}

- E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees

How many purely conjunctive hypotheses (e.g., $Hungry \wedge \neg Rain$)?

- Each attribute can be **in** (positive), **in** (negative), or **out**

  $\Rightarrow 3^n$ distinct conjunctive hypotheses for n attributes
- More expressive hypothesis space
  - increases chance that target function can be expressed
  - increases number of hypotheses consistent with training set

    $\Rightarrow$ may get worse predictions

Decision tree learning

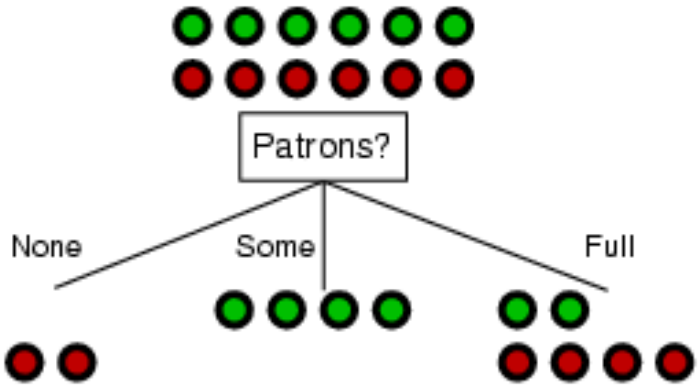
- Aim: find a **small** tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

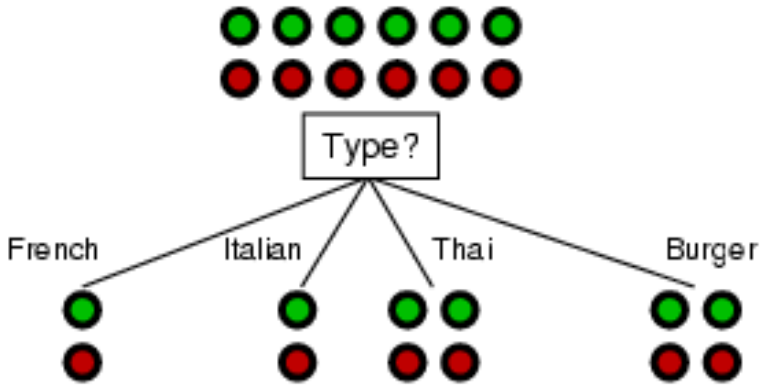
```
function DTL(examples, attributes, default) returns a decision tree
    if examples is empty then return default
    else if all examples have the same classification then return the classification
    else if attributes is empty then return MODE(examples)
    else
        best ← CHOOSE-ATTRIBUTE(attributes, examples)
        tree ← a new decision tree with root test best
        for each value v_i of best do
            examples_i ← {elements of examples with best = v_i}
            subtree ← DTL(examples_i, attributes − best, MODE(examples))
            add a branch to tree with label v_i and subtree subtree
        return tree
```

- Which attribute to choose? **Most discriminating**?

Choosing an attribute

- Idea: a good attribute splits the examples into subsets that are (ideally) "all positive" or "all negative“ for its values





- *Patrons?* is a better choice, Why?
- If we take Patrons: =none: **Red**, Some: **Green**, Full: **Red** (Why Red?). Errors: 2 out of 12=1/6 (on value=full).
- If we take Type: =French: **Red**, Italian: **Green**, Thai: **Red** , Burger: **Red** Errors: 6 out of 12=1/2 (on all values).

Using information theory

- To implement `Choose-Attribute` in the DTL algorithm
- Information Content (Entropy):

$$I(P(v_1), \ldots, P(v_n)) = \Sigma_{i=1} -P(v_i) \log_2 P(v_i)$$

- For a training set containing *p* positive examples and *n* negative examples:

$$I(\frac{p}{p+n}, \frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Here: p=6, n=6; p/(n+p)=1/2; n/(n+p)=1/2; I(1/2,1/2)=1/2+1/2=1bit

If p=3, n=9; p/(n+p)=1/4; n/(n+p)=3/4; I(1/4,3/4)=1/4*2+3/4*.4=0.8bit

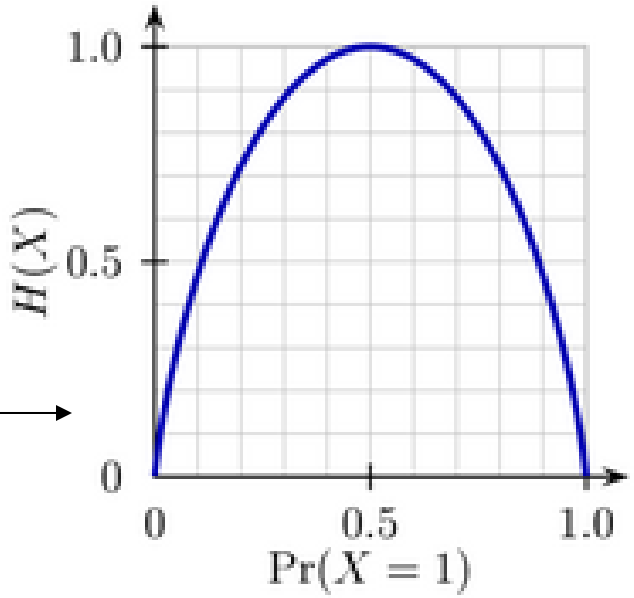
Using information theory

- Entropy measures the amount of uncertainty in a probability distribution:

Consider tossing a biased coin.
If you toss the coin VERY often,
the frequency of heads is, say, p,
and hence the frequency of tails is
1-p. (fair coin p=0.5).

The uncertainty in any actual outcome is given by the entropy:

Note, the uncertainty is zero if p=0 or 1 and maximal if we have p=0.5.



Information gain

- A chosen attribute *A* divides the training set *E* into subsets $E_1, \ldots, E_v$ according to A values, where *A* has v distinct values.

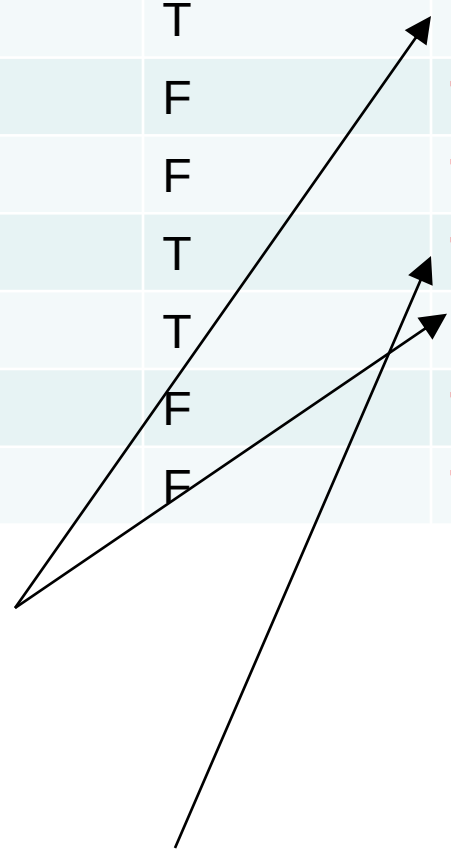
$$remainder(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

- Defines how discriminating this Attribute is
- Information Gain (IG)/reduction in entropy from the attribute test:

$$IG(A) = I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) - remainder(A)$$

- IG: entropy of the parent – weighted sum of entropy of children
- Defines difference (improvement) in discrimination between the **Learned attribute** and the **tested attribute**.
- Choose the attribute with the largest IG

A	B	C	D	E	Concept
				F	T
				T	F
				F	T
				F	T
				T	T
				T	F
				F	T
				F	T



Concept= True: error: 2/8

Entropy=0.8

E: F→True: T→Flase: Error: 1/8

Information gain: example

$$I(\frac{p}{p+n},\frac{n}{p+n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

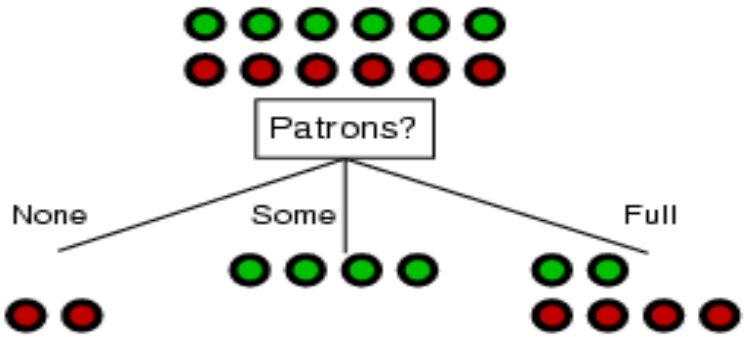
Consider the attributes *Patrons* and *Type* (and others too):

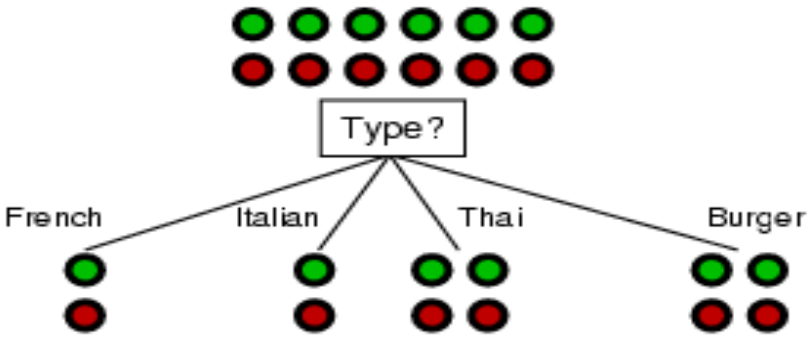
$$remainder(A) = \sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(\frac{p_i}{p_i+n_i},\frac{n_i}{p_i+n_i}) \qquad IG(A) = I(\frac{p}{p+n},\frac{n}{p+n}) - remainder(A)$$

$$IG(Patrons) = 1-[\frac{2}{12}I(0,1)+\frac{4}{12}I(1,0)+\frac{6}{12}I(\frac{2}{6},\frac{4}{6})] = .0541\,\text{bits}$$

$$IG(Type) = 1-[\frac{2}{12}I(\frac{1}{2},\frac{1}{2})+\frac{2}{12}I(\frac{1}{2},\frac{1}{2})+\frac{4}{12}I(\frac{2}{4},\frac{2}{4})+\frac{4}{12}I(\frac{2}{4},\frac{2}{4})] = 0\,\text{bits}$$

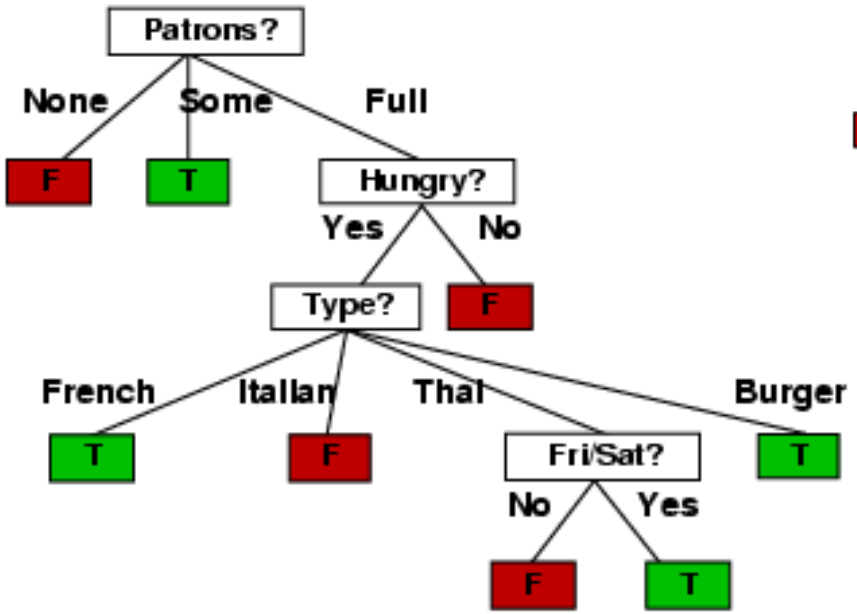
Do that for all 10 : Assume Patrons has the highest IG of all attributes and so is chosen by the DTL algorithm as the root

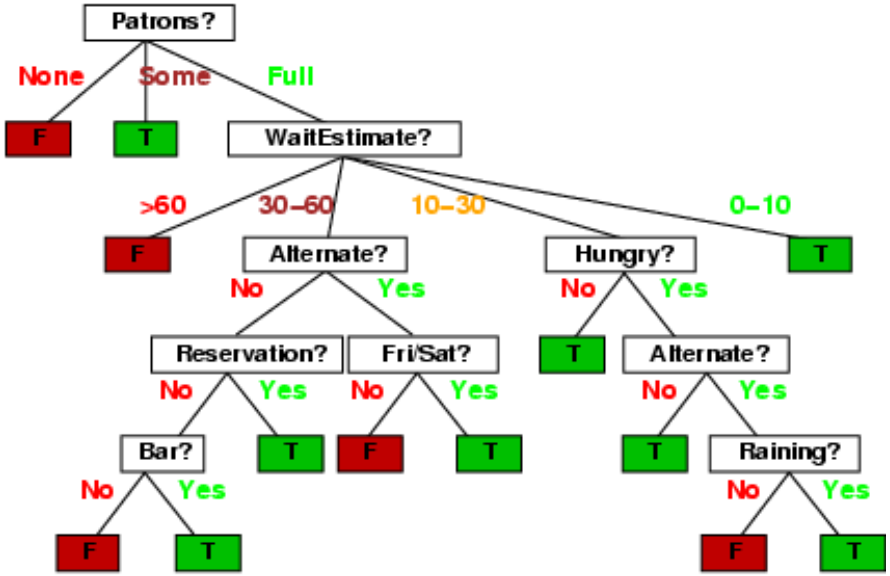




DT Example: contd.

- Decision tree learned from the 12 examples: tested 10 attributes, Started with Patrons, Then tested 9: Hungry,...
- Always Stop when leaves are pure (all T/all F here)





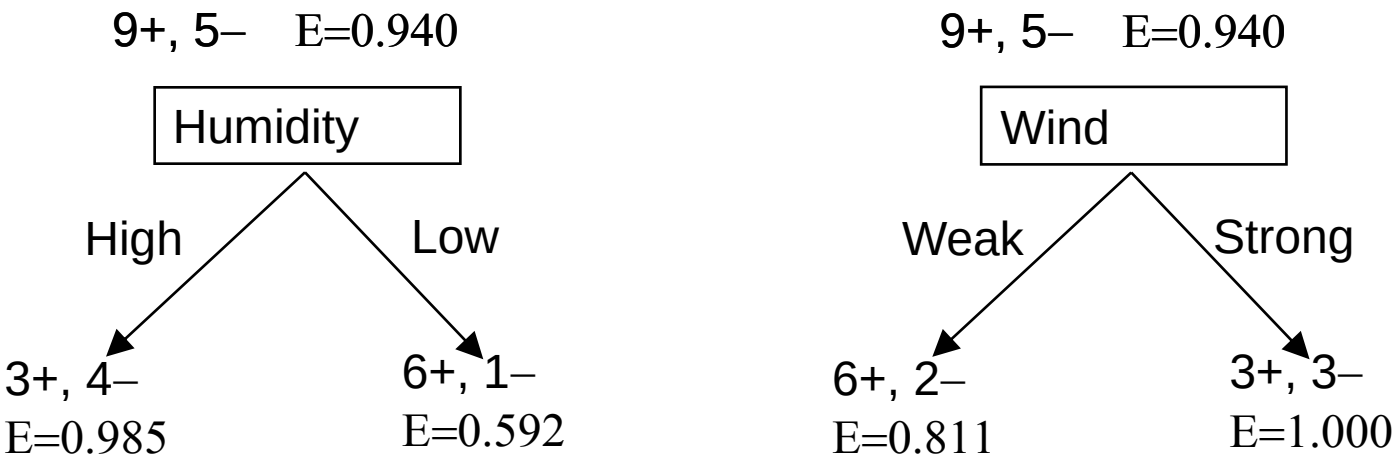
- Substantially simpler than “true” tree. Less leaves also.
- a more complex hypothesis isn’t justified by small amount of data

DT Example2: To Play or Not to Play

14 examples, 4 attributes, Yes/No Concept

Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*
D10	Rain	Mild	Normal	Weak	*Yes*
D11	Sunny	Mild	Normal	Strong	*Yes*
D12	Overcast	Mild	High	Strong	*Yes*
D13	Overcast	Hot	Normal	Weak	*Yes*
D14	Rain	Mild	High	Strong	*No*

Determine the Root Attribute



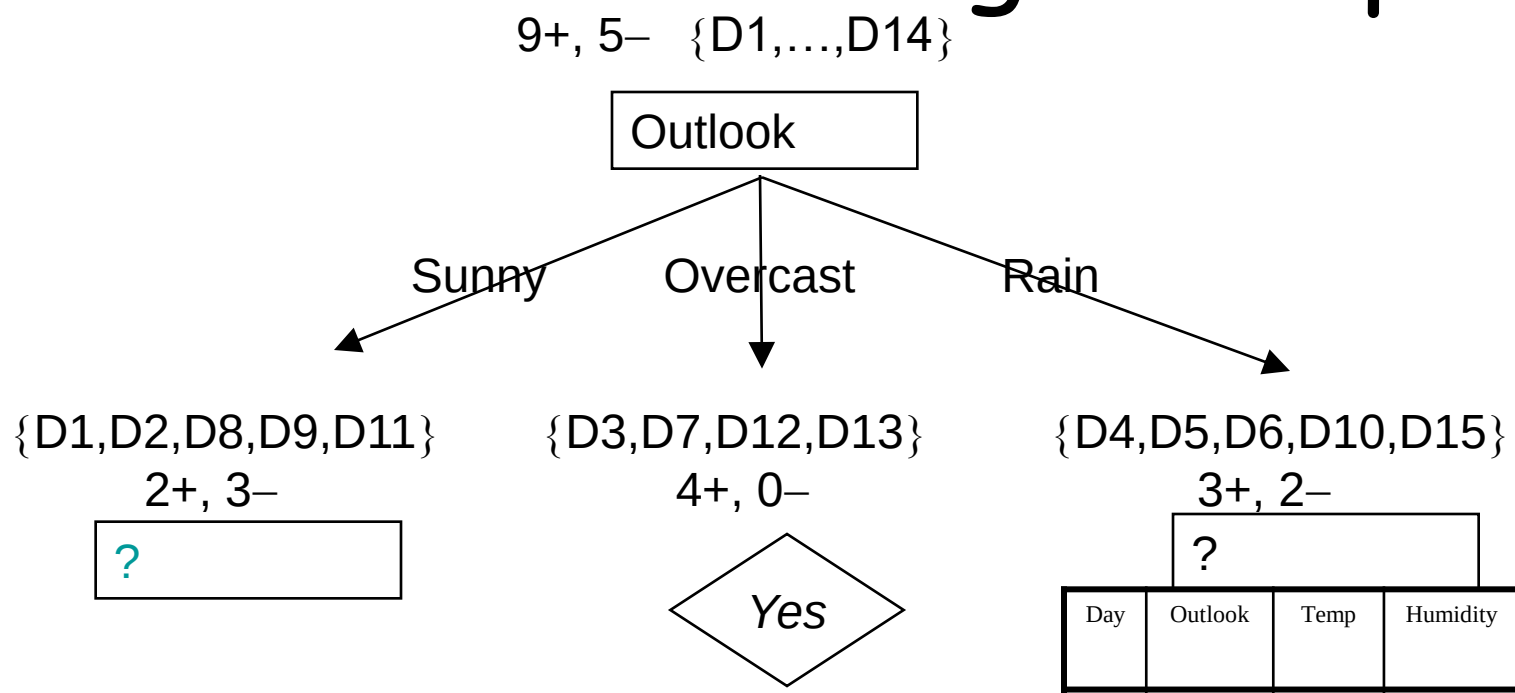
Gain (S, Humidity) = 0.151

Gain (S, Wind) = 0.048

Gain (S, Outlook) = 0.246

Gain (S, Temp) = 0.029

Sort the Training Examples



Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*

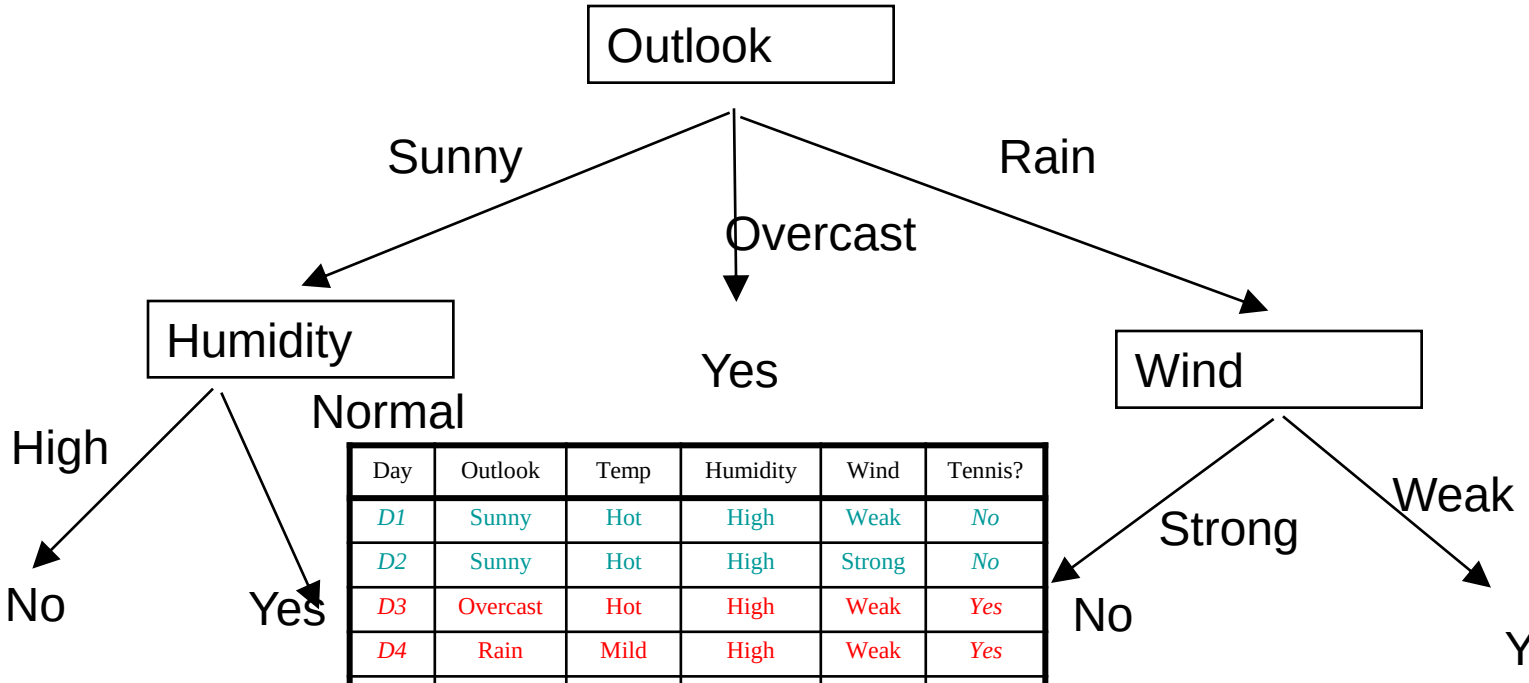
S_{sunny}= {D1,D2,D8,D9,D11}

Gain (S_{sunny}, Humidity) = .970

Gain (S_{sunny}, Temp) = .570

Gain (S_{sunny}, Wind) = .019

Final Decision Tree for Example

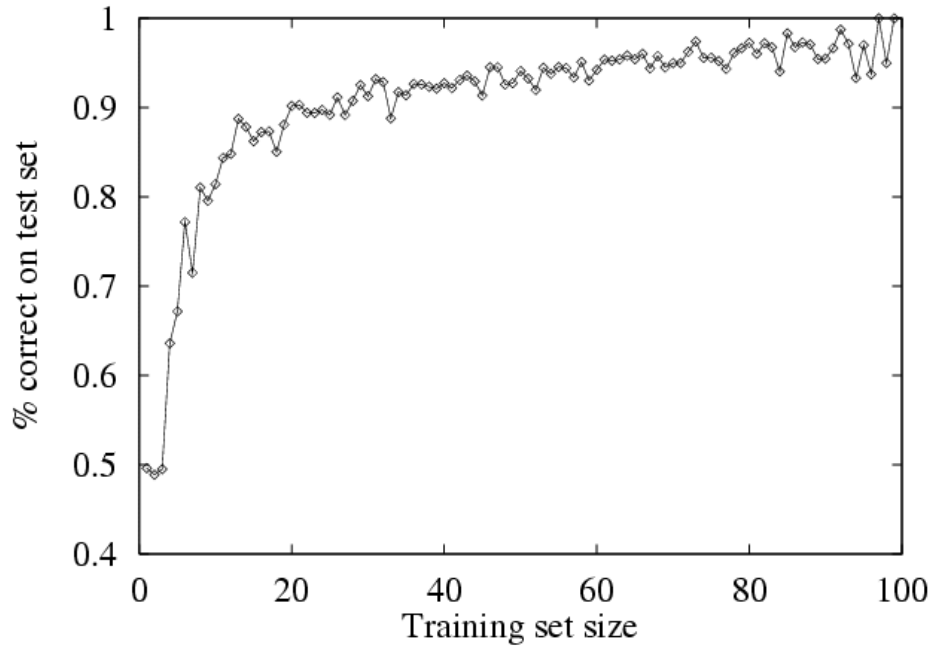


Day	Outlook	Temp	Humidity	Wind	Tennis?
D1	Sunny	Hot	High	Weak	*No*
D2	Sunny	Hot	High	Strong	*No*
D3	Overcast	Hot	High	Weak	*Yes*
D4	Rain	Mild	High	Weak	*Yes*
D5	Rain	Cool	Normal	Weak	*Yes*
D6	Rain	Cool	Normal	Strong	*No*
D7	Overcast	Cool	Normal	Strong	*Yes*
D8	Sunny	Mild	High	Weak	*No*
D9	Sunny	Cool	Normal	Weak	*Yes*
D10	Rain	Mild	Normal	Weak	*Yes*
D11	Sunny	Mild	Normal	Strong	*Yes*
D12	Overcast	Mild	High	Strong	*Yes*
D13	Overcast	Hot	Normal	Weak	*Yes*
D14	Rain	Mild	High	Strong	*No*

Performance measurement

- How do we know that $h \approx f$?
  1. Use theorems of computational/statistical learning theory
  2. Try *h* on a new test set of examples
     (use same distribution over example space as training set)

Learning curve = % correct on test set as a function of training set size



Evaluation

Evaluation methods

- **Holdout set**: The available data set D is divided into two disjoint subsets,
  - the *training set* D_{train} (for learning a model)
  - the *test set* D_{test} (for testing the model)
- **Important:** training set should not be used in testing and the test set should not be used in learning.
  - Unseen test set provides a unbiased estimate of accuracy.
- The test set is also called the holdout set. (the examples in the original data set D are all labeled with classes.)
- This method is mainly used when the data set D is large.

Given 120 examples: Holdout: 25%:75% (30:90) or 50%:50% (60:60) or **34%:66% (40:80) usual [one test, random]**

Evaluation

Evaluation methods (cont...)

- **n-fold cross-validation**: The available data is partitioned into n equal-size disjoint subsets.
- Use each subset as the test set and combine the rest $n-1$ subsets as the training set to learn a classifier.
- The procedure is run n times, which give n accuracies.
- The final estimated accuracy of learning is the average of the n accuracies.
- 10-fold and 5-fold cross-validations are commonly used.
- This method is used when the available data is not large.

Consider our 12 example:6 fold: **6x2**:

S1: Test {1,2}, Training{3,4,…12}, S2: Test {3,4}, Training{1,2,5,6,…12},
S3: Test {5,6}, Training{1-4,7,8-12}, S4: Test {7,8}, Training{1-6,9-12},

S5: Test {9,10}, Training{1-8,11,12}, S6: Test {11,12}, Training{1-10}.

- Each can have: 100% success, 0% success, 50% success: average=?
- Order can vary! ➔6 runs, 12 tests, 8 correct: success=8/12

Evaluation

Evaluation methods (cont...)

- **Leave-one-out cross-validation**: This method is used when the data set is very small.
- It is a special case of cross-validation
- Each fold of the cross validation has only a single test example and all the rest of the data is used in training.
- If the original data has m examples, this is m-fold cross-validation

Given total of 12 examples: **[12 tests, 12 runs]: each success or fail: 8 fails: accuracy: 4/12**

Evaluation

Evaluation methods (cont...)

- **Validation set**: the available data is divided into three subsets,
  - a training set,
  - a validation set and
  - a test set.
- A validation set is used frequently for estimating parameters in learning algorithms.
- In such cases, the values that give the best accuracy on the validation set are used as the final parameter values.
- Cross-validation can be used for parameter estimating as well.

Evaluation

Classification measures

- Accuracy is only one measure (error = 1-accuracy).
- **Accuracy is not suitable in some applications.**
- In text mining, we may only be interested in the documents of a particular topic, which are only a small portion of a big document collection.
- In classification involving skewed or highly imbalanced data, e.g., network intrusion and financial fraud detections, we are interested only in the minority class.
  - High accuracy does not mean any intrusion is detected.
  - E.g., 1% intrusion. Achieve 99% accuracy by doing nothing.
- The class of interest is commonly called the positive class, and the rest negative classes.

- Accuracy is about correct answers.
- Go with the majority class to get it high: very high if negative class dominates: Web Search
- 1000K items, only 1K positive: All negative! **99.9%**

Evaluation: Beyond Accuracy

- So need better measure: we classify to Positive (our class) and Negative (the rest):
- Some of the declared positive are really positive (TP√) and some are not (FP)
- Some of the declared negative are really negative (TN√) and some are not (FN)
- Really (actual) positive= TP√ + FN
- Really (actual) negative= TN√ + FP

Evaluation

- Database with 10 localities: 6 cities and 4 villages:

Jaffa, Ramallah, Ram, Jerusalem, Hebron,

Abu Qash, Birzeit, Surda, Jifna, Gaza

- Query: List **villages** in Palestine: RED: Actual Positive, Green?
- **Answer (Positive):** Abu Qash, Birzeit, Surda, Jifna, Gaza
- TP FP TP TP FP
- **Not Village (negative) by default: implicit not listed!:**
- Jaffa, Ramallah, Ram, Jerusalem, Hebron,
- TN TN FN TN TN
- How many errors made? 3/10
- If we always return all as cities: error =4/10
- If we always return all as Villages: error =6/10 (majority rule)
- What if: 1000 localities, all villages but only 10 cities!

Evaluation

Precision and recall measures

- Used in information retrieval and text classification.
- We use a confusion matrix to introduce them.

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where

TP: the number of correct classifications of the positive examples (**true positive**),

FN: the number of incorrect classifications of positive examples (**false negative**),

FP: the number of incorrect classifications of negative examples (**false positive**), and

TN: the number of correct classifications of negative examples (**true negative**).

Evaluation

Precision and recall measures (cont...)

	Classified Positive	Classified Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

$$p = \frac{TP}{TP + FP}. \qquad r = \frac{TP}{TP + FN}.$$

Precision p is the number of correctly classified positive examples divided by the total number of examples that are classified as positive.

Recall r is the number of correctly classified positive examples divided by the total number of actual positive examples in the test set.

- p (r) has to do with declared (**real**) positive
- Accuracy: (TP+TN)/(TP+FP+TN+FN)

Evaluation

An example

	Classified Positive	Classified Negative
Actual Positive	1	99
Actual Negative	0	1000

- This confusion matrix gives
  - precision $p = 100\%$ and
  - recall $r = 1\%$

  because we only classified one positive example correctly and no negative examples wrongly.
- Note: precision and recall only measure classification on the positive class.

	actual positive	actual negative
predicted positive	TP	FP
predicted negative	FN	TN

(a) Confusion Matrix

Recall	$\frac{TP}{TP+FN}$
Precision	$\frac{TP}{TP+FP}$
True Positive Rate	$\frac{TP}{TP+FN}$
False Positive Rate	$\frac{FP}{FP+TN}$

(b) Definitions of metrics

- Accuracy= (TP+TN)/(TP+FP+TN+FN)=(1+1000)/1100=0.90+

- Can have high accuracy and recall separately easily:
  - Declare all positive: 100% r, low p
  - Declare all negative: 0 recall, High p

  Need a composite measure:

  F measure: $1/F=1/2(1/p+1/r) =1/2((p+r)/p*r) =(p+r)/2p*r$ ➔ **$F=2*p*r/(p+r)$**

Evaluation

F_1-value (also called F_1-score)

- It is hard to compare two classifiers using two measures. F_1 score combines precision and recall into one measure

$$F_1 = \frac{2pr}{p + r}$$

F_1-score is the harmonic mean of precision and recall.

$$F_1 = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

- The harmonic mean of two numbers tends to be closer to the smaller of the two.
- For F_1-value to be large, both p and r much be large.

Examples

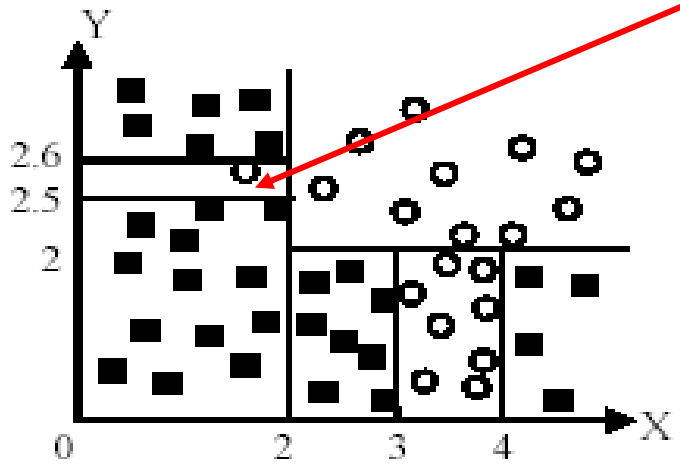
- p=1, r=0, F1=0/1=0
- p=0, r=1, F1=0/1=0
- p=1, r=1, F1=2/2=1
- p=1/2, r=1/2, F1=(1/2)/1=1/2
- p=0.8, r=0.8, F1=1.28/1.6=0.8
- p=0.2, r=0.8, F1=0.32/1=0.32
- p=0.2, r=0.1, F1=.04/0.3=0.13

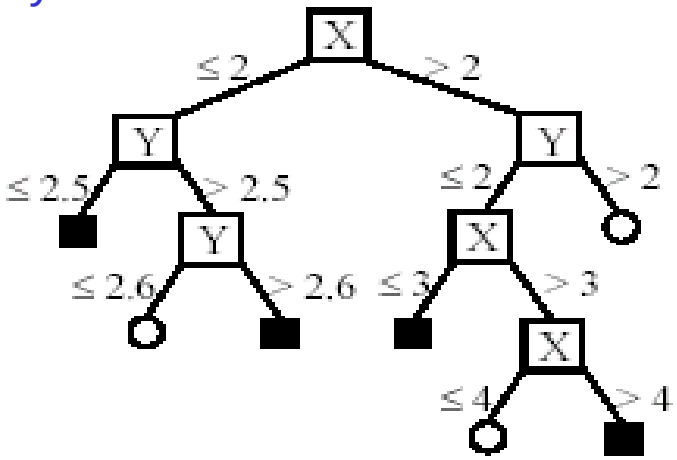
Avoid overfitting in classification

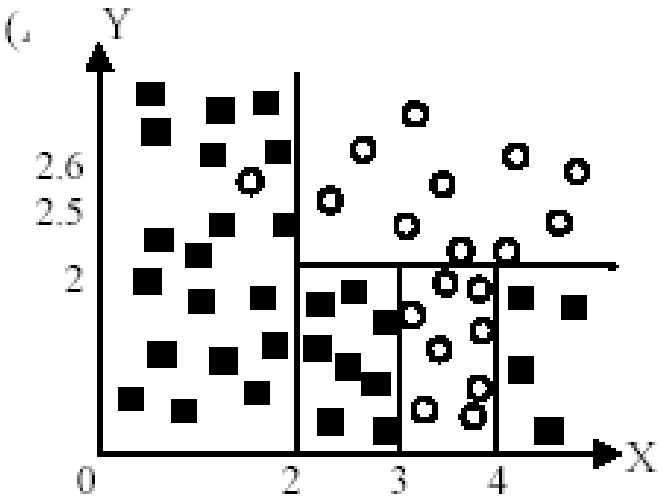
- Ideal goal of classification: Find the simplest decision tree that fits the data and **generalizes** to unseen data
- Overfitting: A tree may overfit the training data
  - **Good** accuracy on training data, poor on test data
  - Symptoms: tree too **deep** and too **many** branches, some may reflect anomalies due to noise or outliers
  - Overfitting results in decision trees that are more complex than necessary
  - Trade-off: full consistency for compactness
    - Larger decision trees can be more consistent
    - Smaller decision trees generalize better

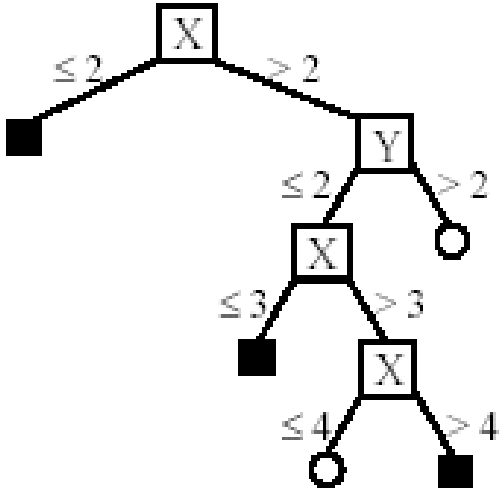
An example

Likely to overfit the data







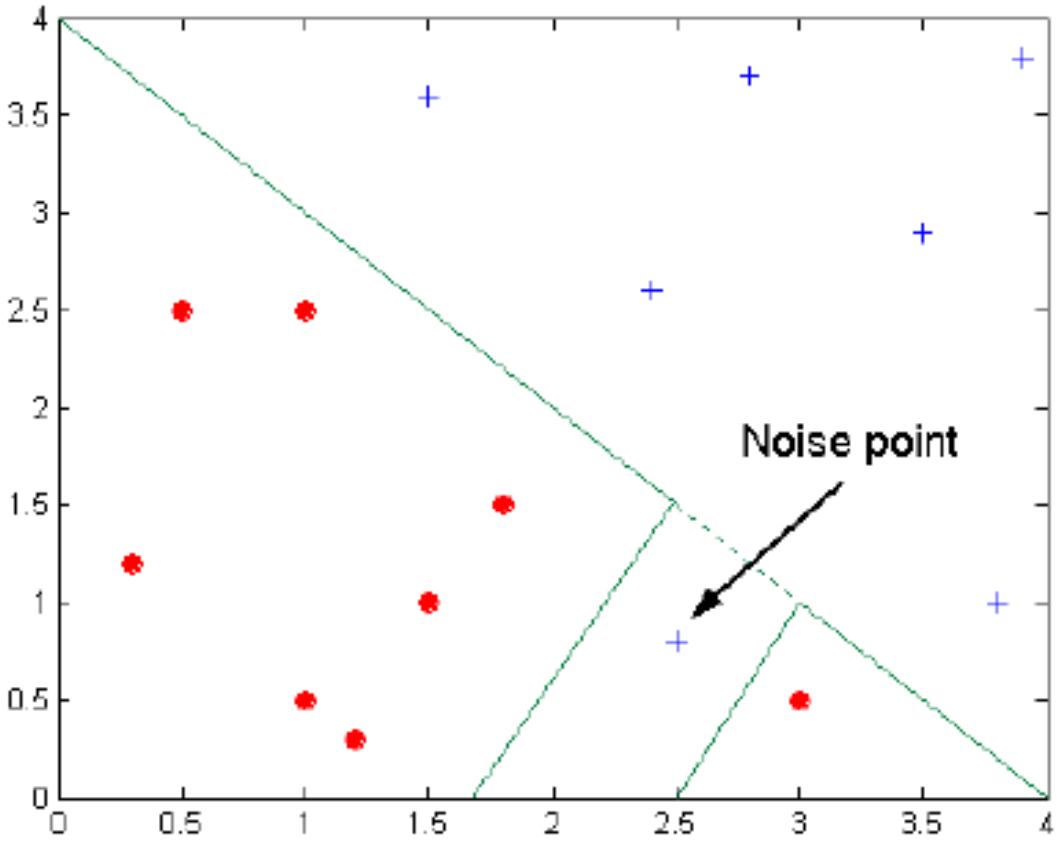


Simple Boolean Example

#	A	B	C	D	F
1	0	0	1	0	T
2	0	0	0	1	T
3	0	0	1	1	T
4	0	0	0	0	T➔ F
5	1	1	1	1	F
6	1	1	0	1	F
7	1	1	1	0	F➔T

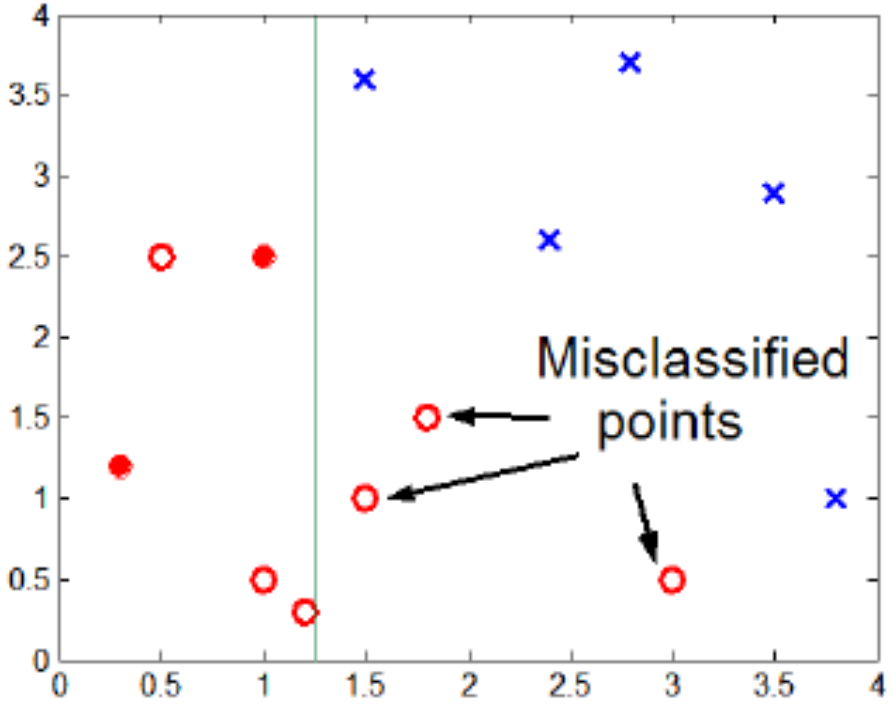
- The function=F=A’
- Change 4 to F: F= A’ and B’
- Change 7 to T: F= ??

Overfitting due to Noise



Decision boundary is distorted by noise point

Overfitting due to Insufficient Examples

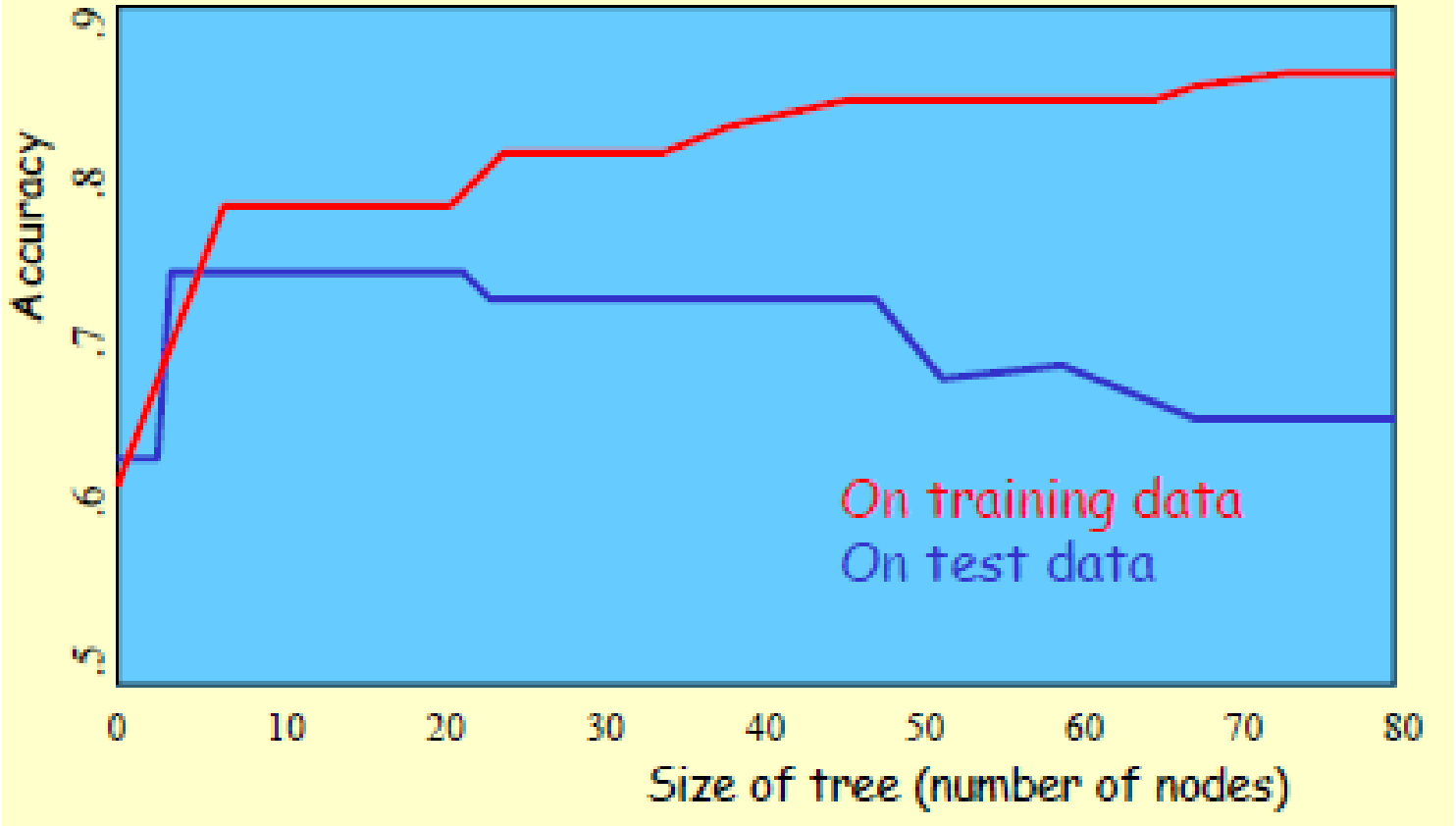


Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task

Overfitting and accuracy

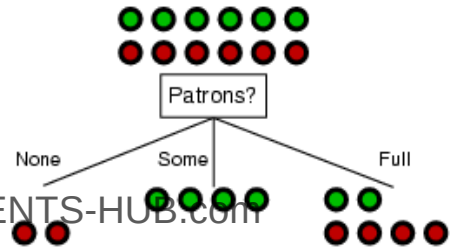
- Typical relation between tree size and accuracy:

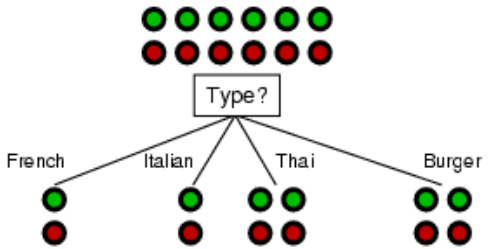


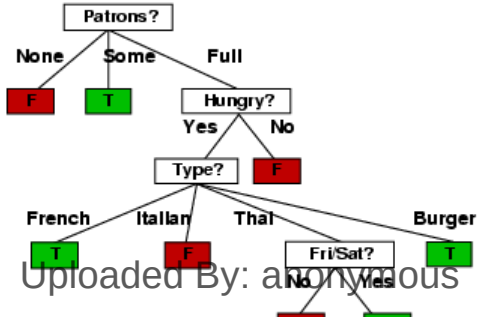
Pruning to avoid overfitting

- **Prepruning**: Stop growing the tree when there is not enough data to make reliable decisions or when the examples are acceptably homogenous (ID3)
  - Do not split a node if this would result in the goodness measure falling below a threshold (e.g. InfoGain)
  - Difficult to choose an appropriate threshold
  - Since we use a hill-climbing search, looking only one step ahead, pre-pruning might stop too early.

- **Postpruning**: Grow the full tree, then remove nodes for which there is not sufficient evidence (C4.5)
  - Replace a split (subtree) with a leaf if the *predicted validation error is* no worse than the more complex tree (use dataset)
- Prepruning easier, but postpruning works better
- Prepruning - hard to know when to stop







Decision Trees: the good and the bad

- Advantages:
  - Easy to understand (Doctors love them!)
  - Easy to generate rules
- Disadvantages:
  - May suffer from overfitting.
  - Classifies by rectangular partitioning (so does not handle correlated features very well).
  - Can be quite large – pruning is necessary.
  - Does not handle streaming data easily

Supervised Learning:
2- Naïve Bayes Classifier

Thomas Bayes
1702 - 1761

Why Probabilities

Kolmogorov showed that three simple axioms lead to the rules of probability theory

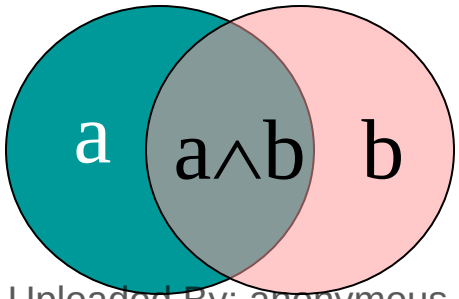
1. All probabilities are between 0 and 1:

$$0 \leq P(a) \leq 1$$

2. Valid propositions (tautologies) have probability 1, and unsatisfiable propositions have probability 0:

$$P(\text{true}) = 1 \; ; \; P(\text{false}) = 0$$

3. The probability of a disjunction is given by: $P(a \vee b) = P(a) + P(b) - P(a \wedge b)$



Probability theory 101

- **Random variables**
  - Domain
- **Atomic event**: complete specification of state
- **Prior probability**: degree of belief without evidence
- **Joint probability**: matrix of combined probabilities of a set of variables

- Alarm, Burglary, Earthquake
  - Boolean, discrete, continuous
- Alarm=T∧Burglary=T∧Earthquake=F
  alarm ∧ burglary ∧ ¬earthquake
- P(Burglary) = 0.1
  P(Alarm) = 0.19
  P(earthquake) = 0.000003
- P(Alarm, Burglary) =

	alarm	**¬alarm**
burglary	.09	.01
¬burglary	.1	.8

Probability101 Contd.

	alarm	¬alarm
burglary	.09	.01
¬burglary	.1	.8

- **Conditional probability**: prob. of effect given causes
- **Computing conditional probs**: a given b (evidence)
  - P(a | b) = P(a ∧ b) / P(b)
  - P(b): **normalizing** constant-known
- **Product rule**:
  - P(a ∧ b) = P(a | b) * P(b)
  - P(a ∧ b) = P(b | a) * P(a)
  - P(a | b) * P(b)=P(b | a) * P(a)
  - **P(a | b) =P(b | a) * P(a)/* P(b)**

- P(burglary | alarm) = .47
  P(alarm | burglary) = .9
- P(burglary | alarm) =
  P(burglary ∧ alarm) / P(alarm)
  = .09/.19 = .47
- P(burglary ∧ alarm) =
  P(burglary | alarm) * P(alarm)
  = .47 * .19 = .09
- P(alarm) = P(alarm ∧ burglary) +
  P(alarm ∧ ¬burglary) = .09+.1 = .19
- P(burglary) = P(alarm ∧ burglary) +
  P(¬ alarm ∧burglary) = .09+.01 = .1

Probability Basics

- **Prior**, **conditional** and **joint** probability for random variables
  - Prior probability: $P(X)$
  - Conditional probability: $P(X_1 \mid X_2), P(X_2 \mid X_1)$
  - Joint probability: $\mathbf{X} = (X_1, X_2), P(\mathbf{X}) = P(X_1, X_2)$
  - Relationship: $P(X_1, X_2) = P(X_2 \mid X_1)P(X_1) = P(X_1 \mid X_2)P(X_2)$
  - Independence: $P(X_2 \mid X_1) = P(X_2), P(X_1 \mid X_2) = P(X_1), P(X_1, X_2) = P(X_1)P(X_2)$

Bayesian Rule: *Recall*: $P(a \wedge b) = P(a \mid b) * P(b)$, $P(a \wedge b) = P(b \mid a) * P(a)$

- $P(a \mid b) * P(b) = P(b \mid a) * P(a)$; $P(a \mid b) = P(b \mid a) * P(a) / * P(b)$; a=C, b=X,

$$P(C \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid C)P(C)}{P(\mathbf{X})}$$

$$Posterior = \frac{Likelihood \times Prior}{Evidence}$$

- P(Swede|Blonde)=P(Blonde|Swede)*P(Swede)/P(Blonde)

Hi (relatively) 70% swedes blonde .008 swedes 10% Blonde

=0.7*0.008/0.1=0.7*0.08=0.056

The Blonde's Passport!

$$P(C \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid C)P(C)}{P(\mathbf{X})}$$

$P(a \wedge b) = P(a \mid b) * P(b) = P(b \mid a) * P(a)$ ➔

P(a|b)=P(b|a)P(a)/p(b)➔ Let C-Class (country), X-Observation-**Blonde**

You saw a **Blonde** : what is the **nationality**? Choices: SW, SA, US

- 10% of world population are **Blonde** **P(Blonde)=0.1**

70% Swedes: **Blonde;** .008 world population are Swedes: P(Swede)=.008

P(Swede| Blonde)=P(Blonde |Swede)*P(Swede)/P(Blonde)

=0.7 *0.008 /0.1= **0.056** **Relatively Hi!**

10% SA : **Blonde**, 0.016 world population are SA

P(SA|Blonde)=P(Blonde|SA)*P(SA)/P(Blonde)

= 0.1 *0.016/0.1=0.1*0.16=**0.016** **Relatively low**

50% USA: **Blonde**, 0.03 world population are USA

P(USA|Blonde)=P(Blonde|USA)*P(USA)/P(Blonde)

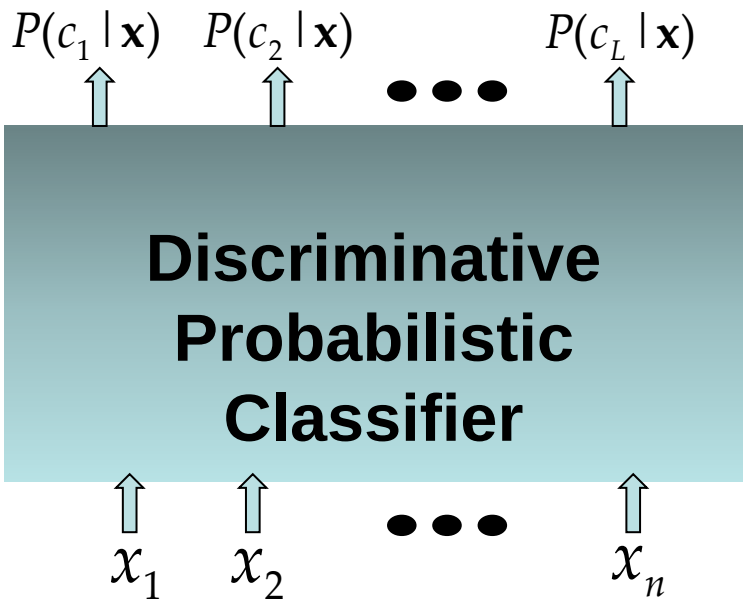
=0.5*0.03/0.1=0.5*0.3=**0.15** Real Hi

What if Blonde, Height, Language,…? Do we need to divide by 0.1

Probabilistic Classification

- Establishing a probabilistic model for classification
  - **Discriminative model**

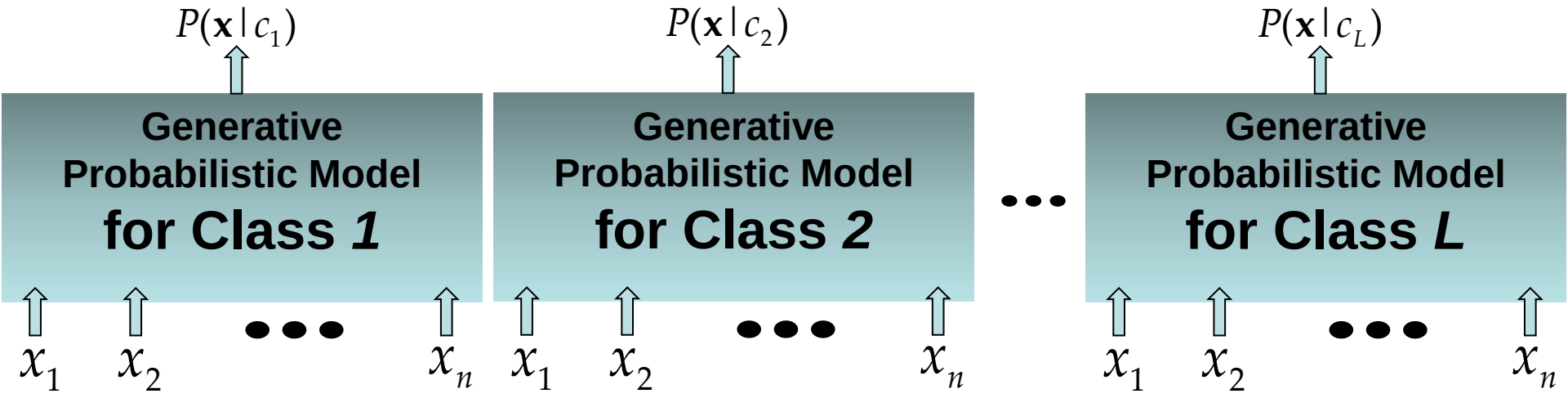
$$P(C \mid \mathbf{X}) \quad C = c_1, \cdots, c_L, \mathbf{X} = (X_1, \cdots, X_n)$$



Probabilistic Classification

- Establishing a probabilistic model for classification (cont.)
  - **Generative model**

$$P(\mathbf{X} \mid C) \quad C = c_1, \cdots, c_L, \mathbf{X} = (X_1, \cdots, X_n)$$



$$\mathbf{x} = (x_1, x_2, \cdots, x_n)$$

Probabilistic Classification

- MAP classification rule
  - **MAP**: **M**aximum **A** **P**osteriori
  - Assign example x to class (category, concept) c^* if

$$P(C = c^* \mid \mathbf{X} = \mathbf{x}) > P(C = c \mid \mathbf{X} = \mathbf{x}) \quad c \neq c^*,\ c = c_1, \cdots, c_L$$

- Generative classification with the MAP rule
  - Apply Bayesian rule to convert them into posterior probabilities
  - Then apply the MAP rule

$$P(C = c_i \mid \mathbf{X} = \mathbf{x}) = \frac{P(\mathbf{X} = \mathbf{x} \mid C = c_i) P(C = c_i)}{P(\mathbf{X} = \mathbf{x})}$$
$$\propto P(\mathbf{X} = \mathbf{x} \mid C = c_i) P(C = c_i)$$
$$\text{for } i = 1, 2, \cdots, L$$

Naïve Bayes

- Bayes classification

$$P(C \mid \mathbf{X}) \propto P(\mathbf{X} \mid C)P(C) = P(X_1, \cdots, X_n \mid C)P(C)$$

Difficulty: learning the joint probability $P(X_1, \cdots, X_n \mid C)$

- Naïve Bayes classification
  - Assumption that all input features are conditionally **independent**!
  - **Recall:** $P(X_1, X_2) = P(X_2 \mid X_1)P(X_1) = P(X_1 \mid X_2)P(X_2)$

$$\begin{aligned} P(X_1, X_2, \cdots, X_n \mid C) &= P(X_1 \mid X_2, \cdots, X_n, C)P(X_2, \cdots, X_n \mid C) \\ &= P(X_1 \mid C)P(X_2, \cdots, X_n \mid C) \\ &= P(X_1 \mid C)P(X_2 \mid C) \cdots P(X_n \mid C) \end{aligned}$$

$$[P(x_1 \mid c^*) \cdots P(x_n \mid c^*)]P(c^*) > [P(x_1 \mid c) \cdots P(x_n \mid c)]P(c), \quad c \neq c^*, c = c_1, \cdots, c_L$$

Naïve Bayes

- Algorithm: Discrete-Valued Features
  - Learning Phase: Given a training set **S** of F features and L classes,

For each target value of c_i $(c_i = c_1, \cdots, c_L)$

$\hat{P}(C = c_i) \leftarrow$ estimate $P(C = c_i)$ with examples in **S**;

For every feature value x_{jk} of each feature X_j $(j = 1, \cdots, F; k = 1, \cdots, N_j)$

$\hat{P}(X_j = x_{jk} \mid C = c_i) \leftarrow$ estimate $P(X_j = x_{jk} \mid C = c_i)$ with examples in **S**;

Output: $F * L$ conditional probabilistic (generative) models

  - Test Phase: Given an unknown instance $\mathbf{X}' = (a_1', \cdots, a_n')$

"Look up tables" to assign the label c^* to **X'** if

$$[\hat{P}(a_1' \mid c^*) \cdots \hat{P}(a_n' \mid c^*)]\hat{P}(c^*) > [\hat{P}(a_1' \mid c) \cdots \hat{P}(a_n' \mid c)]\hat{P}(c), \quad c \neq c^*, c = c_1, \cdots, c_L$$

We want to find the class (*Class*) given the features (*feautre*)

$$P(Class \mid feature)$$

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Using Bayes rule

$$P(Class \mid feature) = \frac{P(feature \mid Class) \times P(Class)}{P(feature)}$$

From the table, we will count the number of events for the **class** and each **attribute/class** combination

Feature is a vector!!! Not only **blonde** as earlier, instead:
outlook, humidity, temperature, wind

Note: When selecting examples: we have control over class, not care about individual attribute values!!!

Example

- Example: Play Tennis:
- Our class:
- C1:Play= *yes*
- C2:Play= *No*

- P(*Play=yes*)= 9/14
- P(*Play=No*)= 5/14

PlayTennis: training examples

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny*	Hot	High	Weak	No
Sunny*	Hot	High	Strong	No
Overcast**	Hot	High	Weak	Yes
Rain***	Mild	High	Weak	Yes
Rain***	Cool	Normal	Weak	Yes
Rain**	Cool	Normal	Strong	No
Overcast**	Cool	Normal	Strong	Yes
Sunny*	Mild	High	Weak	No
Sunny*	Cool	Normal	Weak	Yes
Rain***	Mild	Normal	Weak	Yes
Sunny*	Mild	Normal	Strong	Yes
Overcast**	Mild	High	Strong	Yes
Overcast**	Hot	Normal	Weak	Yes
Rain**	Mild	High	Strong	No

Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Temperature	Play=*Yes*	Play=*No*
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Humidity	Play=*Yes*	Play=N*o*
High	3/9	4/5
Normal	6/9	1/5

Outlook	Temperature	Humidity	Wind	PlayTennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Wind	Play=*Yes*	Play=*No*
Strong	3/9	3/5
Weak	6/9	2/5

Example

- Learning Phase

Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

Temperature	Play=*Yes*	Play=*No*
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5

Humidity	Play=*Yes*	Play=*No*
High	3/9	4/5
Normal	6/9	1/5

Wind	Play=*Yes*	Play=*No*
Strong	3/9	3/5
Weak	6/9	2/5

***P*(Play=*Yes*) = 9/14**

***P*(Play=*No*) = 5/14**

P(Wind=Strong/Play=*No*) = 3/5

P(Outlook=Sunny/Play=*Yes*) = 2/9

What if we have 4 classes: C0-C3? [play = yes/no/maybe/NoRec

What if we had 10 attributes: A0-A9? [Add: sick, count_ready, fee,]

Example

- Test Phase
  - Given a new instance, predict its label:

    x'=(Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)

  We compute: (Remember: Naiveté: independence):

  P(Play=Yes | Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)=
  P(Play=Yes | Outlook=*Sunny*)*P(Play=Yes | Temperature=*Cool*)*
  P(Play=Yes | Humidity=*High*)*P(Play=Yes | Wind=*Strong*)

  P(Play=No | Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)=
  P(Play=No | Outlook=*Sunny*)*P(Play=No | Temperature=*Cool*)*
  P(Play=No | Humidity=*High*)*P(Play=No | Wind=*Strong*)

$$P(C \mid \mathbf{X}) \propto P(\mathbf{X} \mid C)P(C) = P(X_1, \cdots, X_n \mid C)P(C)$$

P(*Yes* | **x'**) ≈ [**P(**Outlook=***Sunny*** | **Play=Yes)***P(**Temperature=***Cool*** | **Play=*Yes*)***

P(Humidity=***High*** | **Play=Yes)*** **P(**Wind=***Strong*** | **Play=*Yes*)]*P(Play=*Yes*)**

P(*No* | **x'**) ≈ [**P(*Sunny* | Play=No) *P(*Cool* | Play=No)*P(*High* | Play=*No*)***

P(*Strong* | Play=*No*)]*P(Play=*No*)

Example

Humidity	Play=*Yes*	Play=*No*
High	3/9	4/5
Normal	6/9	1/5

Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5

- Test Phase
  - Given a new instance, predict its label
    x'=(Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)
  - Look up tables achieved in the learning phrase

P(Outlook=*Sunny*|Play=*Yes*) = 2/9
P(Temperature=*Cool*|Play=*Yes*) = 3/9
P(Huminity=*High*|Play=*Yes*) = 3/9
P(Wind=*Strong*|Play=*Yes*) = 3/9
P(Play=*Yes*) = 9/14

P(Outlook=*Sunny*|Play=*No*) = 3/5
P(Temperature=*Cool*|Play==*No*) = 1/5
P(Huminity=*High*|Play=*No*) = 4/5
P(Wind=*Strong*|Play=*No*) = 3/5
P(Play=***No***) = 5/14

Temperature	**Play=*Yes***	Play=*No*
Hot	**2/9**	2/5
Mild	**4/9**	2/5
Cool	**3/9**	1/5

Wind	**Play=*Yes***	Play=*No*
Strong	**3/9**	3/5
Weak	**6/9**	2/5

P(*Yes*|**x'**) ≈ [P(*Sunny*|Yes)P(*Cool*|*Yes*)P(*High*|Yes)P(*Strong*|*Yes*)]P(Play=*Yes*) = 0.0053
P(*No*|**x'**) ≈ [P(*Sunny*|No) P(*Cool*|No)P(*High*|*No*)P(*Strong*|*No*)]P(Play=*No*) = **0.0206**

Given the fact P(*Yes*|**x'**) < P(*No*|**x'**), we label **x'** to be "*No*".

Advantages/Disadvantages of Naïve Bayes

- Advantages:
  - Fast to train (single scan). Fast to classify
  - Not sensitive to irrelevant features
  - Handles real and discrete data
  - Handles streaming data well
- Disadvantages:
  - Assumes independence of features

SUPERVISED LEARNING

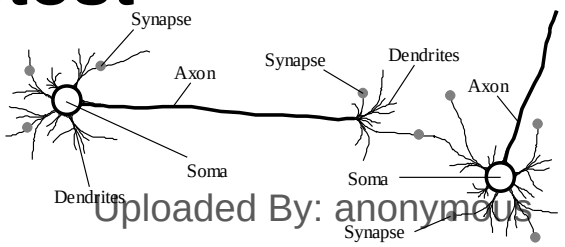
3- Artificial Neural Networks ANN

Introduction: how the brain works

- Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy.
- Learning capabilities can improve the performance of an intelligent system over time.
- The most popular approaches to machine learning is **artificial neural networks**

Neural Networks

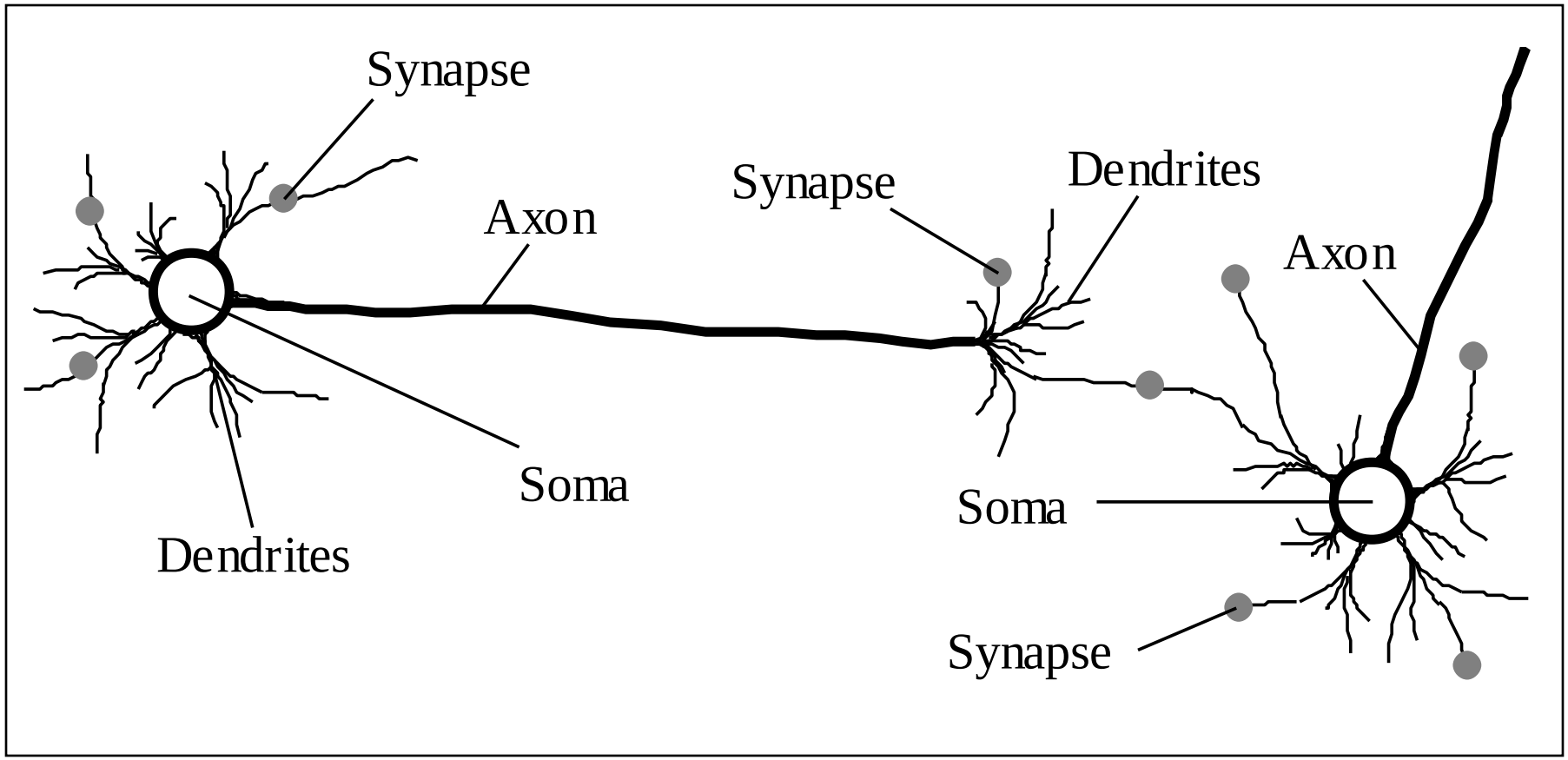
- A **neural network** can be defined as a model of reasoning based on the human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons**.
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, *synapses*, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.



Neural Networks

- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power.
- A neuron consists of a cell body, **soma**, a number of fibers called **dendrites**, and a single long fiber called the **axon**.

Biological neural network

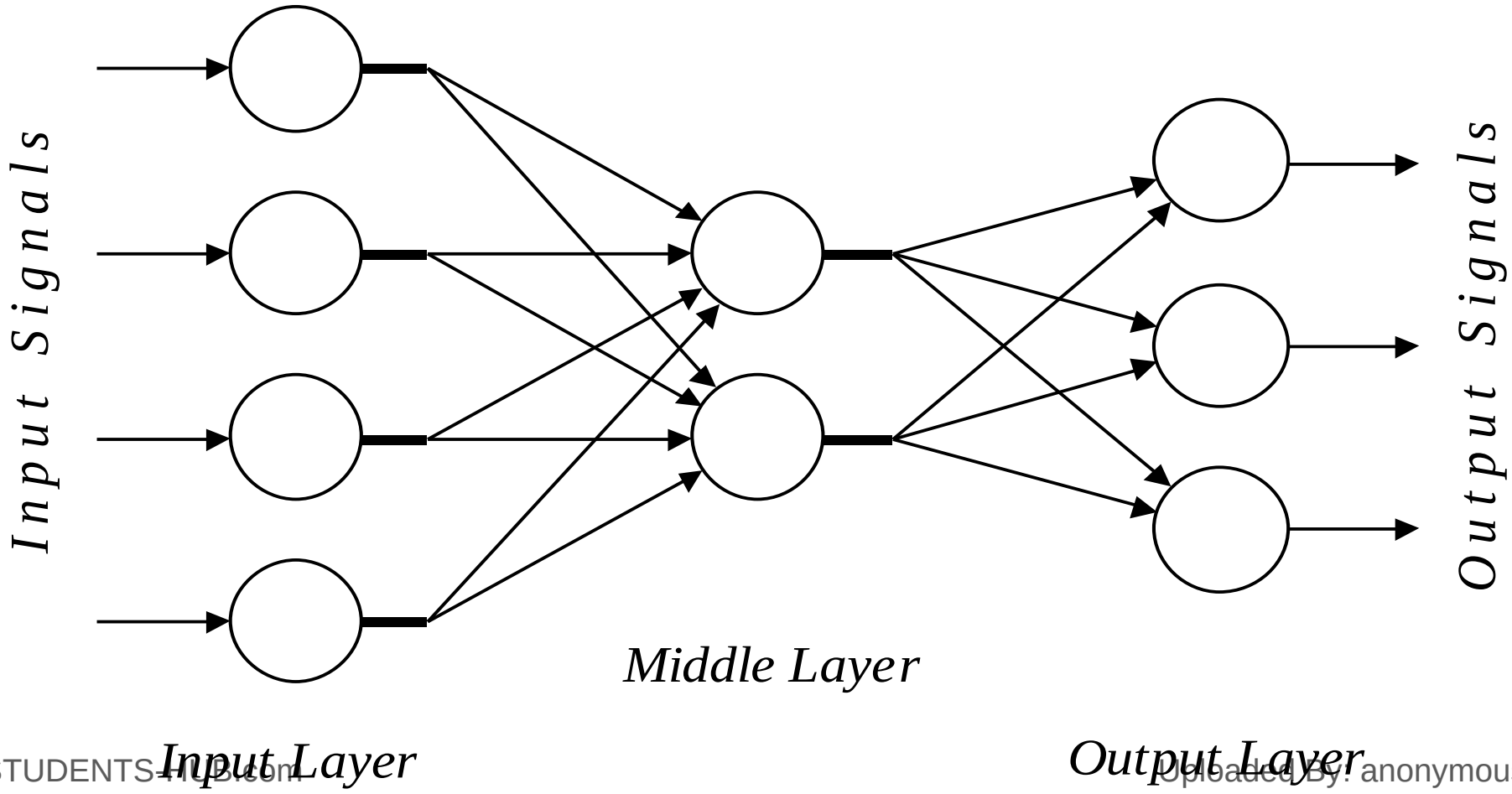


Artificial Neural Network (ANN)

- An artificial neural network consists of a number of very simple **processors**, called **neurons**, analogous to the biological neurons in the brain.
- The neurons are connected by **weighted links** passing signals from one neuron to another.
- The output signal is transmitted through the neuron's outgoing connection.
- The outgoing connection splits into a number of branches that transmit the same signal.
- The outgoing branches terminate as the incoming connections of next layer neurons in the network.

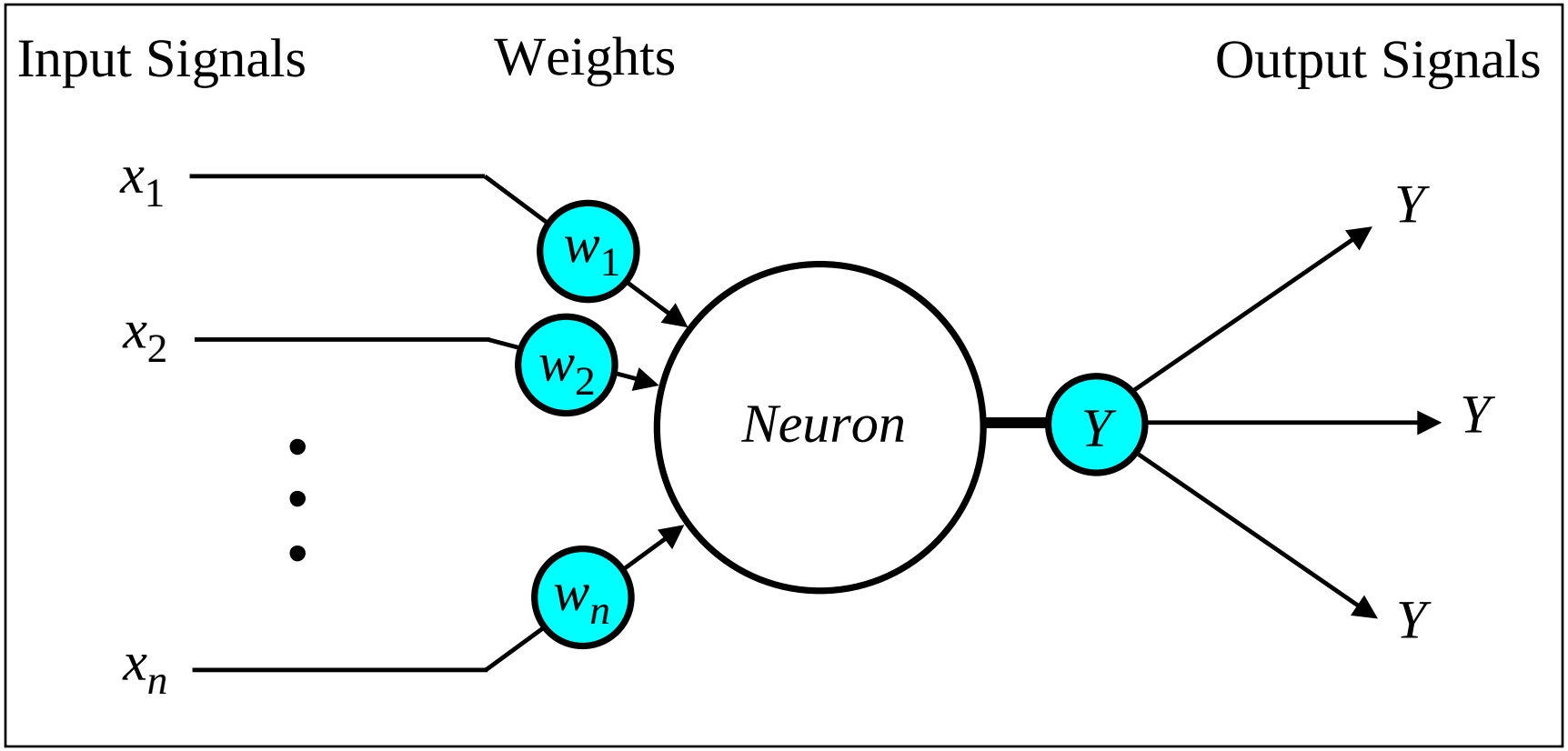
Architecture of a typical artificial neural network

- Here is a video: 19 minuted [handwritten numbers recognition using ANN] https://www.youtube.com/watch?v=aircAruvnKk



The neuron as a simple computing element

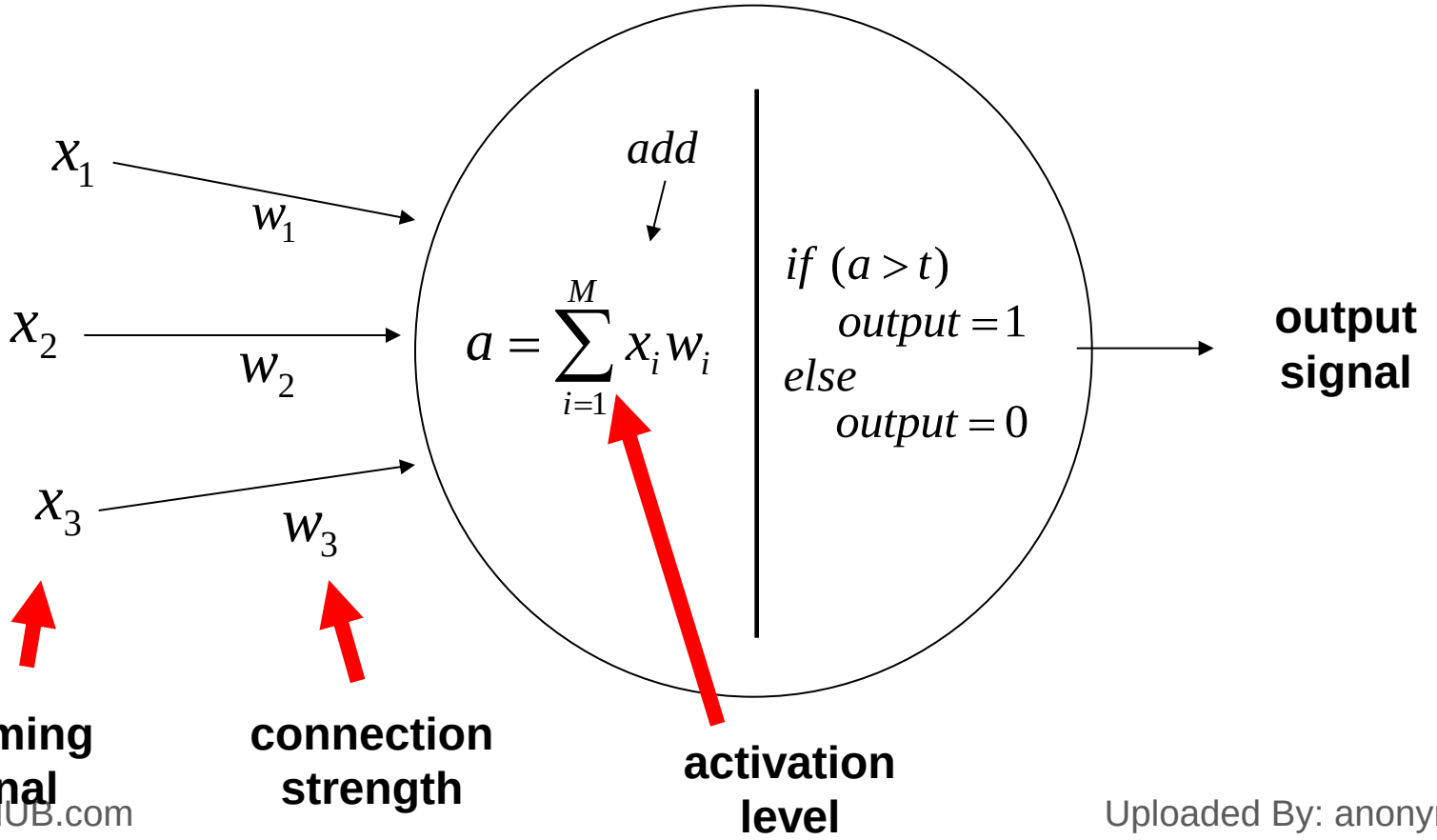
Diagram of a neuron



The neuron as a simple computing element

- input signals ‘x’ and coefficients ‘w’ are multiplied
- weights correspond to connection strengths (how important the input signal there)
- signals are added up. If sum is enough (>t-threshold) , FIRE! (output a 1) else 0 (or -1)

•Default threshold: 0: If sum is >0, FIRE! (output a 1) else 0 (or -1)



Calculation…

$$a = \sum_{i=1}^{M} x_i w_i$$

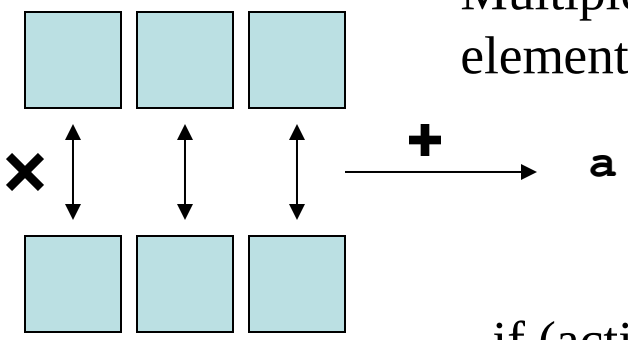
Sum notation, (just like a loop from 1 to M). Linear sum: a line in 2- dimensional space, a plane in 3- dimentional space and so on

Question: When an input has no INFLUENCE?

Answer: x=0 or w=0 for that input!!

`double[] x =`

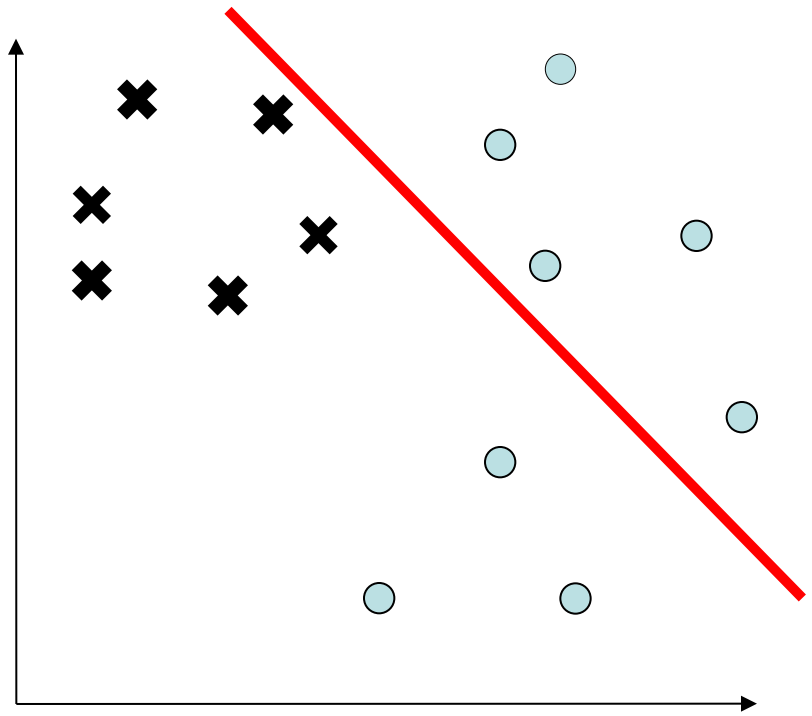
`double[] w =`



Multiple corresponding elements and add them up

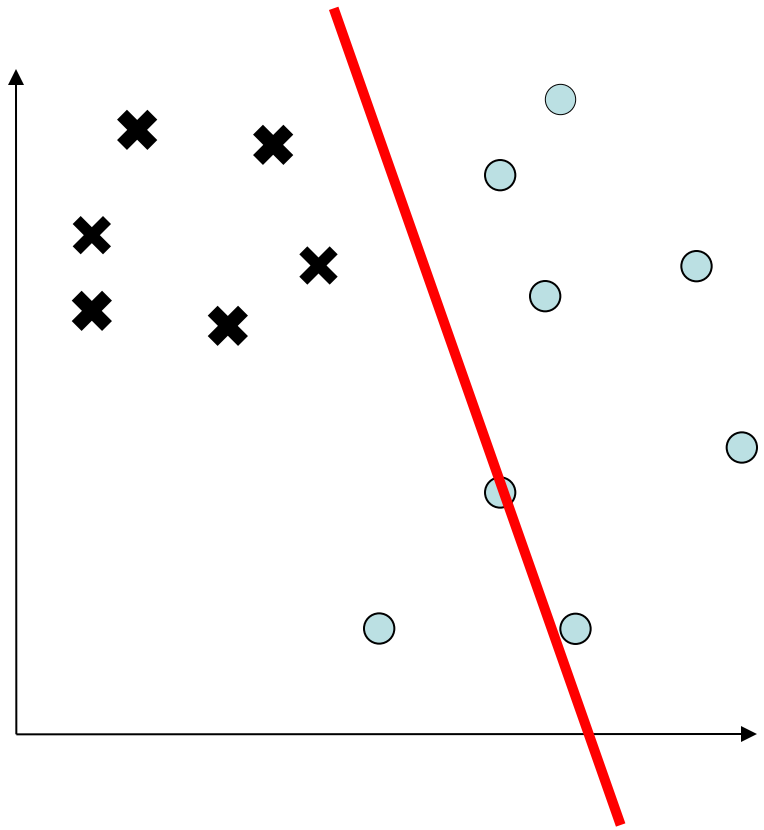
(activation)

if (activation > threshold) FIRE !



Is this a good decision boundary? **Weights:** w_1, w_2 **and t**

$$\textit{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$

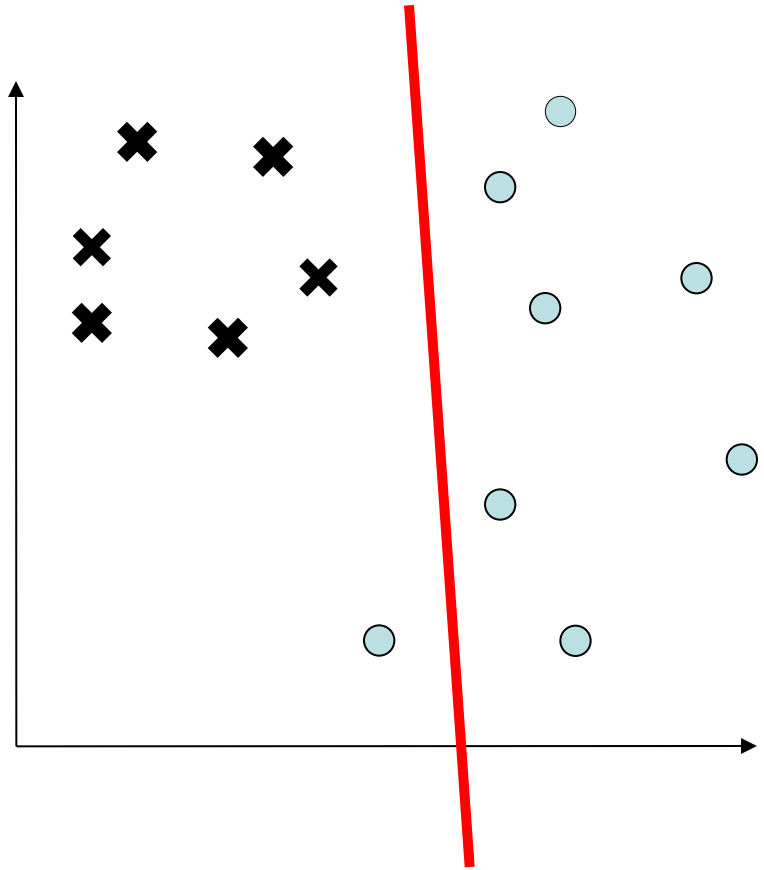


$w_1 = 2.1$

$w_2 = 0.2$

$t = 0.05$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) > t \quad \text{then } output = 1, \text{ else } output = 0$$

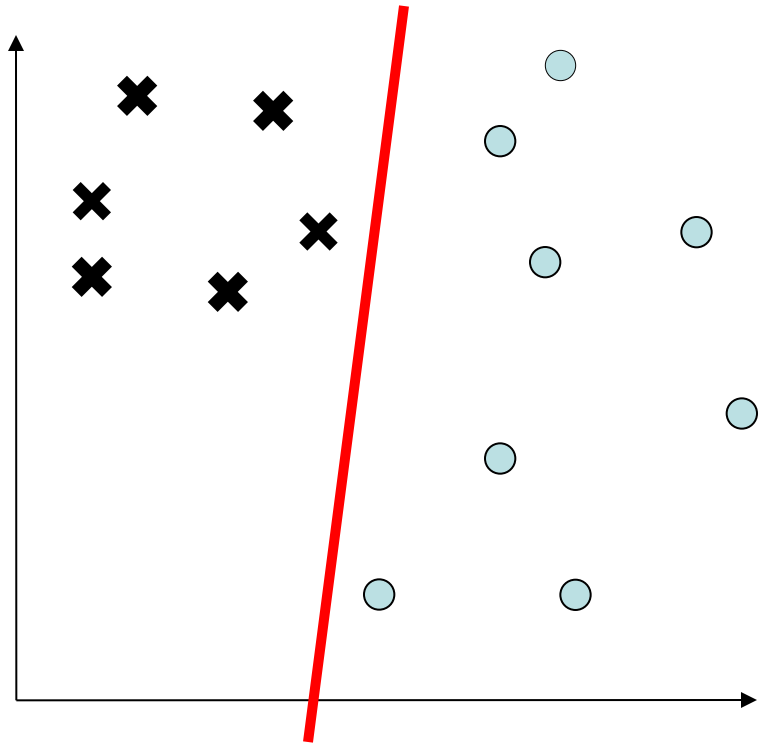


$w_1 = 1.9$

$w_2 = 0.02$

$t = 0.05$

$$\text{if} \quad \left(\sum_{i=1}^{M} x_i w_i \right) \quad > \quad t \qquad \text{then } output = 1, \text{ else } output = 0$$



$w_1 = -0.8$

$w_2 = 0.03$

$t = 0.05$

Changing the weights/threshold makes the decision boundary move.

Pointless / impossible to do it by hand – only ok for simple 2-D case.

We need an algorithm….

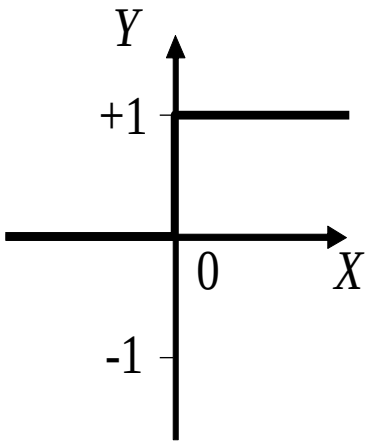
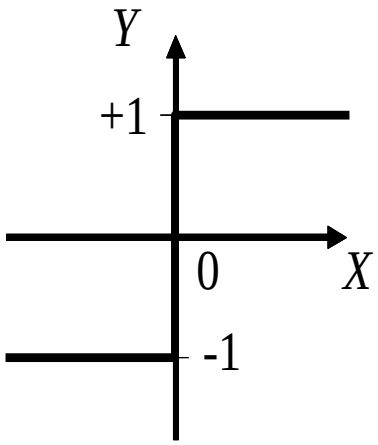
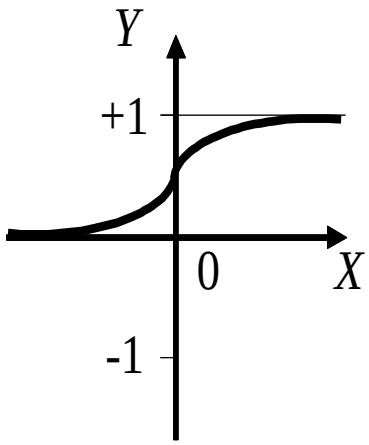
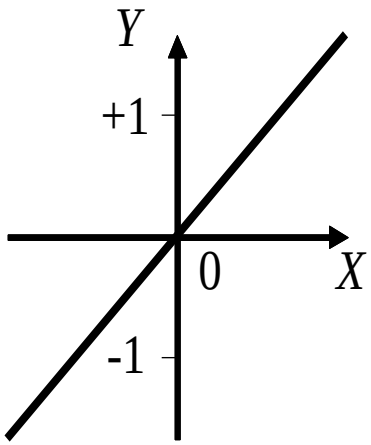
The neuron a simple computing element

- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net output is less than the threshold, the neuron output is –1. But if the net output is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**:

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}$$

- This activation function is called a **sign function**.

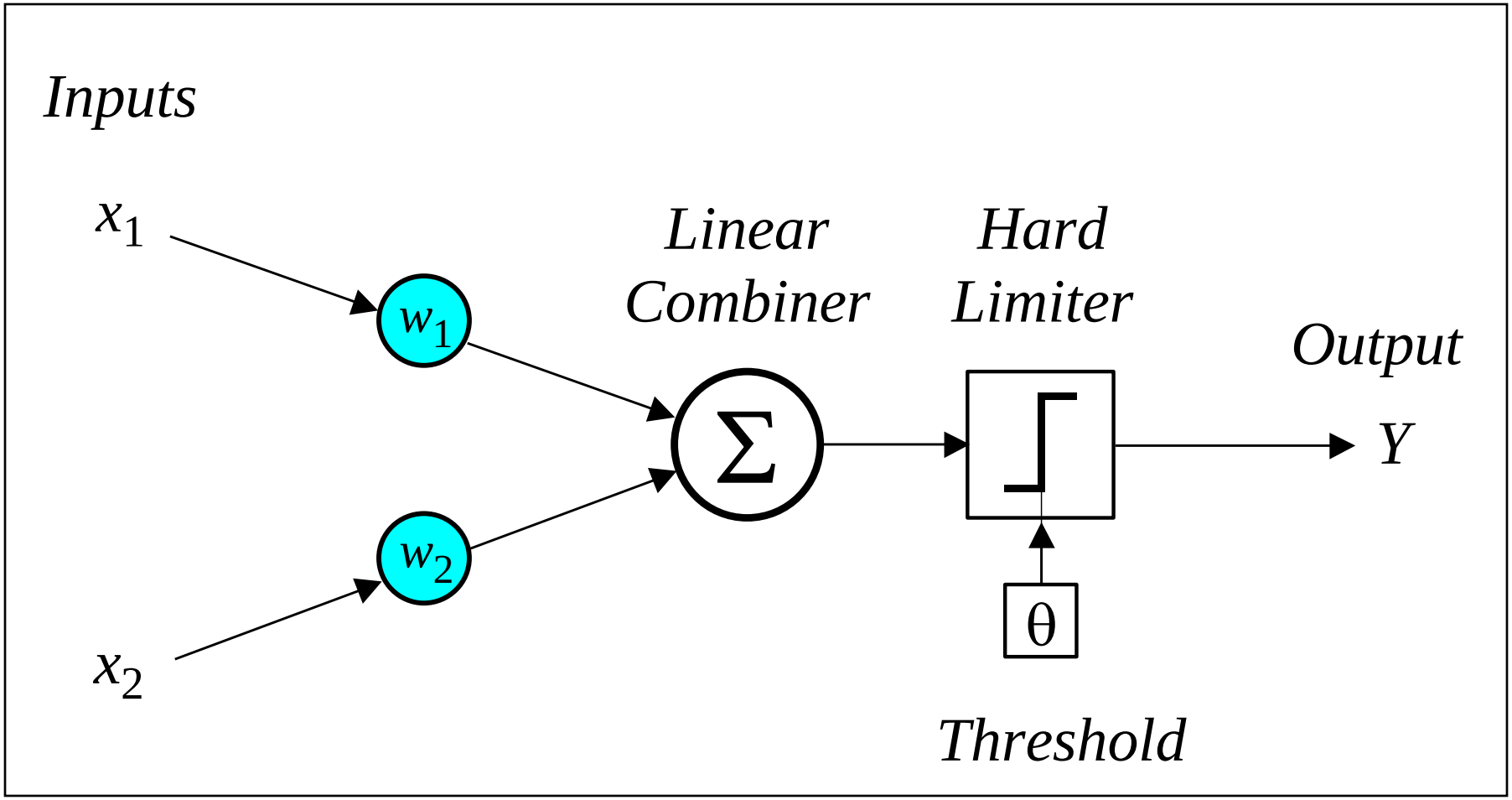
Activation functions of a neuron

Step function	*Sign function*	*Sigmoid function*	*Linear function*
			
$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, \text{if } X \geq 0 \\ -1, \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \frac{1}{1+e^{-X}}$	$Y^{linear} = X$

Can a single neuron learn a task?

- In 1958, **Frank Rosenblatt** introduced a training algorithm that provided the first procedure to train a simple ANN: **perceptron**.
- The ***perceptron*** is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a *hard limiter*.

Single-layer two-input perceptron



The Perceptron

- The operation of Rosenblatt's perceptron is based on the **McCulloch and Pitts neuron model**. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and –1 if it is negative.

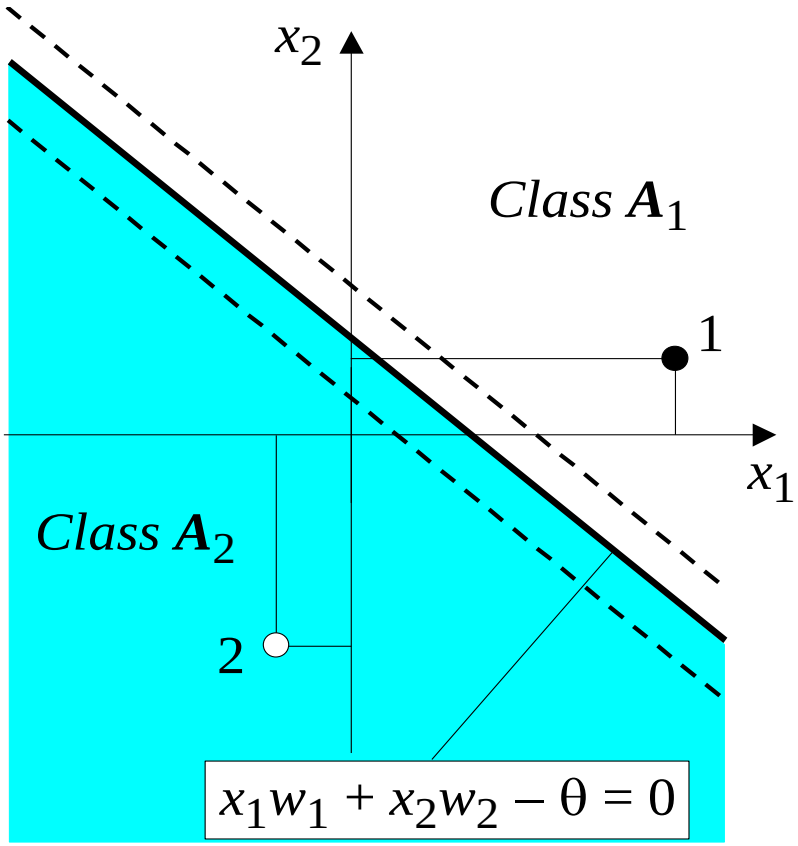
The Perceptron

- The perceptron classifies inputs, $x_1, x_2, \ldots, x_n$, into one of two classes, say $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$.
- In the case of an elementary perceptron, the n-dimensional space is divided by a *hyperplane* into two decision regions. The hyperplane is defined by the ***linearly separable* function**:

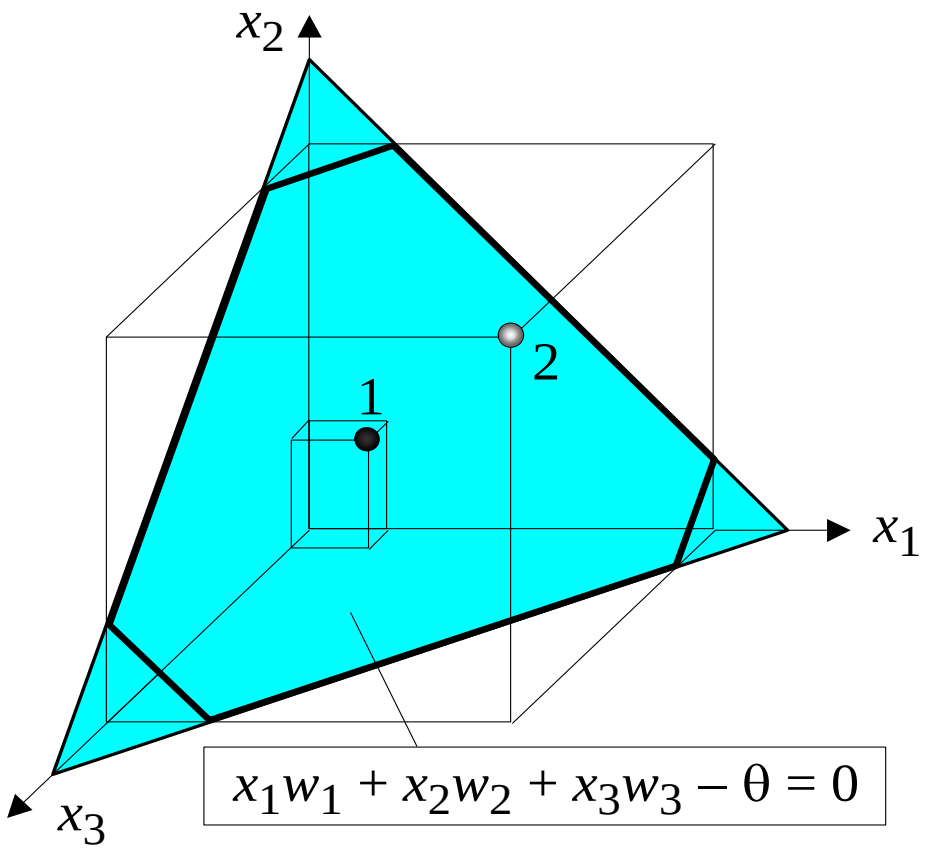
$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

- How many inputs: 1(line/point)? 2(surface/line)? 3 (volume/surface)? 4??? Not, And, Majority,…
- $x_1\ w_1$ $-\theta \rightarrow x_1 * 1$ $-0.5 \rightarrow$ function?
- $x_1\ w_1$ $-\theta \rightarrow x_1 * -1$ $+0.5 \rightarrow$ function?

Linear separability in the perceptrons



(*a*) Two-input perceptron.



(*b*) Three-input perceptron.

How does the perceptron learn its classification tasks?

- We know the desired result: supervised learning!. We know the Perceptron actual output, we compute the difference (ERROR)!
- Make small adjustments in weights to reduce the difference between the actual and desired outputs of the perceptron (reduce error) until OK
- The initial weights are randomly assigned, usually in the range [–0.5, 0.5], and then updated to obtain the output consistent with the training examples.

How does the perceptron learns?

- If at iteration p, the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by (positive or negative error):

$$e(p) = Y_d(p) - Y(p)$$ where $p = 1, 2, 3, \ldots$

Iteration p here refers to the p-th training example presented to the perceptron.

- If the error, $e(p)$, is **positive** (actual too low), we need to **increase** perceptron output $Y(p)$, but if it is **negative**, we need to **decrease** $Y(p)$.
- Increase/Decrease by HOW MUCH?

The perceptron learning rule

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$

where p = 1, 2, 3, . . .

α is the **learning rate**, a positive constant <1

Note that adjustment also depends on x_i: if x_i =0, no contribution/no adjustment

The perceptron learning rule was first proposed by **Rosenblatt** in 1960.

Using this rule we can derive the perceptron training algorithm for classification tasks.

Example: AND Gate training:

x1	x2	Yd
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron Training Algorithm

Step 1: Initialization

Set initial weights w_1, w_2,…, w_n and threshold θ to random numbers in the range [–0.5, 0.5].

(any numbers will do but conversion time may be quite high).

Perceptron Training Algorithm (continued)

Step 2: Activation

- Given the desired output $Y_d(p)$, activate the perceptron by applying inputs $x_1(p), x_2(p),\ldots,x_n(p)$ Calculate the **actual** output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and *step* is a step activation function.

- Compute the error: If the error, $e(p)$, is positive, we need to *increase* perceptron output $Y(p)$ [by changing weights], but if it is negative, we need to decrease $Y(p)$ [also by changing weights].

$$e(p) = Y_d(p) - Y(p)$$

Perceptron's Training Algorithm (continued)

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration *p*.

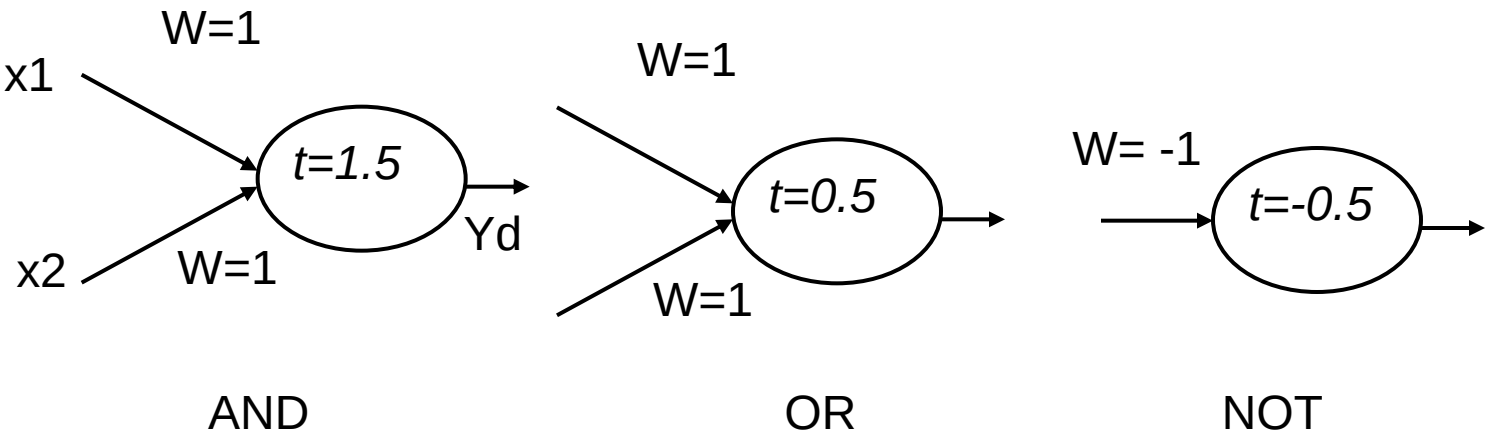
The weight correction is computed by the **delta rule**:

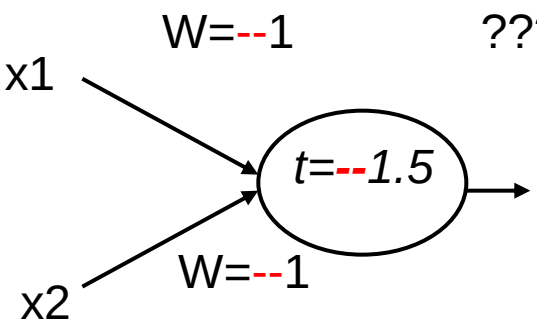
$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

Step 4: Iteration

Increase iteration *p* by one, go back to *Step 2* and repeat the process until convergence using the new weights.

Boolean Gates: unit weights, all w=1, with a step function (0/1)





using a step function

Perceptron Learning Example : Logical *AND*

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

Perceptron Learning: *AND Gate*

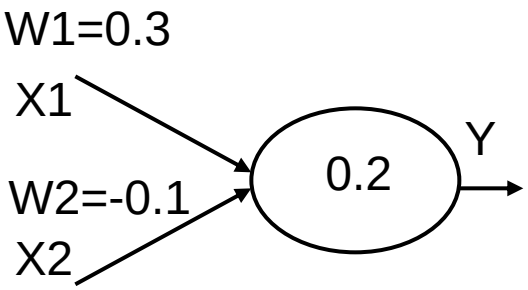
Epoch	x1	x2	Yd	w1	w2	Y	e	w1’	W2’
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.2	-0.1
	1	1	1	0.2	-0.1	0	1	0.3	0.0

$$e(p) = Y_d(p) - Y(p)$$

We have only 2 weights: w1 and w2 initially = 0.3 and -0.1, respectively

Threshold: **θ = 0.2**; learning rate: **α = 0.1**

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$



$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p) \qquad w_i(p+1) = w_i(p) + \Delta w_i(p)$$

Y(p)=0+0-.2=-**0.2**=>0 ➔ e()=yd-y=0 ρ ➔ Δ= 0➔ w1’=w1, w2’=w2 **<0.3,-0.1>**

Y(p)=0+-1*0.1-.2=+0.3=>0 ➔ e()=yd-y=0 ➔ Δ= 0➔ w1’=w1, w2’=w2 **<0.3,-0.1>**

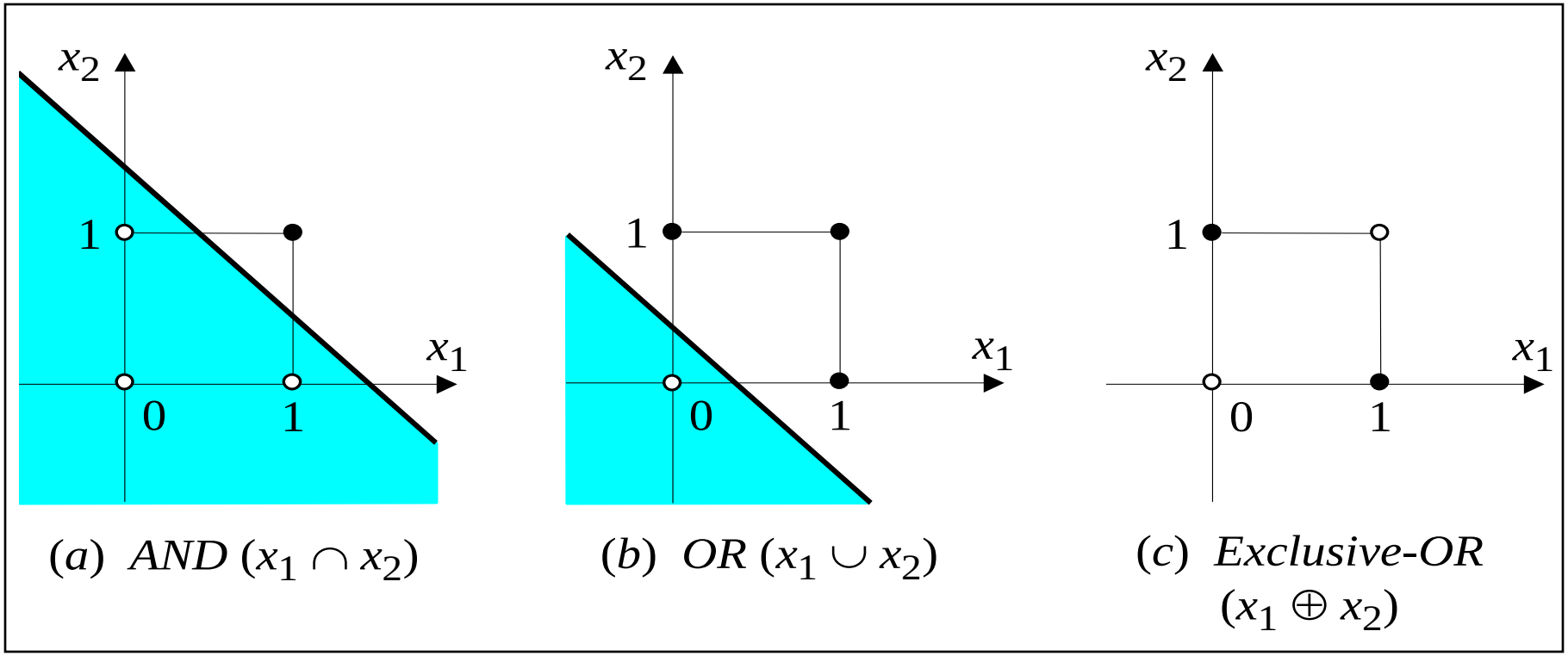
Y(p)=1*0.3+0-.2=0.1=>1 ➔ e()=yd-y=-1 ➔ w1’=w1-0.1=0.2, w2’=-0.1 **<0.2,-0.1>**

Δ1= 0.1*1*-1=-0.1, Δ2= 0

Y(p)=1*0.3+1*-0.1-.2=0=>0 ➔ e()=yd-y=1➔ w1’=0.2+*0.1=0.3, w2’=0.1+0.1*1=0.0: **<0.3,0.0>**

Δ2= 0, Δ1= 0.1*1*1=0.1,

Two-dimensional plots of basic logical operations

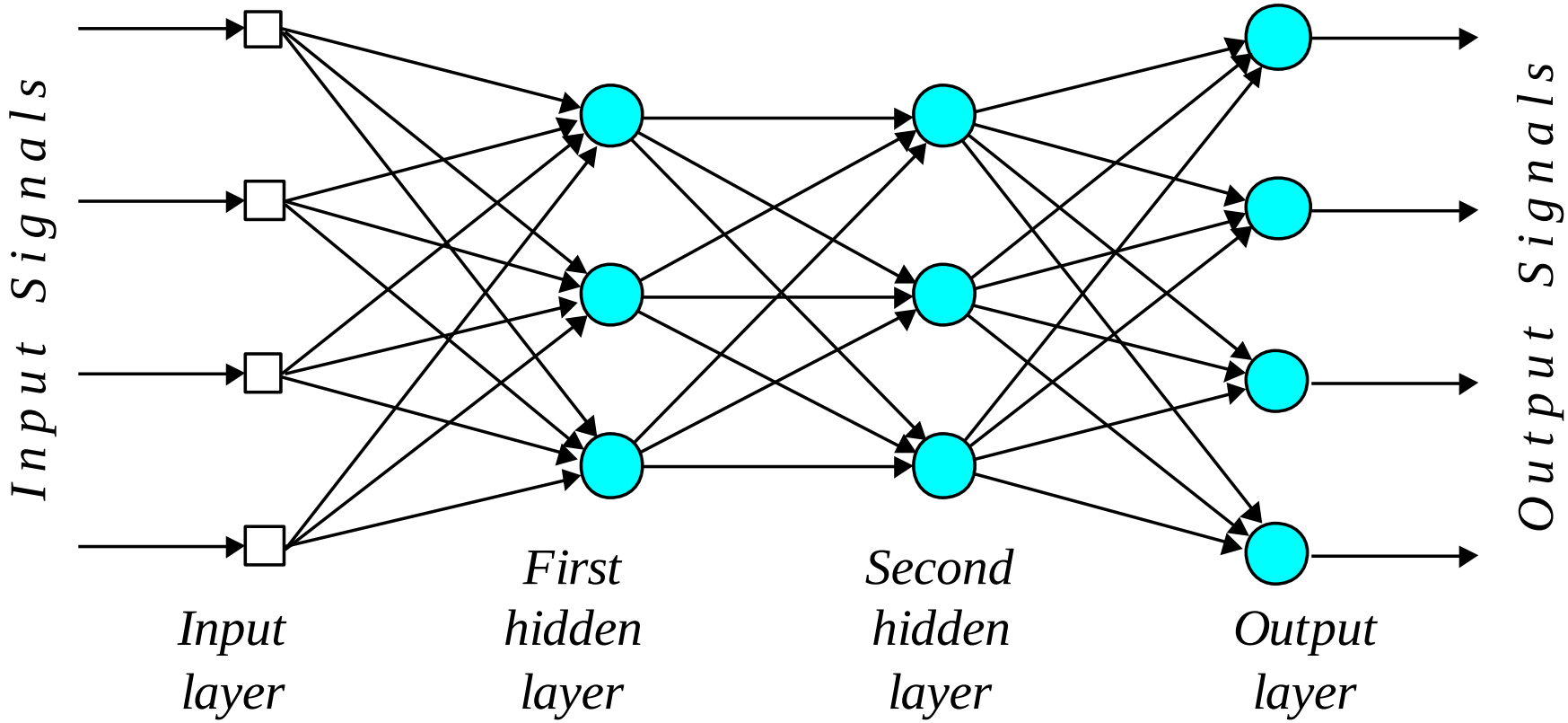


A perceptron can learn the operations *AND* and *OR*, but not *Exclusive-OR*.

Multilayer neural networks

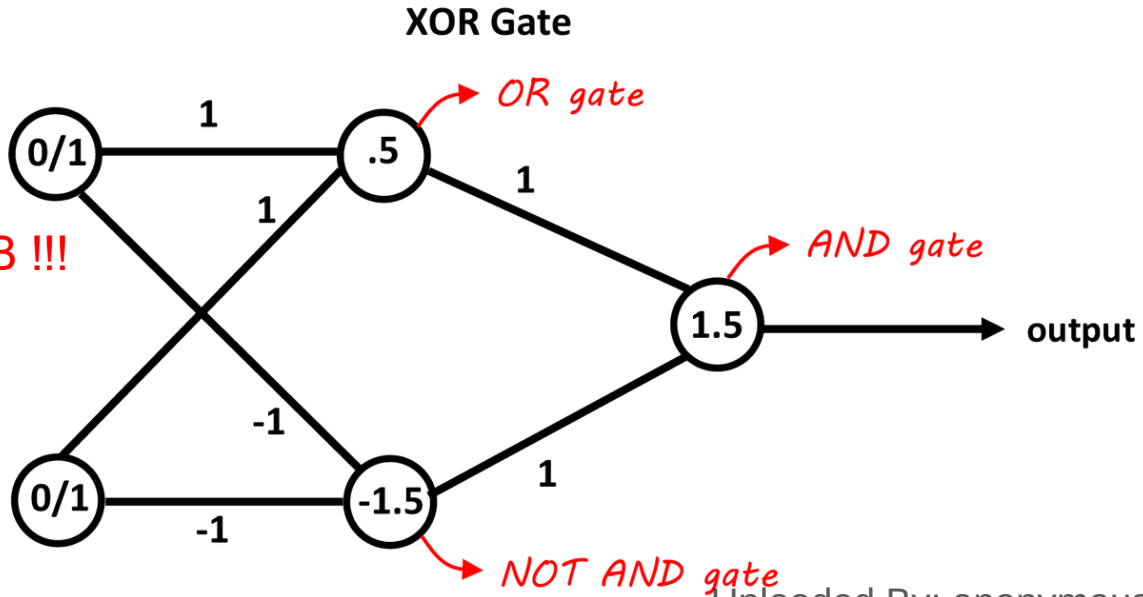
- A multilayer perceptron is a feedforward neural network with one or more hidden layers.
- The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons.
- The input signals are propagated in a forward direction on a layer-by-layer basis.

Multilayer perceptron with two hidden layers



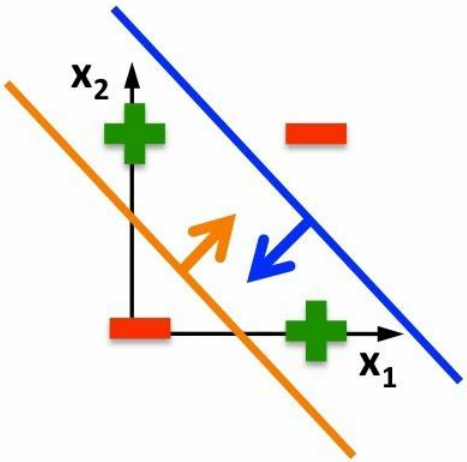
What does the middle layer hide?

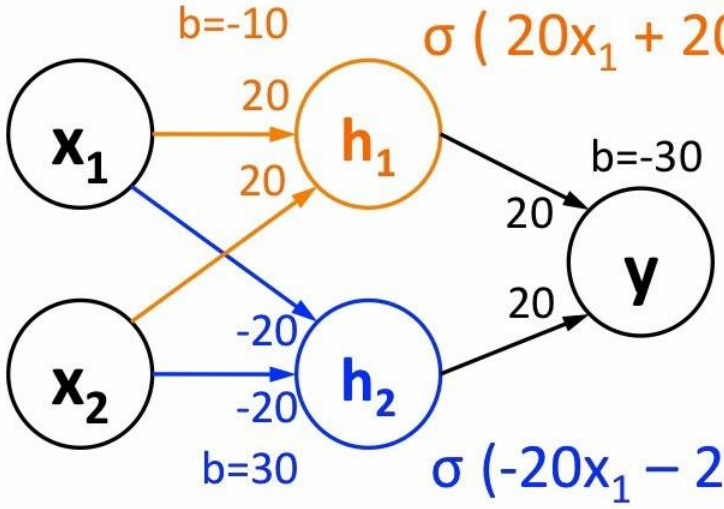
- A hidden layer “hides” its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Neurons in the hidden layer detect the features; the weights of the neurons represent the features hidden in the input patterns. These features are then used by the output layer in determining the output pattern.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have five or even six layers, including three or four hidden layers, and utilise millions of neurons.
- XOR; 3 layers: one hidden
- A XOR B=(A + B)(AB)’=
- (A + B)(A’+B’)=
- AA’+AB’+BA’+BB’=AB’+A’B !!!
- Using step!
- Next Using sugmoid!

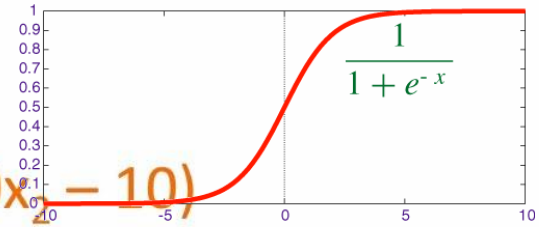


Solving XOR with a Neural Net

Linear classifiers cannot solve this







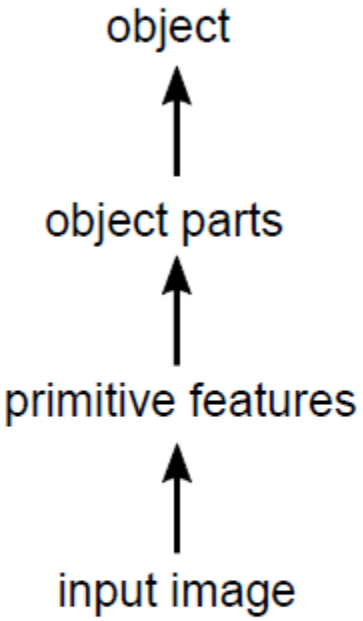
σ(20*0 + 20*0 – 10) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1
σ(20*0 + 20*1 – 10) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1

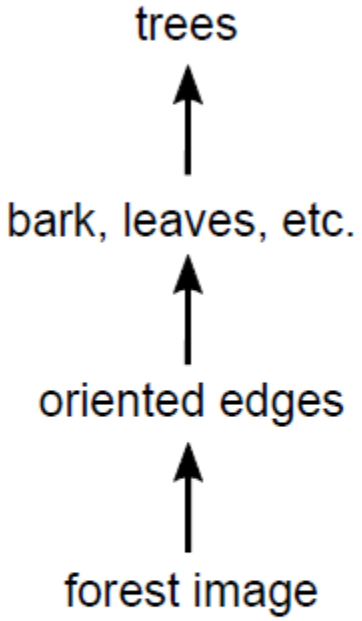
σ (-20*0 – 20*0 + 30) ≈ 1
σ (-20*1 – 20*1 + 30) ≈ 0
σ (-20*0 – 20*1 + 30) ≈ 1
σ (-20*1 – 20*0 + 30) ≈ 1

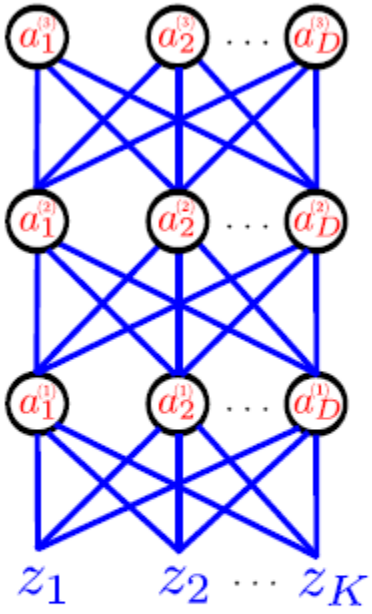
σ (20*0 + 20*1 – 30) ≈ 0
σ (20*1 + 20*0 – 30) ≈ 0
σ (20*1 + 20*1 – 30) ≈ 1
σ (20*1 + 20*1 – 30) ≈ 1

Why use hierarchical multi-layered models?

Argument 1: visual scenes are hierachically organised

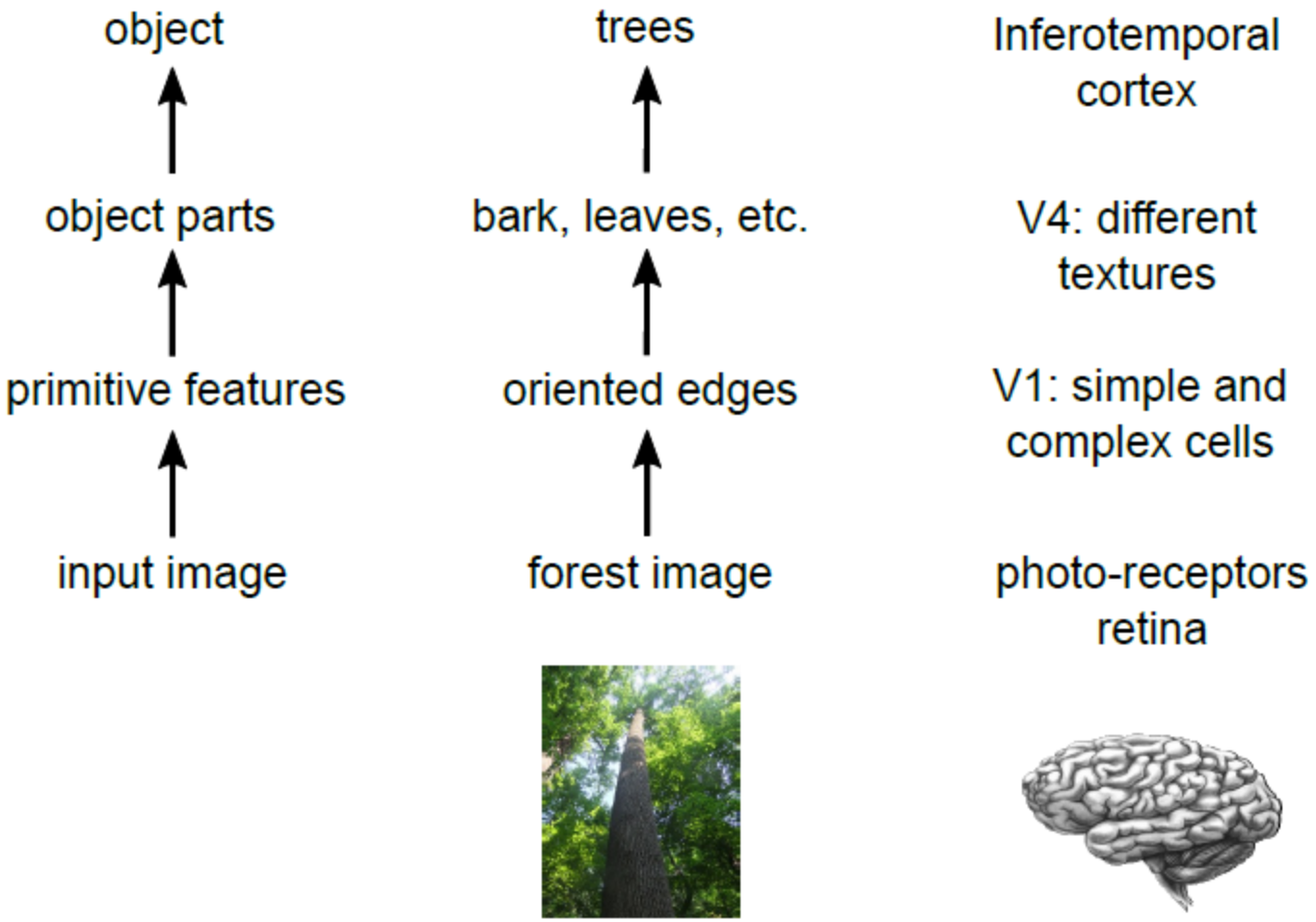




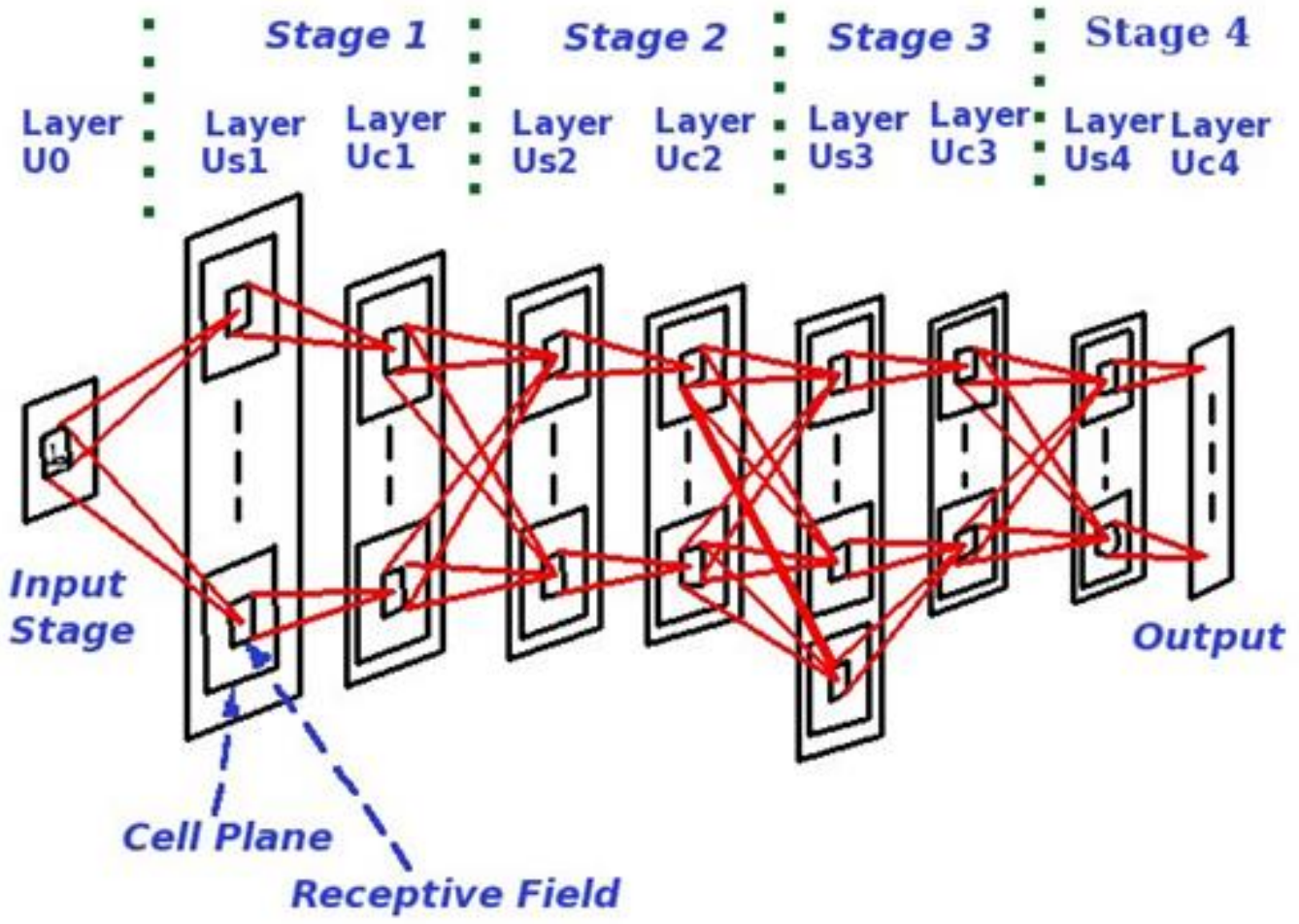


Why use hierarchical multi-layered models?

Argument 2: biological vision is hierachically organised



Neocognitron For Handwritten Text



Back-propagation neural network

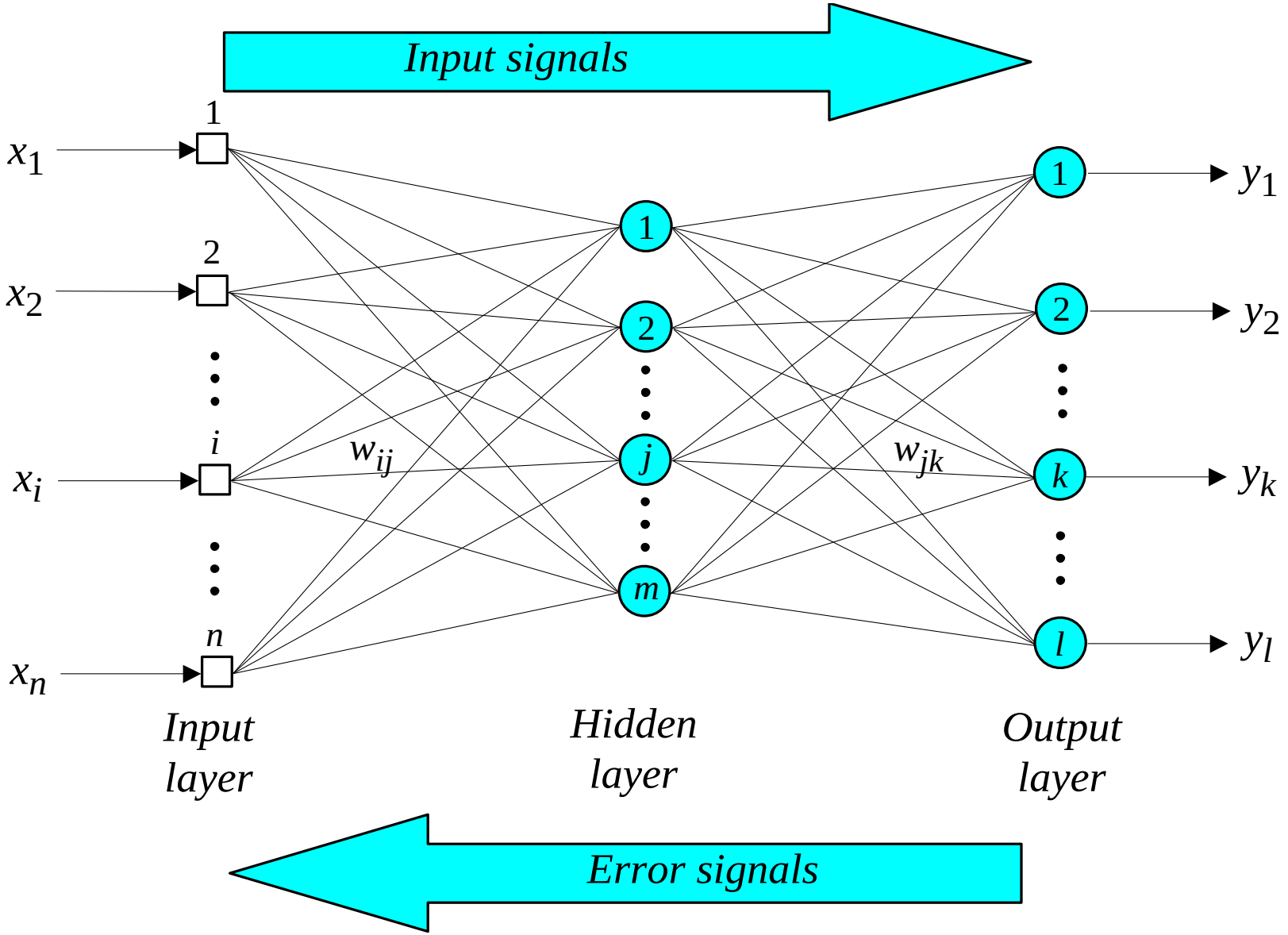
- Learning in a multilayer network proceeds the same way as for a perceptron.
- A training set of input patterns is presented to the network, as usual.
- The network computes its output pattern, and if there is an error – a difference between actual and desired output pattern:
- $$e(p) = Y_d(p) - Y(p)$$

the weights are adjusted to reduce this **error**.

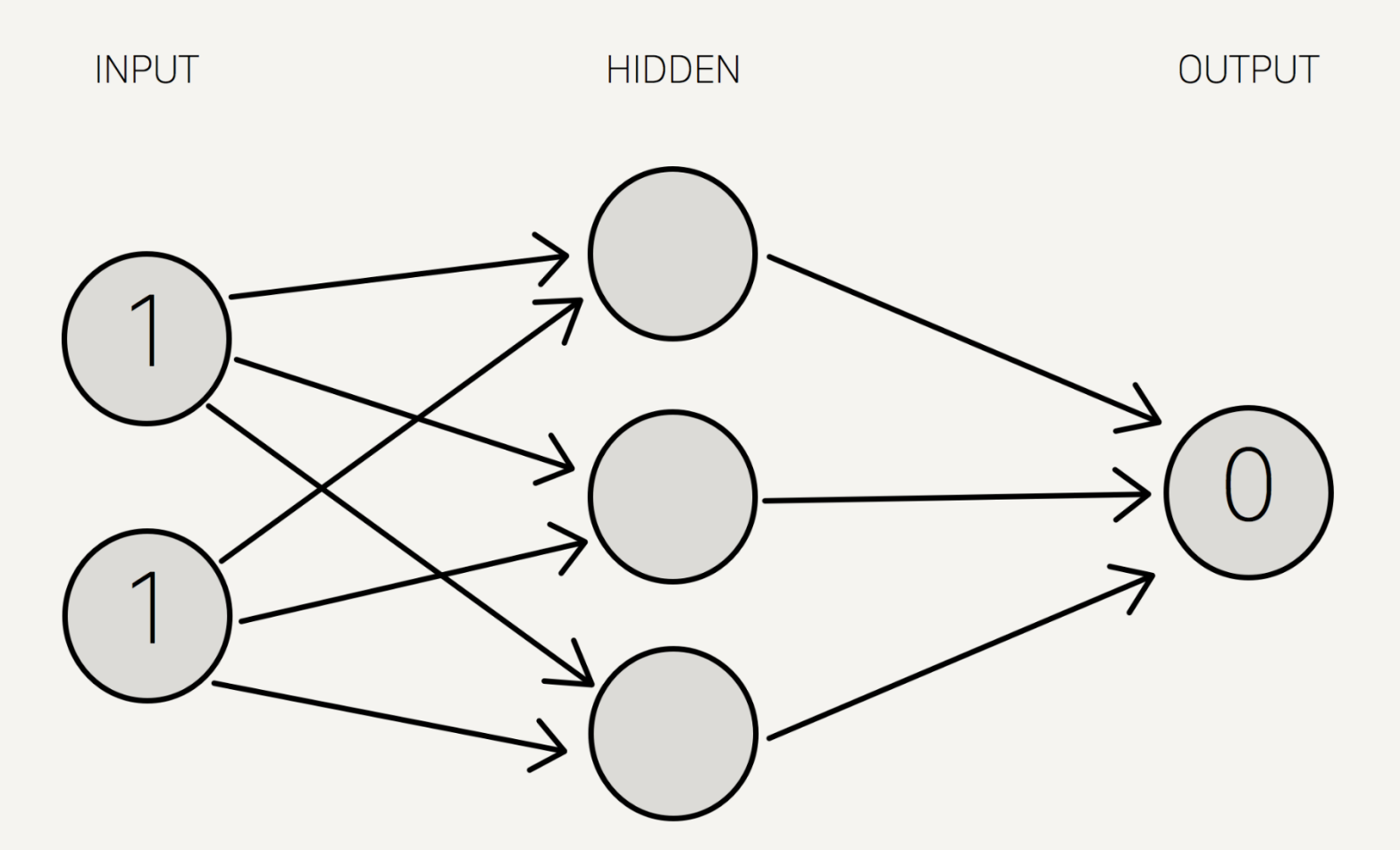
Back-propagation neural network

- In a back-propagation neural network, the learning algorithm has two phases.
  - First,
    - a training input pattern is presented to the input layer.
    - The network propagates **forward** the input pattern from layer to layer until the output pattern is generated by the output layer.
  - Then:
    - If this pattern is different from the desired output, **an error** is calculated [**we see the error only on output!**]
    - Then the error is propagated **backwards** through the network **from the output layer to the input layer**.
    - The weights are modified as the error is propagated.

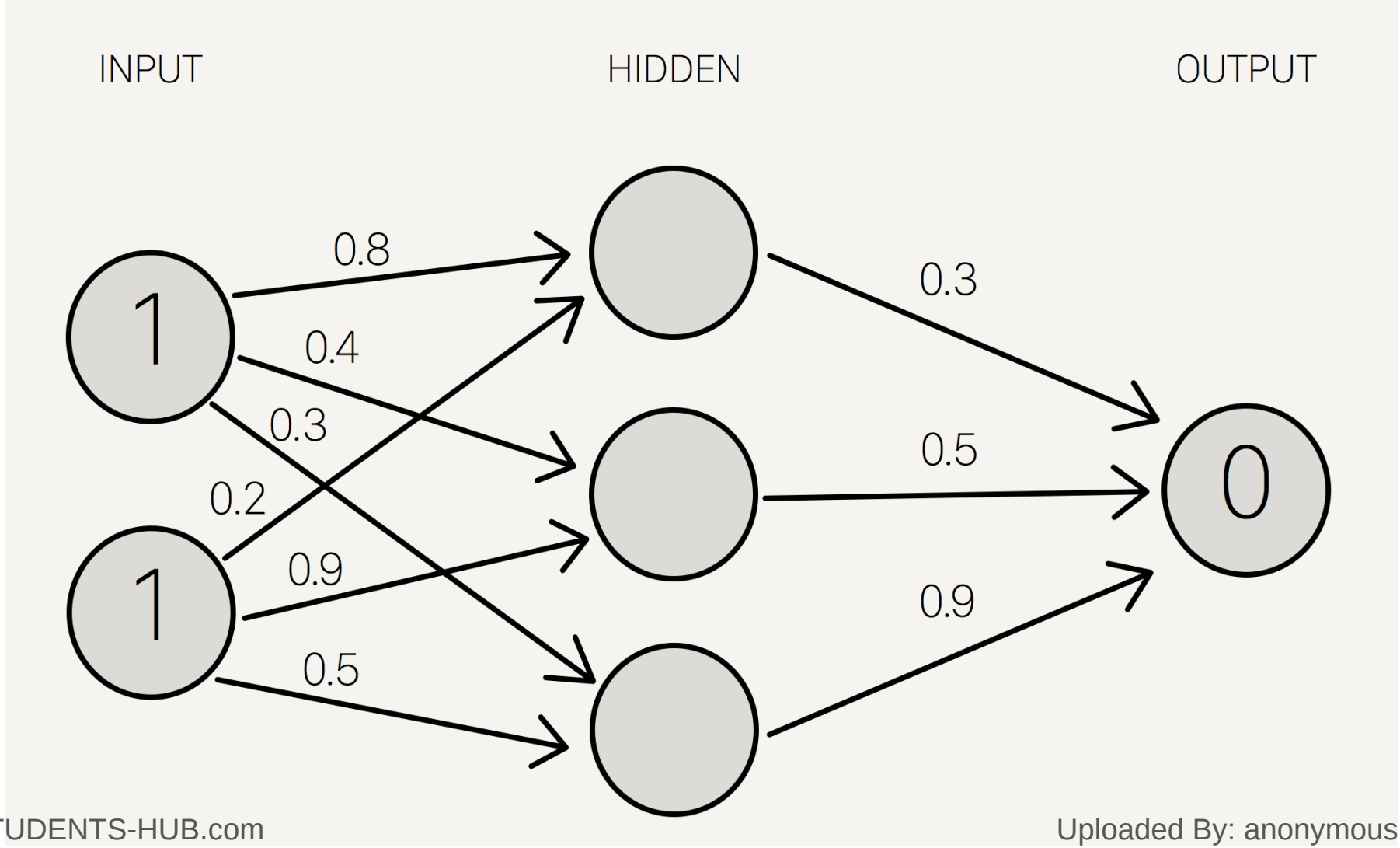
Three-layer back-propagation neural network



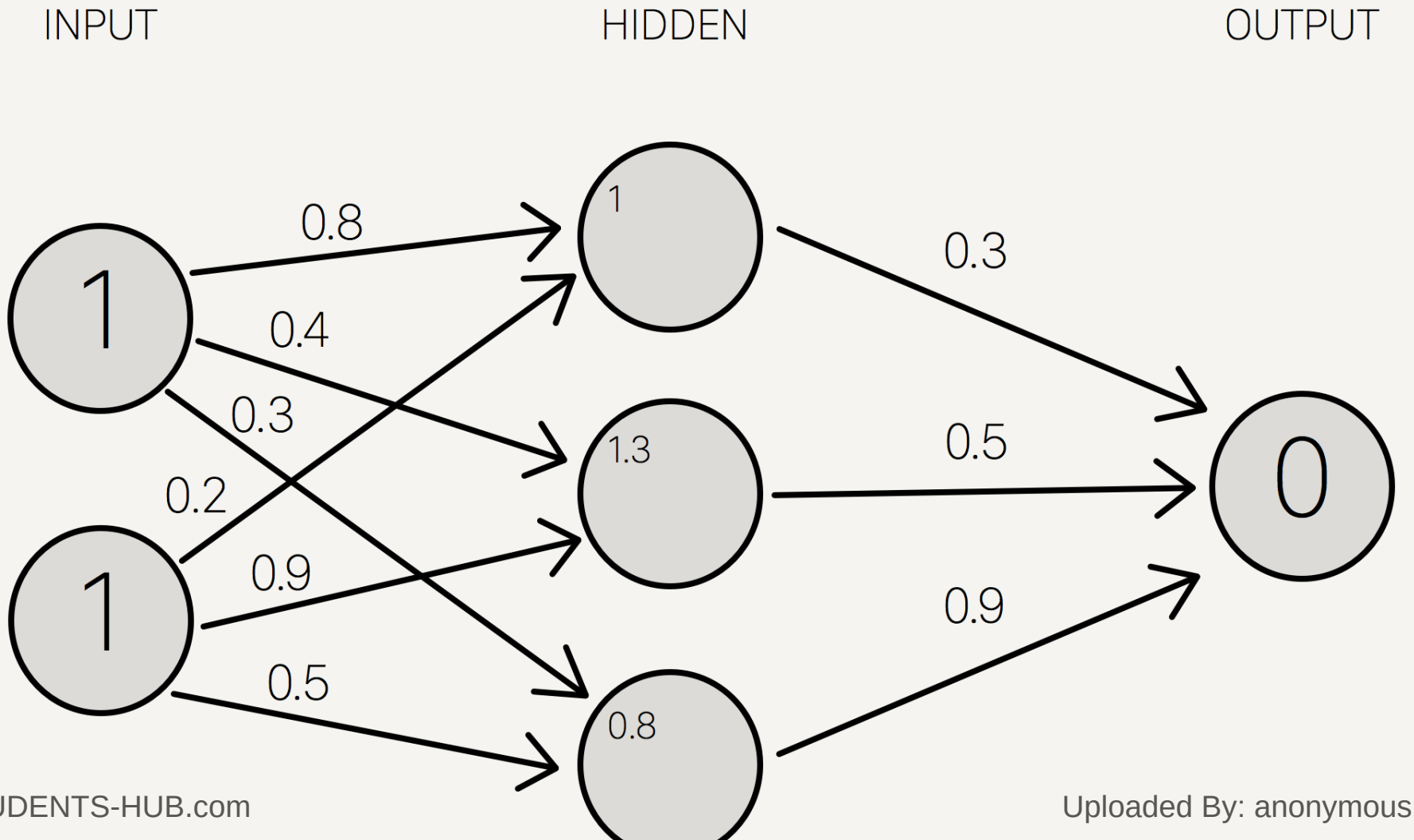
Original



With Weights



Compute Sums: 1 * 0.8 + 1 * 0.2 = 1
1 * 0.4 + 1 * 0.9 = 1.3
1 * 0.3 + 1 * 0.5 = 0.8

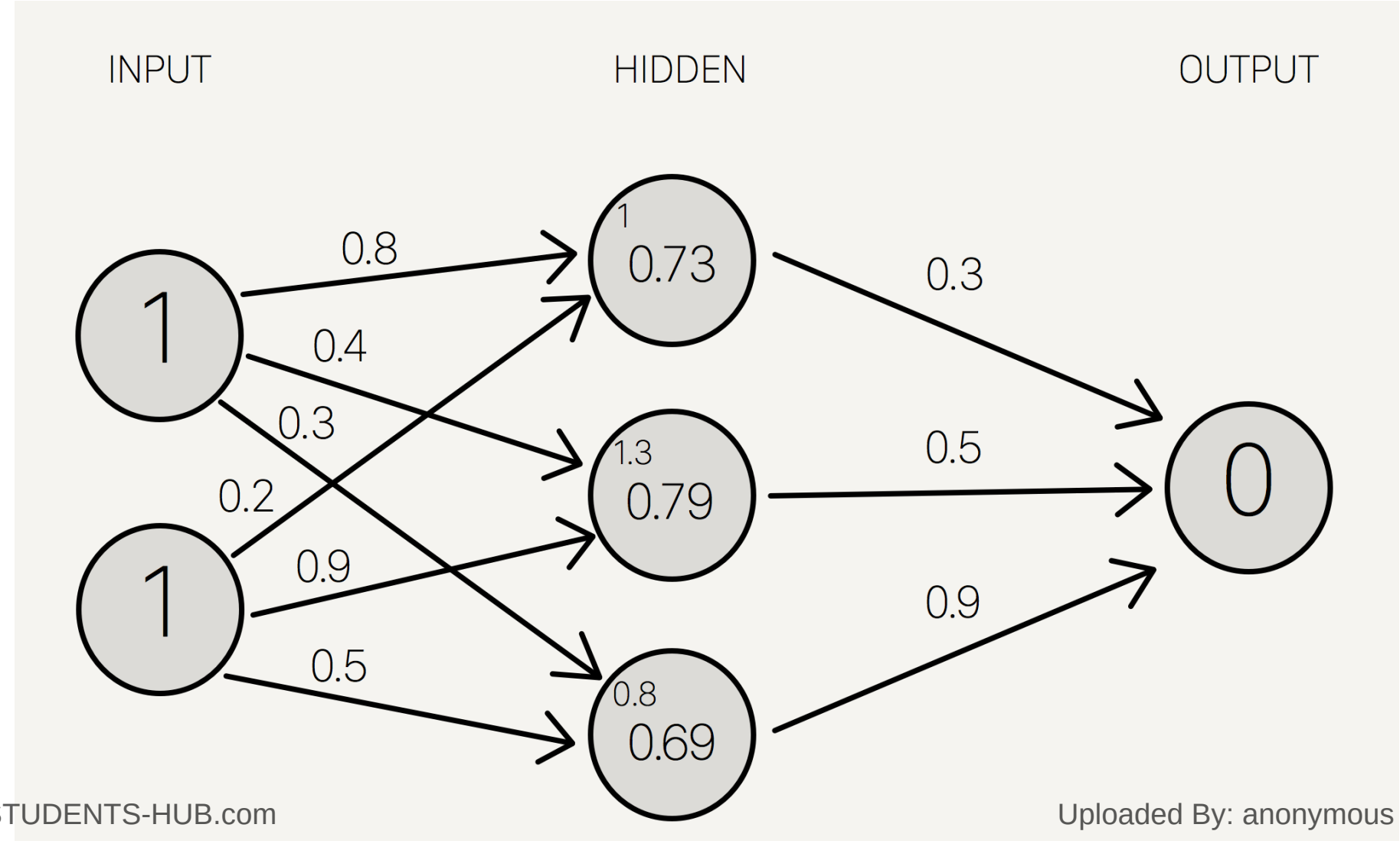


Compute Sigmoids: $S(1.0) = 0.73105857863 \simeq 0.73$

$S(1.3) = 0.78583498304 \simeq 0.79$

$S(0.8) = 0.68997448112 \simeq 0.69$

Sigmoid Calculator online: https://www.vcalc.com/wiki/vCalc/Sigmoid+Function



Compute Sigmoid: S(1.0) = 0.73105857863 ≃0.73

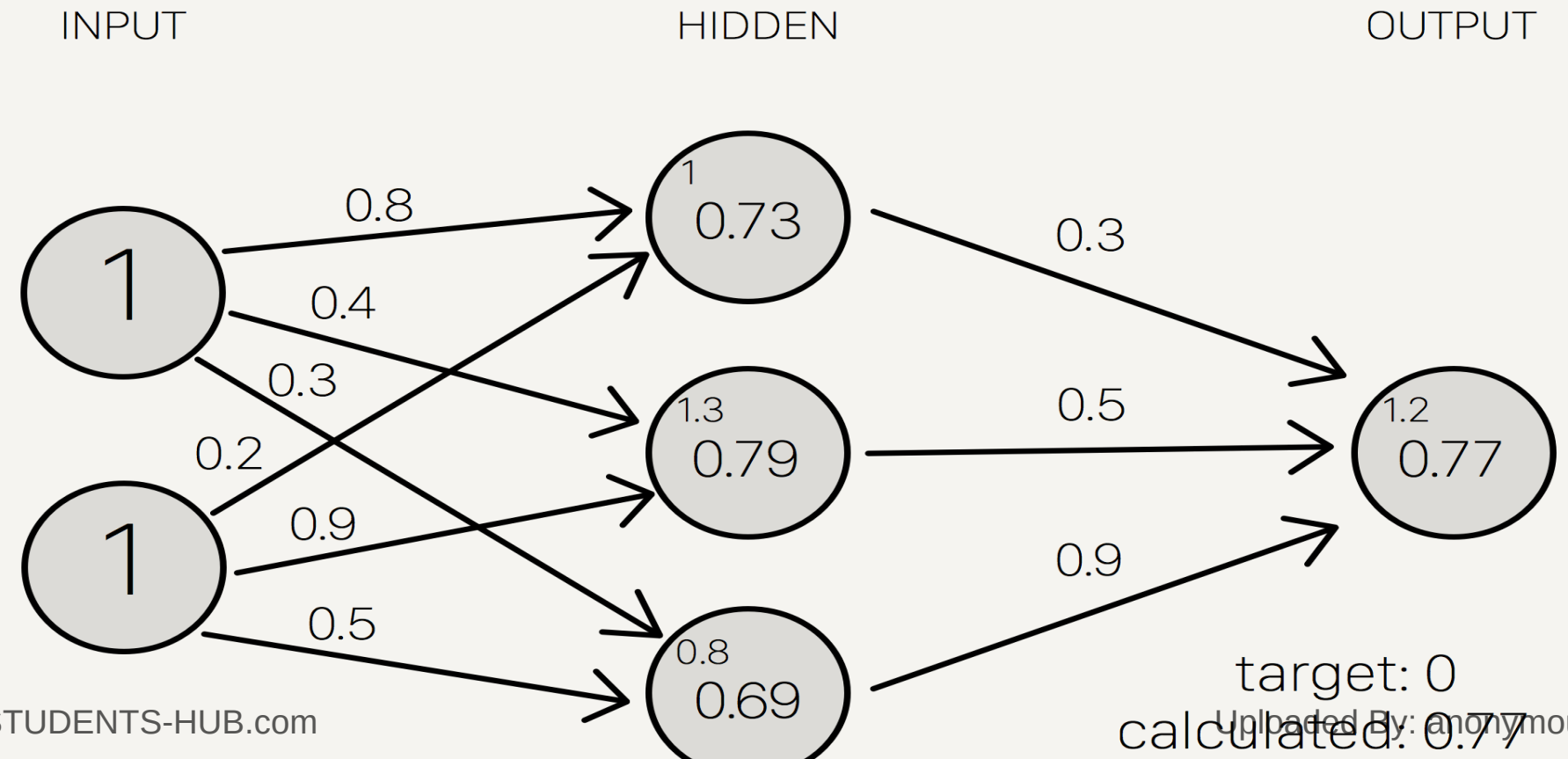
For output layer S(1.3) = 0.78583498304 ≃0.79

S(0.8) = 0.68997448112 ≃0.69

Computing output sum: 0.73*0.3+0.79*0.5+0.69*0.9=1.235≃1.2; **S(1.2)=0.77**

Computing Error: **Error = Target/Desired - calculated =0-0.77= -0.77**

https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/



The back-propagation training algorithm

Step 1: Initialisation

- Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

$$\left(-\frac{2.4}{F_i}, \quad +\frac{2.4}{F_i} \right)$$

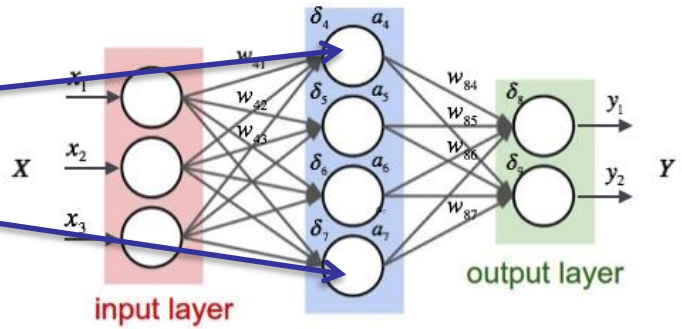
- where F_i is the number of inputs of neuron ***i*** in the network: for a 6 input neuron: weights in [-0.4,0.4].
- The weight initialisation is done on a neuron-by-neuron basis.

Step 2: Activation

- Activate the neural network by applying inputs $x_1(p), x_2(p),\ldots, x_n(p)$ [may have *bias*].
- Using desired outputs $y_{d,1}(p), y_{d,2}(p),\ldots, y_{d,n}(p)$:
  - (*a*) Calculate the actual outputs of the neurons in the **hidden** layer:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p)\cdot w_{ij}(p) - \theta_j\right]$$

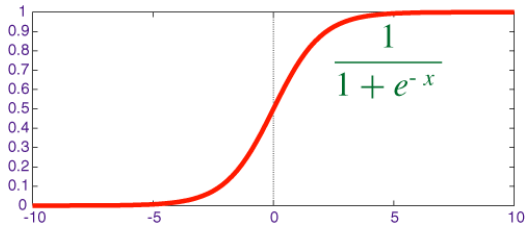
where ***n*** is the number of inputs of neuron ***j*** in the hidden layer, and *sigmoid* is the activation function.



Step 2: Activation (continued)

(*b*) Calculate the actual outputs of the neurons in the **output** layer [1 or many]:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p)\cdot w_{jk}(p) - \theta_k\right]$$



where *m* is the number of inputs of neuron *k* in the output layer.

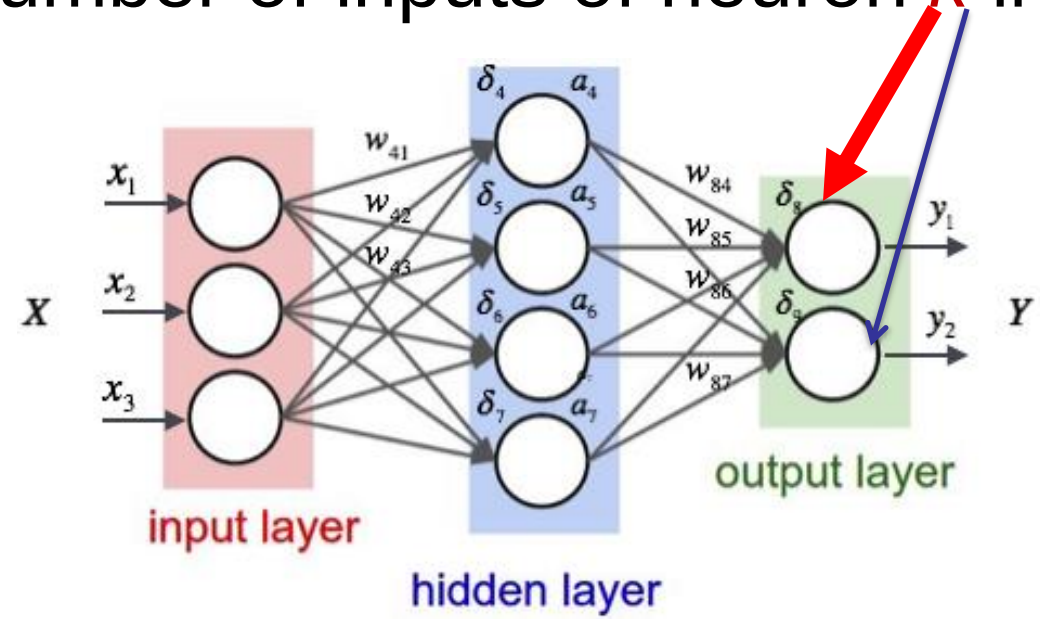
Derivative relative to SUM

Derivative of Sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = y(1-y)$$

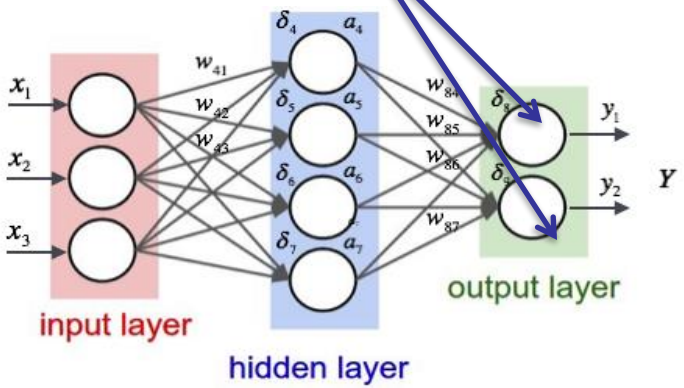


Step 3: Weight training (recall: $\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$)

- Update the weights in the back-propagation network propagating backward the errors associated with output neurons.
  - (*a*) Calculate the error gradient for the neurons in the **output** layer:

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

where $e_k(p) = y_{d,k}(p) - y_k(p)$



Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

y_j is the **Sigmoid** value –output- in the hidden layer into k

Update weights **to** the **output** neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

Derivative of Sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

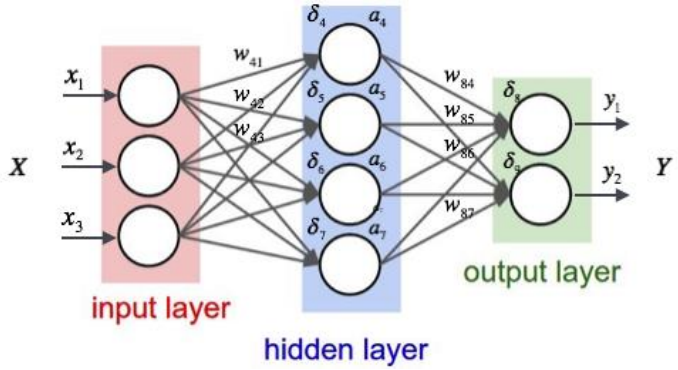
$$= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = y(1-y)$$

Step 3: Weight training (continued)

(*b*) Calculate the error gradient for the neurons in the **hidden** layer (weights still unchanged):

$$\delta_j(p) = y_j(p)\cdot[1-y_j(p)]\cdot\sum_{k=1}^{l}\delta_k(p)\,w_{jk}(p)$$

The sum is for all output neurons connected to the hidden j, L of them. Instead of e(p) we used the gradient of all outputs.



Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha\cdot x_i(p)\cdot\delta_j(p)$$

Update the weights **to** the **hidden** neurons:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

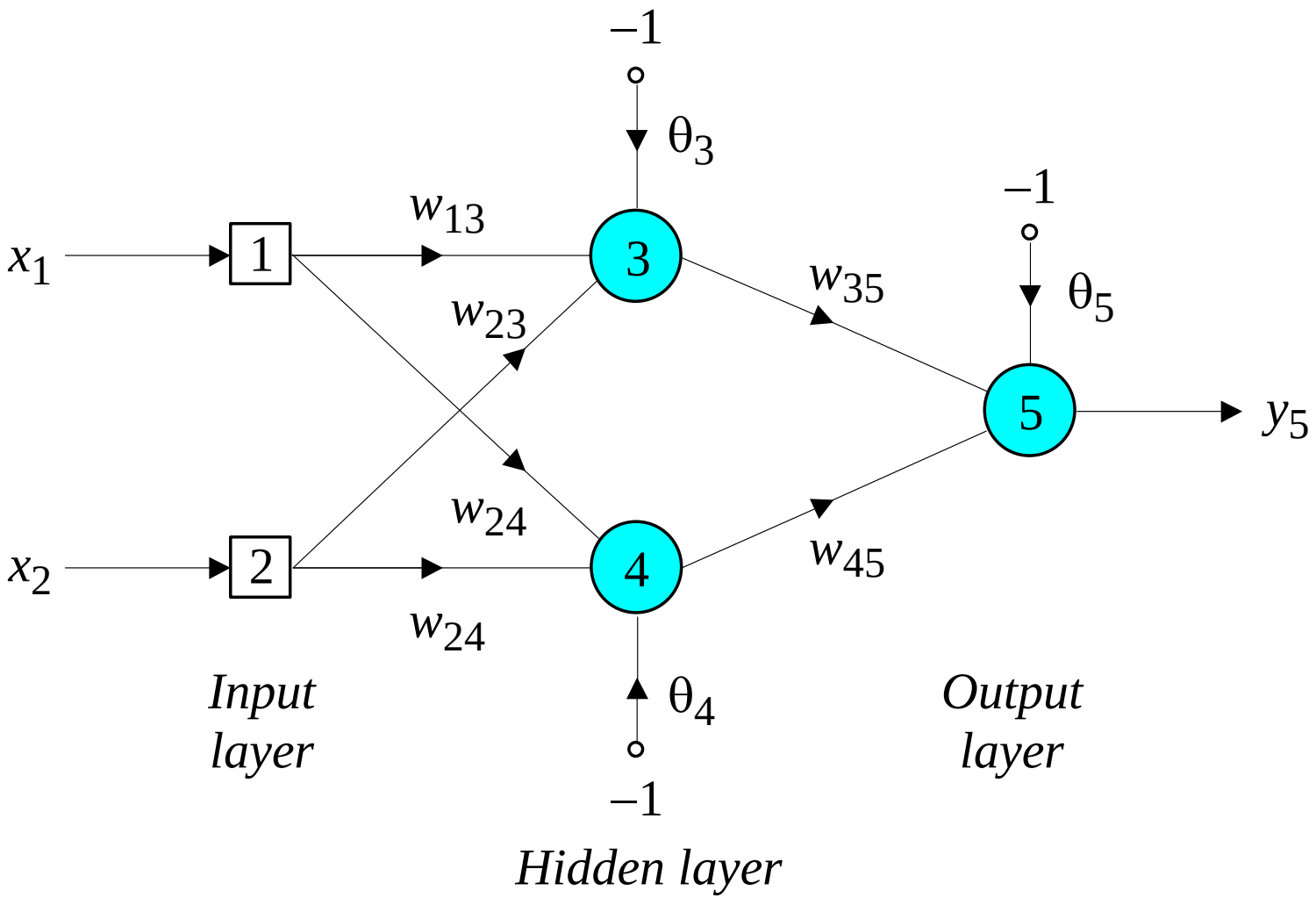
Step 4: Iteration

Let *p=p+1*, go back to *Step 2* and repeat the process until selected error criterion is **satisfied**.

Example:

- As an example, we may consider the three-layer back-propagation network.
- Suppose that the network is required to perform logical operation *Exclusive-OR*.
- Recall: a single-layer perceptron could not do this operation.
- We will apply the three-layer net.

Three-layer network for solving the Exclusive-OR operation



- The effect of the threshold applied to a neuron in the hidden or output layer is represented by its weight, θ, connected to a fixed input (–1 in our case).
- **x_1 and x_2 are 1**, desired output $y_{d,5}$ is 0
- The initial weights and threshold levels are se

randomly: $w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$, $\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.

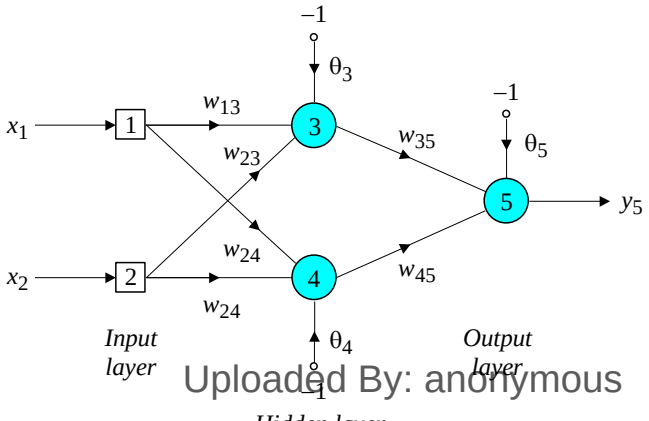
$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

$$y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250\cdot 1.2 + 0.8808\cdot 1.1 - 1\cdot 0.3)}\right] = 0.5097$$

- **y_3=0.5250, y_4=0.8808, y_5=0.5097**
- Thus, the following error is obtained:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$



- Given that x_1 and x_2 are equal to 1 and desired output $y_{d,5}$ is 0. The actual outputs of neurons 3 and 4 in the hidden layer are calculated as

$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

- Now the actual output of neuron 5 in the output layer is determined as:

$$y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250\cdot 1.2 + 0.8808\cdot 1.1 - 1\cdot 0.3)}\right] = 0.5097$$

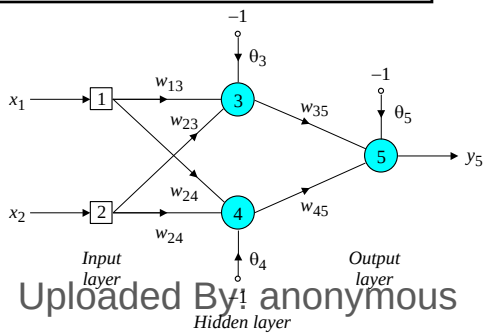
- Thus, the following error is obtained:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

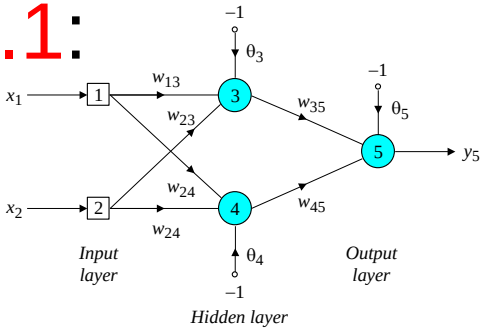
y_3=0.5250

y_4=0.8808

y_5=0.5097



- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, *e*, from the output layer backward to the input layer (Backward).
- First, we calculate the error gradient for neuron 5 in the output layer: $y_3=0.5250$, $y_4=0.8808$, $y_5=0.5097$
- $$\delta_5 = y_5 (1-y_5)\, e = 0.5097 \cdot (1-0.5097) \cdot (-0.5097) = -0.1274$$
- Then we determine the weight corrections assuming that the learning rate, $\alpha = 0.1$:



$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$
$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$
$$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

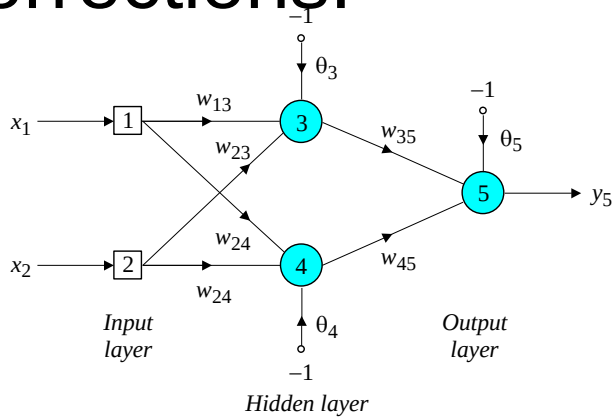
- Next we calculate the error gradients for neurons 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$
$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147$$

y_3=0.5250, y_4=0.8808, y_5=0.5097, x_1=1 and x_2=1

- We then determine the weight corrections:

$$\Delta w_{13} = \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta\theta_4 = \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015$$



Recall: w_{13} = 0.5, w_{14} = 0.9, w_{23} = 0.4, w_{24} = 1.0, w_{35} = –1.2, w_{45} = 1.1, θ_3 = 0.8, θ_4 = –0.1 and θ_5 = 0.3.

- At last, we update all weights and threshold:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$
$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$
$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$
$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$
$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$
$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$
$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$
$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$
$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

$$\Delta w_{13} = \alpha \cdot x_1 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha \cdot x_2 \cdot \delta_3 = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha \cdot (-1) \cdot \delta_3 = 0.1 \cdot (-1) \cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha \cdot x_1 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha \cdot x_2 \cdot \delta_4 = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$
$$\Delta\theta_4 = \alpha \cdot (-1) \cdot \delta_4 = 0.1 \cdot (-1) \cdot (-0.0147) = 0.0015$$

$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$

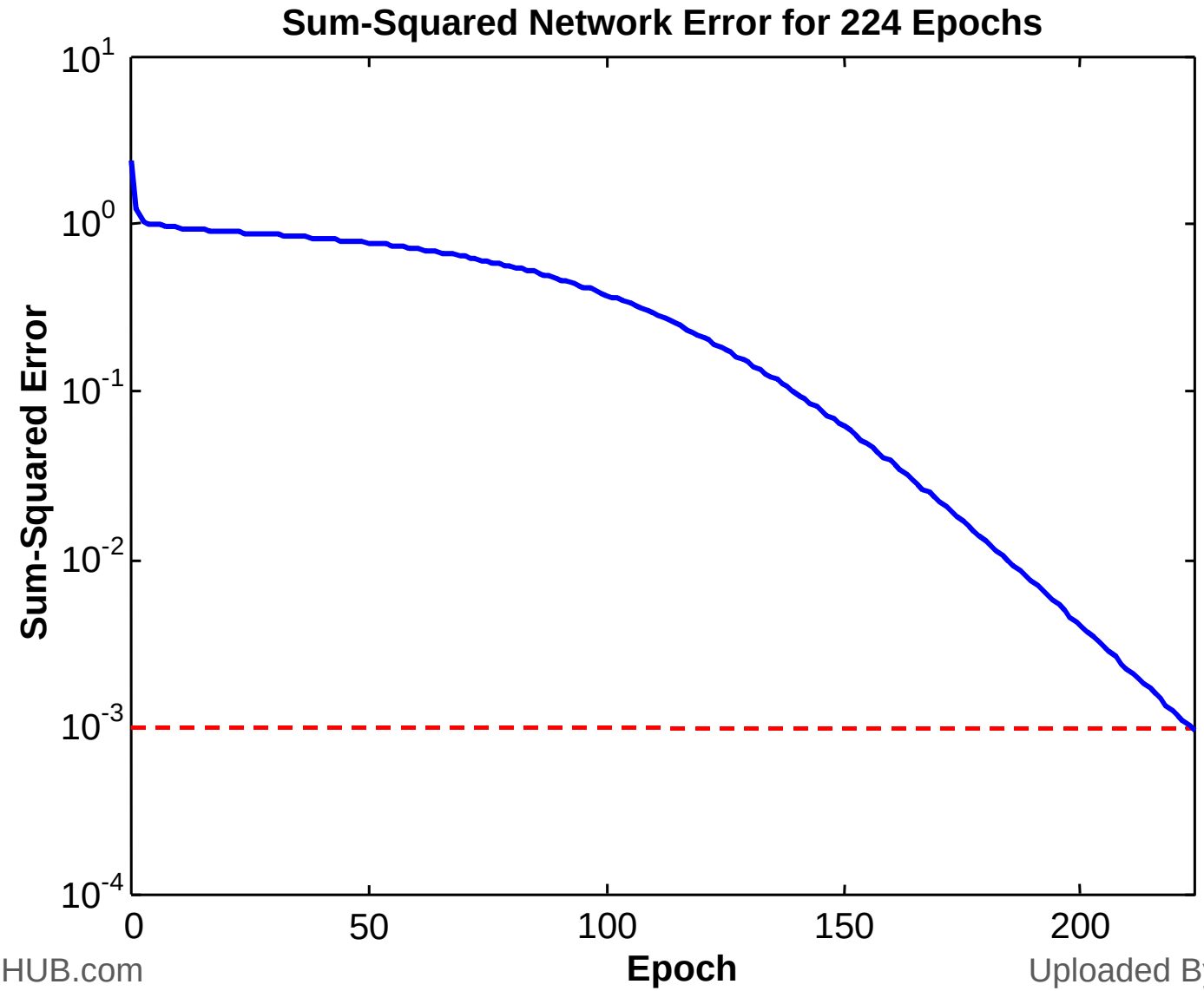
$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$

$$\Delta\theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

- The training process is repeated until the sum of squared errors is less than 0.001.

w_{13} = 0.5, w_{14} = 0.9, w_{23}= 0.4,w_{24} =1.0, w_{35}= –1.2, w_{45} = 1.1, θ_3 = 0.8, θ_4 = –0.1,θ_5 = 0.3

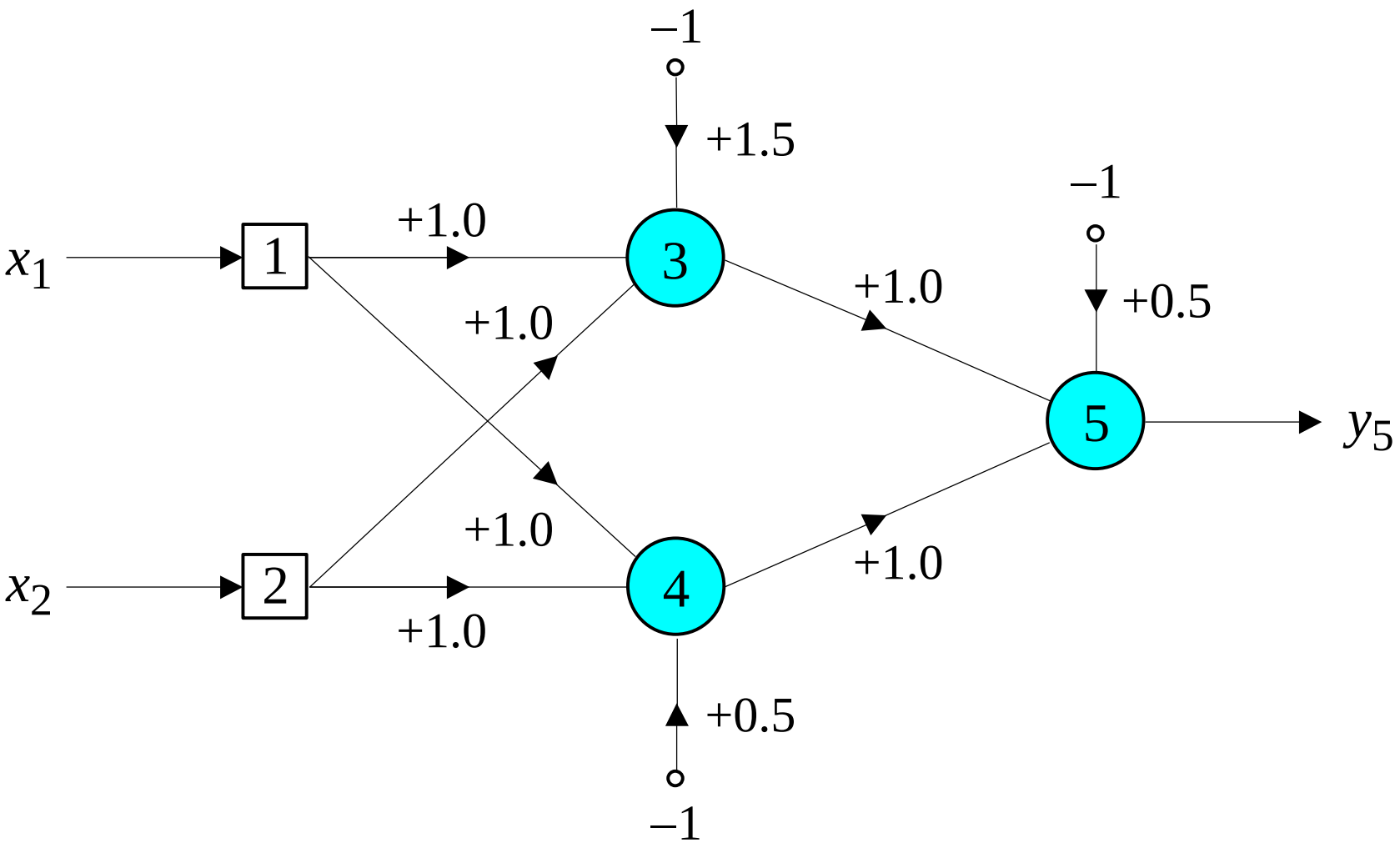
Learning curve for operation *Exclusive-OR*



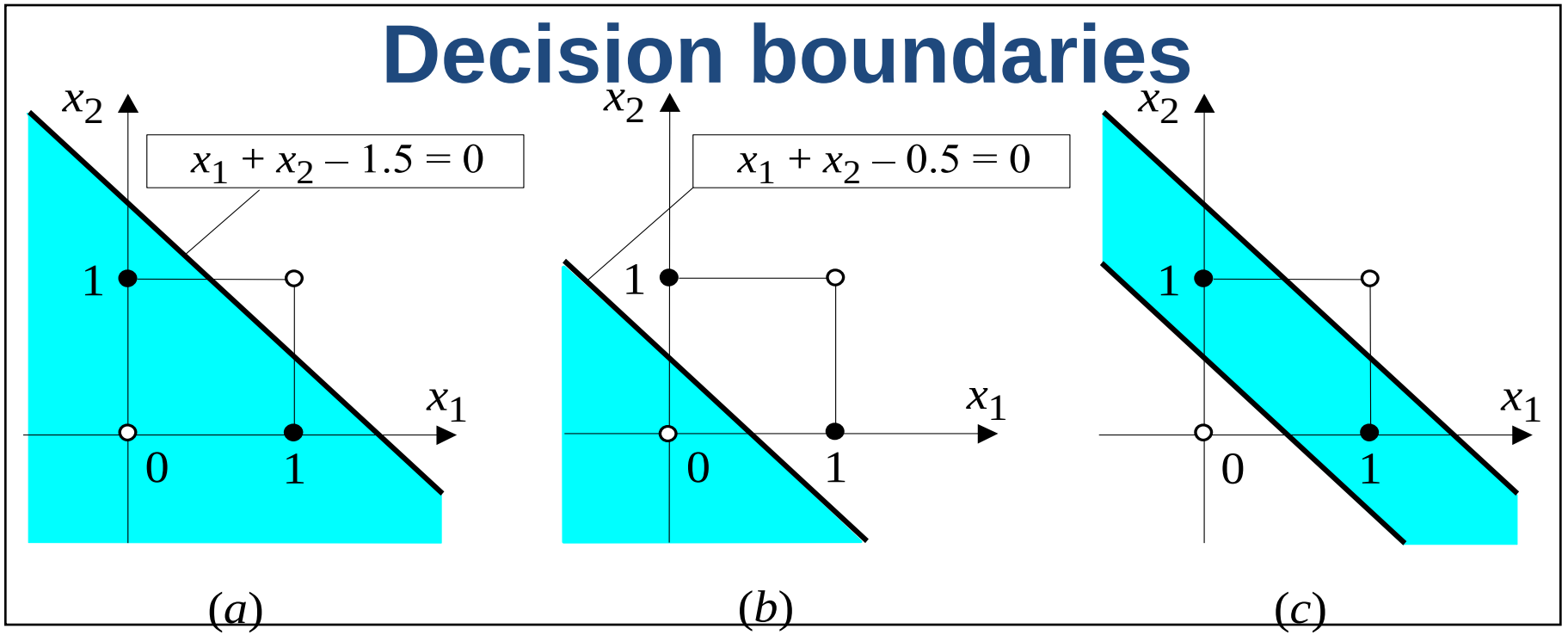
Final results of three-layer network learning

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

Network represented by McCulloch-Pitts model for solving the *XOR* operation



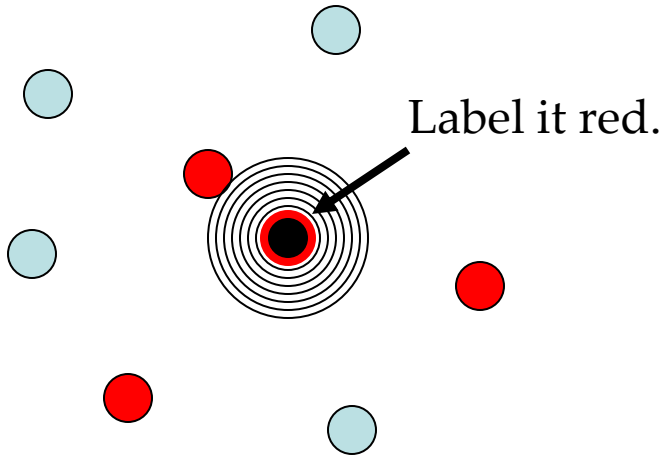
Decision boundaries



(*a*) Decision boundary constructed by hidden neuron 3;
(*b*) Decision boundary constructed by hidden neuron 4;
(*c*) Decision boundaries constructed by the complete three-layer network

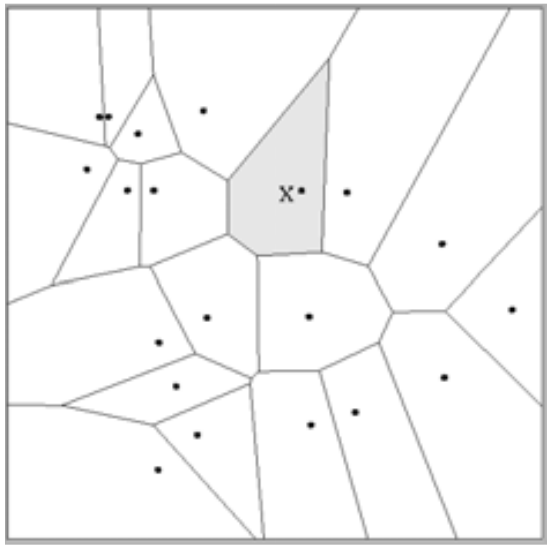
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point

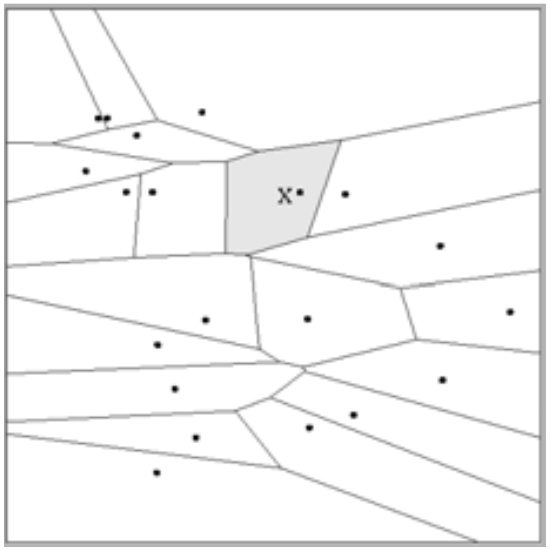


Distance Metrics

- Different metrics can change the decision surface: given points (examples) **a** and **b**



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (a_2 - b_2)^2$



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

- Standard Euclidean distance metric:
  - Two-dimensional: $\text{Dist}(a,b) = \text{sqrt}((a_1 - b_1)^2 + (a_2 - b_2)^2)$
  - Multivariate: $\text{Dist}(a,b) = \text{sqrt}(\sum (a_i - b_i)^2)$

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

1-NN's Aspects as an Instance-Based Learner:

A distance metric

- Euclidean (as usual)
  → D(x1,x2) =number of features on which x1 and x2 differ
- Others (e.g., normal, cosine)

How many nearby neighbors to look at?

- One: 1-NN,

How to fit with the local points?

- Just predict the same output as the nearest neighbor.

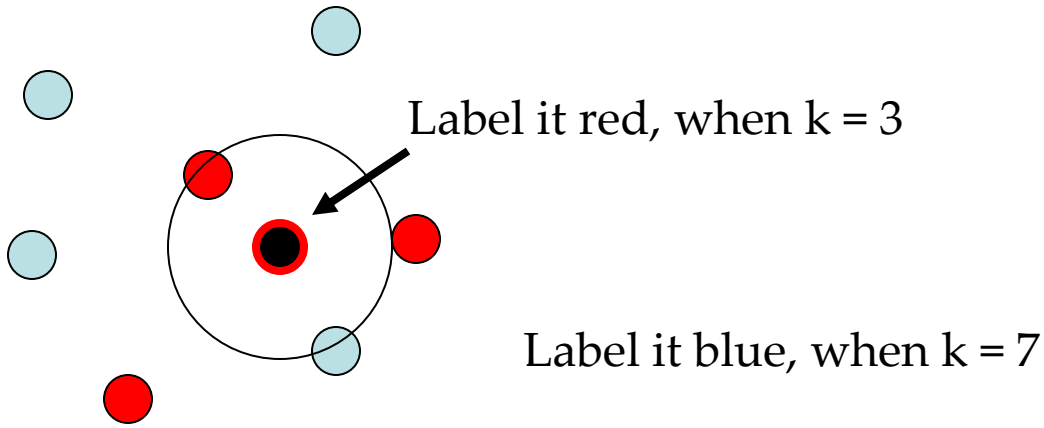
What if this only point is incorrect: Noise?

- Use more points (K), predict based on class of largest number of nearest neighbors.

Adapted from "Instance-Based Learning" lecture slides by Andrew Moore, CMU.

k – Nearest Neighbor

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its *k* nearest neighbors



KNN Example

	Food (3)	Chat (2)	Fast (2)	Price (3)	Bar (2)	BigTip
1	great	yes	yes	normal	no	yes
2	great	no	yes	normal	no	yes
3	mediocre	yes	no	high	no	no
4	great	yes	yes	normal	yes	yes

Similarity metric: Number of matching attributes (k=2)

- New examples:
  - Example 1 (great, no, no, normal, no) →? Yes/No

    → most similar: number 2 (1 mismatch, 4 match) → yes

    →Second most similar example: number 1 (2 mismatch, 3 match) → yes

  - Example 2 (mediocre, yes, no, normal, no) →? Yes/No

    → Most similar: number 3 (1 mismatch, 4 match) → no

    →Second most similar example: number 1 (2 mismatch, 3 match) → yes

Selecting the Number of Neighbors

- Increase k:
  - Makes KNN less sensitive to noise
- Decrease k:
  - Allows capturing finer structure of space, sensitive to noise.
- ➔Pick k not too large, but not too small (depends on data)

Curse-of-Dimensionality

- Prediction accuracy can quickly degrade when number of attributes grows.
  - **Irrelevant** attributes easily “swamp” information from **relevant** attributes
  - When many irrelevant attributes, similarity/distance measure becomes less reliable
- Remedy
  - Try to remove irrelevant attributes in pre-processing step
  - Weight attributes differently
  - Increase k (but not too much)

Advantages and Disadvantages of KNN

- Need distance/similarity measure and attributes that “match” target function.
- For large training sets,
- → Must make a pass through the entire dataset for each classification. This can be prohibitive for large data sets.
- Prediction accuracy can quickly degrade when number of attributes grows.

Unsupervised Learning: Clustering

Introduction

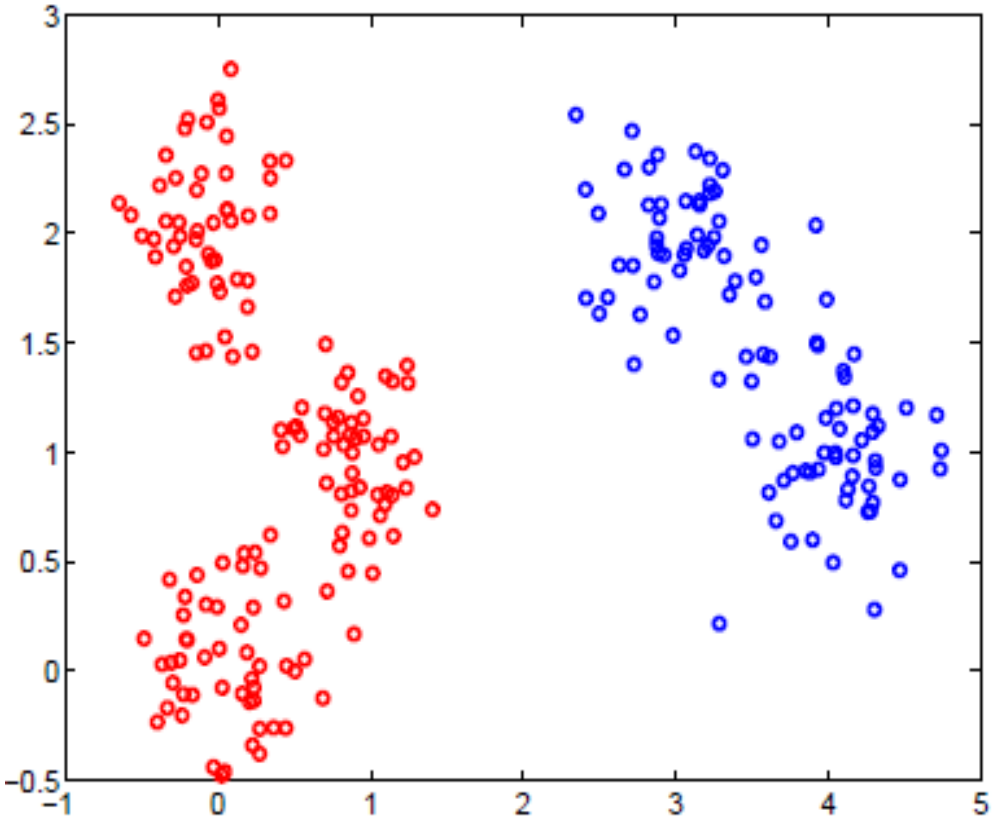
- Cluster: A collection/group of data objects/ points such that:
  - **similar** (related) to one another in same group
  - **dissimilar** (unrelated) to the objects in other groups
- Cluster Analysis
  - find ***similarities*** between data according to characteristics underlying the data and grouping similar data objects into clusters

Unsupervised Learning: Clustering

- Clustering Analysis: Unsupervised learning
  - No predefined classes for a training data set
  - Two general tasks:
    - identify the “natural” clusters **number** and
    - properly grouping objects into **“sensible”** clusters
- Typical applications
  - as a stand-alone tool to gain an insight into data distribution
  - as a preprocessing step of other algorithms in intelligent systems

Introduction

- Illustrative Example 1: how many clusters?



Introduction

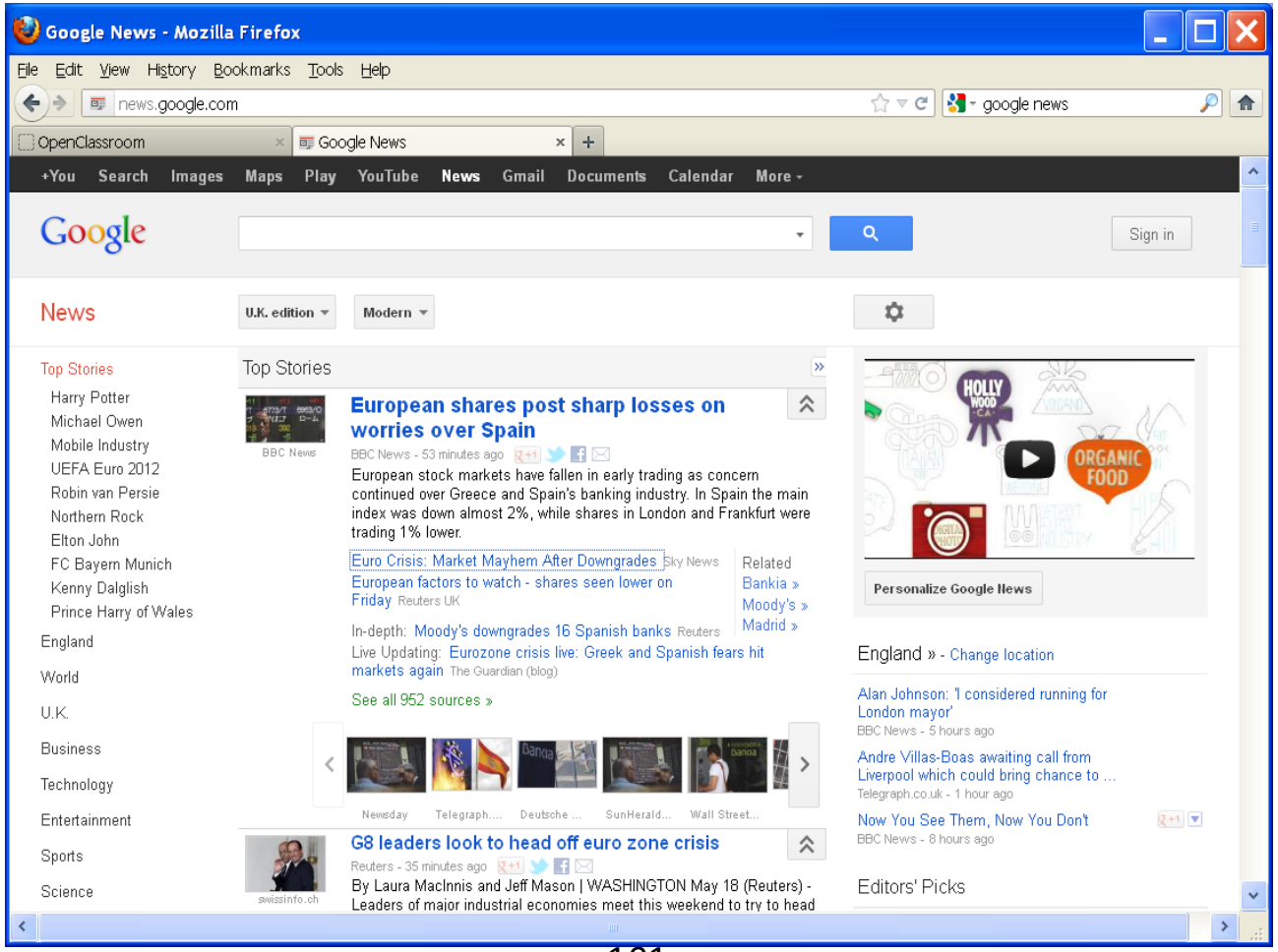
- Illustrative Example 2: are they in the same cluster? Which Features are important?

1. Two clusters
2. Clustering criterion:
   Article Language

1. Two clusters
2. Clustering criterion:
   Article Topic

Introduction

- Real Applications: Google News



Introduction

- Real world tasks:
  - Bank/Internet Security: fraud/spam pattern discovery
  - Biology: taxonomy of living things such as kingdom, phylum, class, order, family, genus and species
  - City-planning: Identifying groups of houses according to their house type, value, and geographical location
  - Climate change: understanding earth climate, find patterns of atmospheric and ocean
  - Finance: stock clustering analysis to uncover correlation underlying shares
  - Image Compression/segmentation: coherent pixels grouped
  - Information retrieval/organisation: Google search, topic-based news
  - Land use: Identification of areas of similar land use in an earth observation database
  - Marketing: Help marketers discover distinct groups in their customer bases, and then use this knowledge to develop targeted marketing programs
  - Social network mining: special interest group automatic discovery

Aspects of clustering

- A clustering algorithm
  - Partitional clustering
  - Hierarchical clustering
  - …
- A distance (similarity, or dissimilarity) function
- Clustering quality
  - Inter-clusters distance ⇒ maximized
  - Intra-clusters distance ⇒ minimized
- Clustering Quality depends on algorithm, distance function, and application.

Data Types and Representations

- **Discrete vs. Continuous**
  - Discrete Feature
    - Has only a finite set of value e.g., zip codes, rank, or the set of words in a collection of documents
    - Sometimes, represented as integer variable
  - Continuous Feature
    - Real numbers as feature values e.g. temperature, height, or weight, location: practically measured and represented using a finite number of digits
    - Typically represented as floating-point variables

Data Types and Representations

- Data representations
  - **Data** matrix (object-by-feature structure)

$$\begin{bmatrix} x_{11} & \cdots & x_{1f} & \cdots & x_{1p} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{i1} & \cdots & x_{if} & \cdots & x_{ip} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{n1} & \cdots & x_{nf} & \cdots & x_{np} \end{bmatrix}$$

- *n* data points (objects) with *p* dimensions (features)
- **Two modes**: row and column represent different entities
- E.g. Document/word matrix

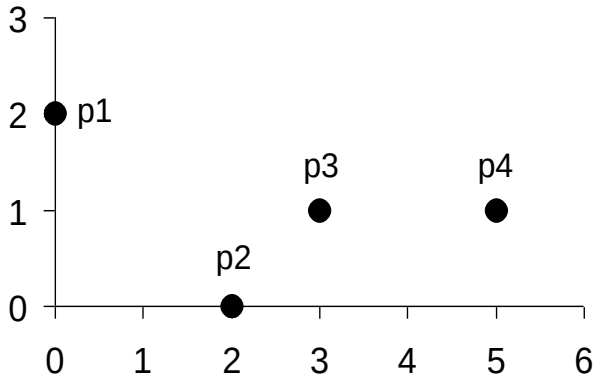
Distance/dissimilarity matrix: object-by-object structure

$$\begin{bmatrix} 0 & & & & \\ d(2,1) & 0 & & & \\ d(3,1) & d(3,2) & 0 & & \\ \vdots & \vdots & \vdots & & \\ d(n,1) & d(n,2) & \cdots & \cdots & 0 \end{bmatrix}$$

- *n* data points, but registers only the distance
- A symmetric/triangular matrix
- **Single mode**: row and column for the same entity (distance)

Data Types and Representations

- Examples



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Data Matrix

	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance Matrix (i.e., Dissimilarity Matrix) for Euclidean Distance

Distance Measures

- Minkowski Distance (http://en.wikipedia.org/wiki/Minkowski_distance)

  For $\mathbf{x} = (x_1\ x_2 \cdots x_n)$ and $\mathbf{y} = (y_1\ y_2 \cdots y_n)$

  $$d(\mathbf{x},\mathbf{y}) = \left(|\ x_1 - y_1\ |^p + |\ x_2 - y_2\ |^p \cdots + |\ x_n - y_n\ |^p\right)^{\frac{1}{p}}, \quad p > 0$$

  – $p = 1$: Manhattan (city block) distance

  $$d(\mathbf{x},\mathbf{y}) = |\ x_1 - y_1\ | + |\ x_2 - y_2\ | \cdots + |\ x_n - y_n\ |$$

  – $p = 2$: Euclidean distance

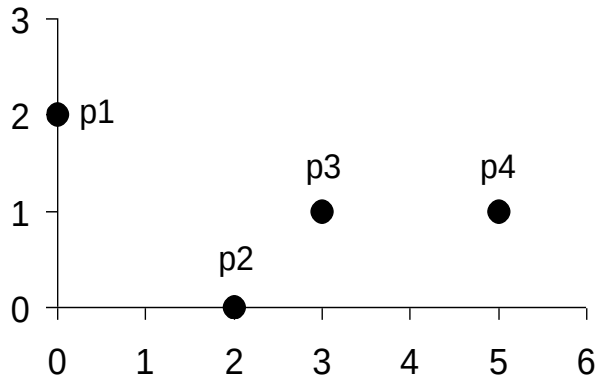
  $$d(\mathbf{x},\mathbf{y}) = \sqrt{|\ x_1 - y_1\ |^2 + |\ x_2 - y_2\ |^2 \cdots + |\ x_n - y_n\ |^2}$$

  – Do not confuse p with n, i.e., all these distances are defined based on all numbers of features (dimensions).

  – A generic measure: use appropriate p in different applications

Distance Measures

- Example: Manhatten and Euclidean



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

Data Matrix

L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

Distance Matrix for Manhattan Distance

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

Distance Matrix for Euclidean Distance

Distance Measures

- Cosine Measure (Similarity vs. Distance)

For $\mathbf{x} = (x_1\ x_2 \cdots x_n)$ and $\mathbf{y} = (y_1\ y_2 \cdots y_n)$

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}}$$

$$d(\mathbf{x}, \mathbf{y}) = 1 - \cos(\mathbf{x}, \mathbf{y})$$

$$0 \le d(\mathbf{x}, \mathbf{y}) \le 2$$

  - *x=(1,0,1), y=(0,1,1): cos(x,y)=1/2*
  - Property:
    - Nonmetric vector objects: keywords in documents, gene features in micro-arrays, …
    - Applications: information retrieval, biologic taxonomy, ...

Distance Measures

- Example: Cosine measure

$$\mathbf{x}_1 = (3, 2, 0, 5, 2, 0, 0), \mathbf{x}_2 = (1,0, 0, 0, 1, 0, 2)$$

$$3\times1+2\times0+0\times0+5\times0+2\times1+0\times0+0\times2=5$$

$$\sqrt{3^2+2^2+0^2+5^2+2^2+0^2+0^2} = \sqrt{42} \approx 6.48$$

$$\sqrt{1^2+0^2+0^2+0^2+1^2+0^2+2^2} = \sqrt{6} \approx 2.45$$

$$\cos(\mathbf{x}_1, \mathbf{x}_2) = \frac{5}{6.48\times2.45} \approx 0.32$$

$$d(\mathbf{x}_1, \mathbf{x}_2) = 1-\cos(\mathbf{x}_1, \mathbf{x}_2) = 1-0.32 = 0.68$$

K-means Clustering

Introduction

- Partitioning Clustering Approach
  - a typical clustering analysis approach via iteratively partitioning training data set to learn a partition of the given data space
  - learning a partition on a data set to produce several non-empty clusters (usually, the number of clusters **given** in advance)
  - in principle, optimal partition achieved via minimising the sum of squared distance to its "representative object", **or centroid**, in each cluster

$$E = \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} d^2(\mathbf{x}, \mathbf{m}_k)$$

e.g., Euclidean distance $d^2(\mathbf{x}, \mathbf{m}_k) = \sum_{n=1}^{N} (x_n - m_{kn})^2$

Introduction

- The *K-means* algorithm: a heuristic method
  - K-means algorithm (MacQueen'67): each cluster is represented by the center of the cluster and the algorithm converges to stable centroids of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis: widely used in data mining applications.

K-means Algorithm

- Given the cluster number *K*, the *K-means* algorithm works as follows:

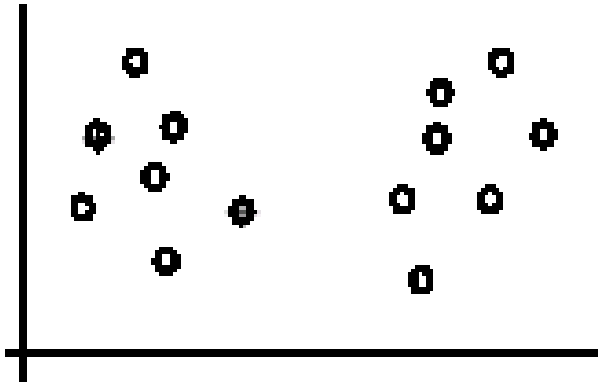
0)Initialisation: set initial **K** seed points –centroids- (randomly)

1)Assign each object to the cluster of the nearest **seed** point using the specific **distance metric**

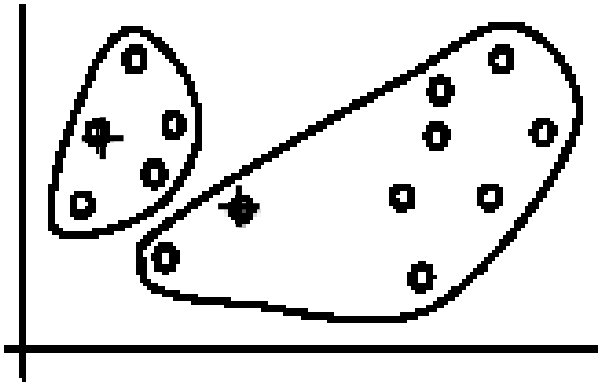
2)Compute the new seed points as the centroids of the clusters of the current partition (the centroid is the center, or *mean point*, of the cluster after additions)

3)Stop when no more new assignment (i.e., membership in each cluster no longer changes) else Go back to Step 1.

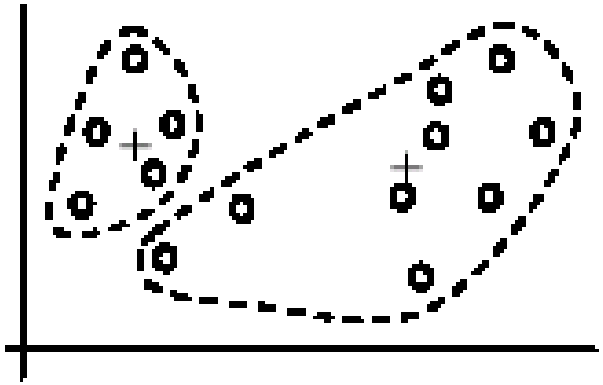
An example



(A). Random selection of k centers

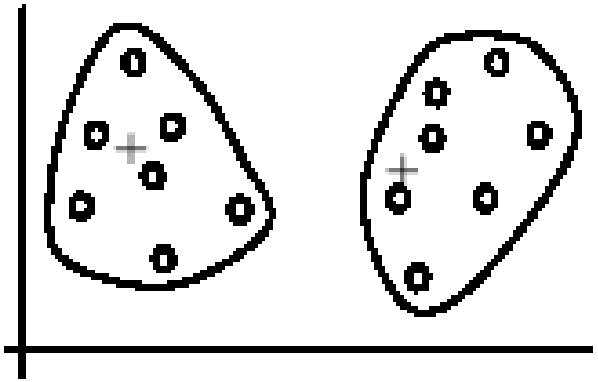


Iteration 1: (B). Cluster assignment

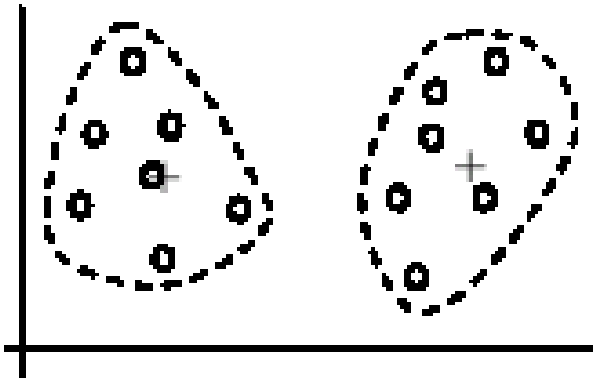


(C). Re-compute centroids

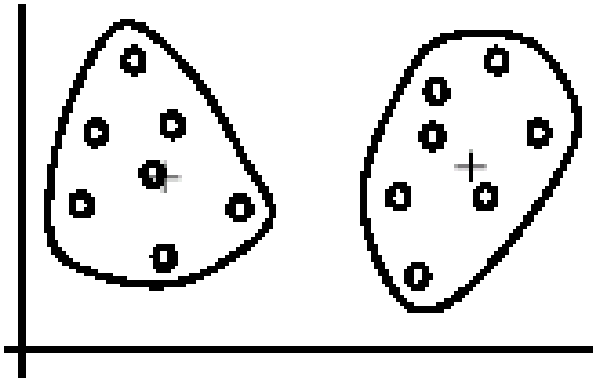
An example (cont …)



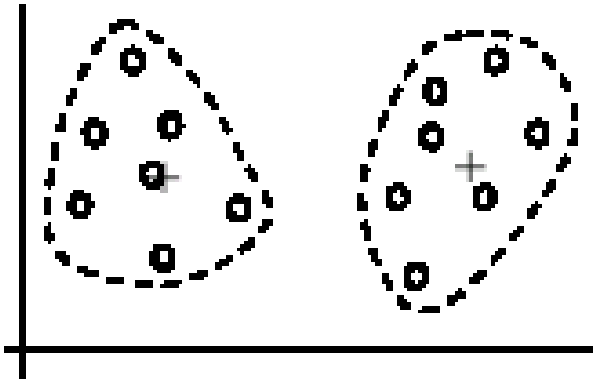
Iteration 2: (D). Cluster assignment



(E). Re-compute centroids



Iteration 3: (F). Cluster assignment



(G). Re-compute centroids

Stopping/convergence criterion

1. no (or minimum) re-assignments of data points to different clusters,
2. no (or minimum) change of centroids, or
3. minimum decrease in the **sum of squared error** (SSE) over all clusters,

$$SSE = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} dist(\mathbf{x}, \mathbf{m}_j)^2 \quad (1)$$

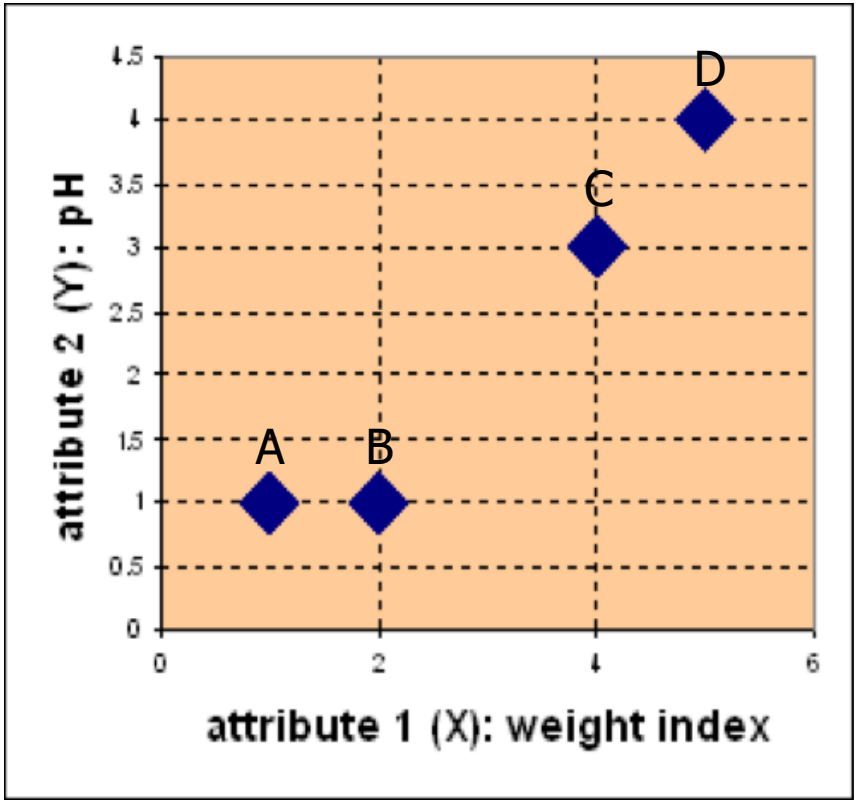
   - C_i is the *jth* cluster, $\mathbf{m}_j$ is the centroid of cluster C_j (the mean vector of all the data points in C_j), and $dist(\mathbf{x}, \mathbf{m}_j)$ is the distance between data point $\mathbf{x}$ and centroid $\mathbf{m}_j$.

Example

- Problem

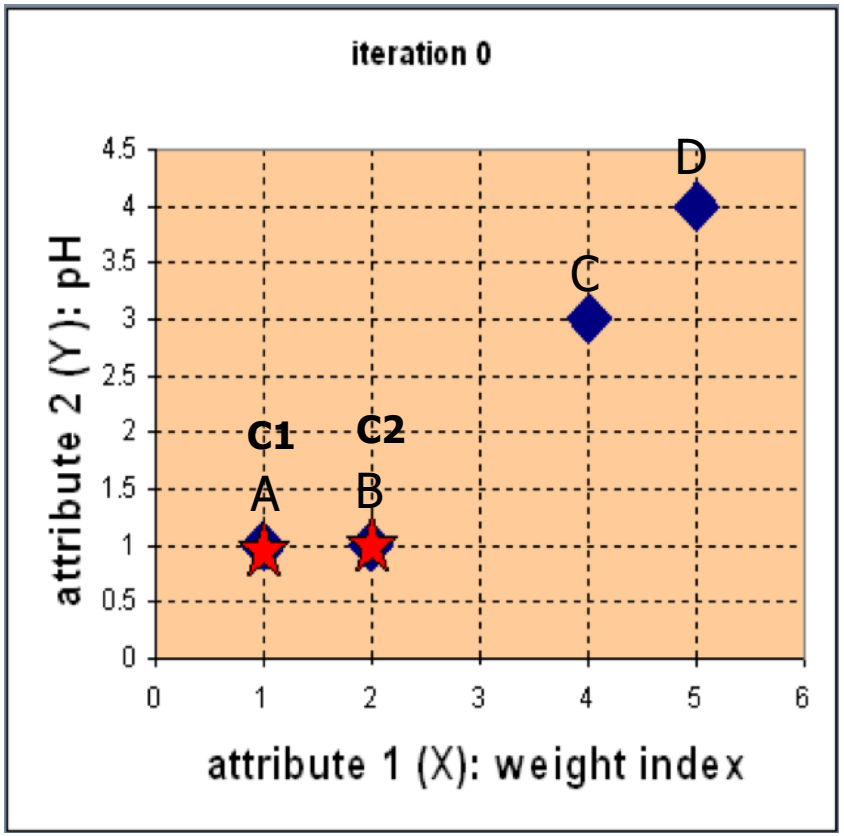
  Suppose we have 4 types of medicines and each has two attributes (pH and weight index). Our goal is to group these objects into *K=2* group of medicine.

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Example

- Step 1: Use initial seed points for partitioning



$$c_1 = A, c_2 = B$$

$$D^0 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{bmatrix} \quad \begin{matrix} c_1 = (1,1) & group-1 \\ c_2 = (2,1) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

Euclidean distance

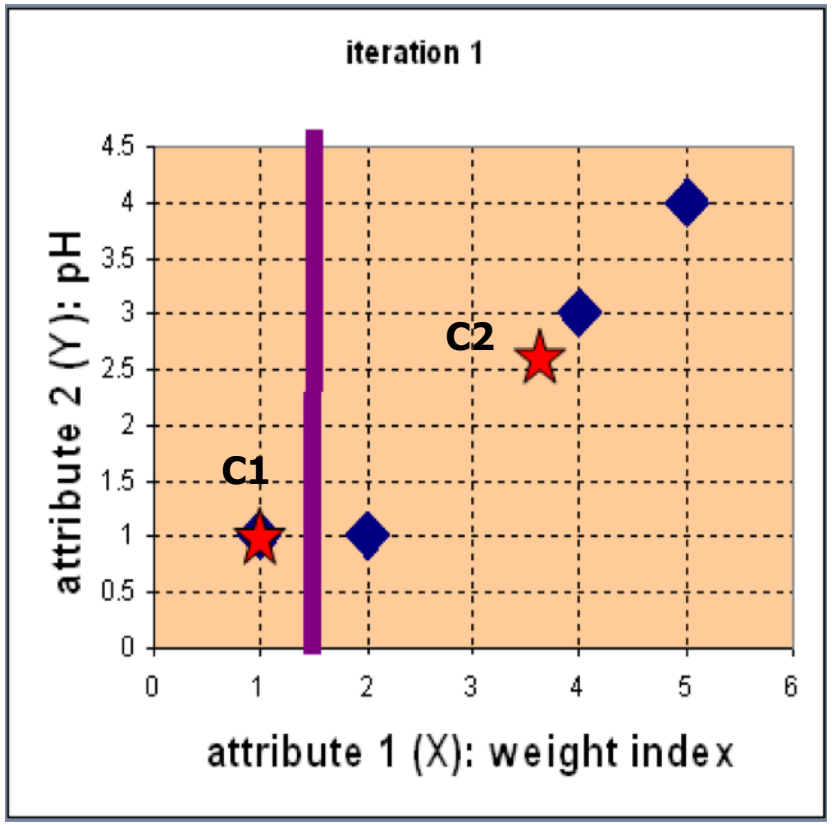
$$d(D, c_1) = \sqrt{(5-1)^2 + (4-1)^2} = 5$$

$$d(D, c_2) = \sqrt{(5-2)^2 + (4-1)^2} = 4.24$$

Assign each object to the cluster with the nearest seed point

Example

- Step 2: Compute new centroids of the current partition



Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

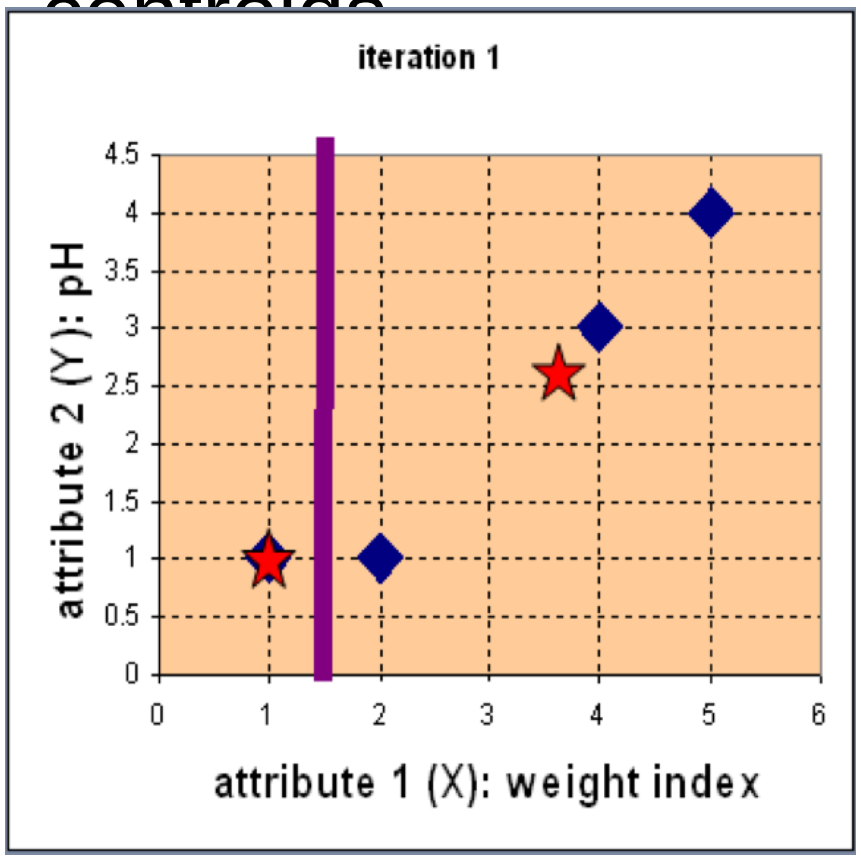
$$c_1 = (1,\ 1)$$

$$c_2 = \left(\frac{2+4+5}{3},\ \frac{1+3+4}{3}\right)$$

$$= \left(\frac{11}{3},\ \frac{8}{3}\right)$$

Example

- Step 2: Renew membership based on new centroids



Compute the distance of all objects to the new centroids

$$D^1 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{bmatrix} \quad \begin{matrix} c_1 = (1,1) & group-1 \\ c_2 = (\frac{11}{3}, \frac{8}{3}) & group-2 \end{matrix}$$

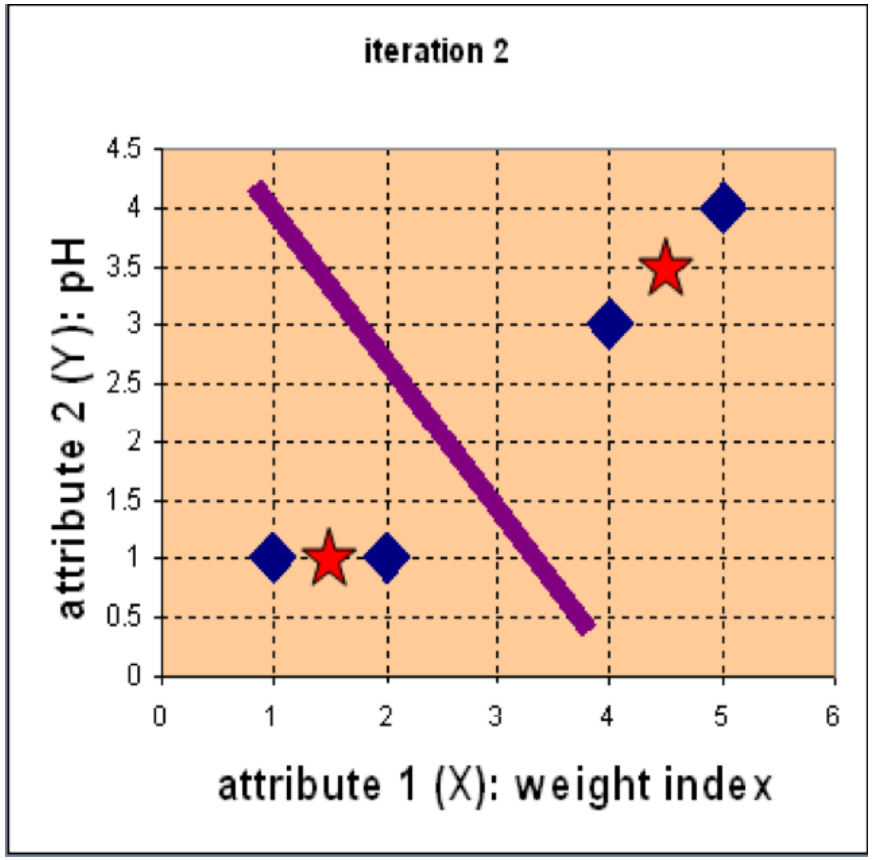
$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

Assign the membership to objects

Example

- Step 3: Repeat the first two steps until its



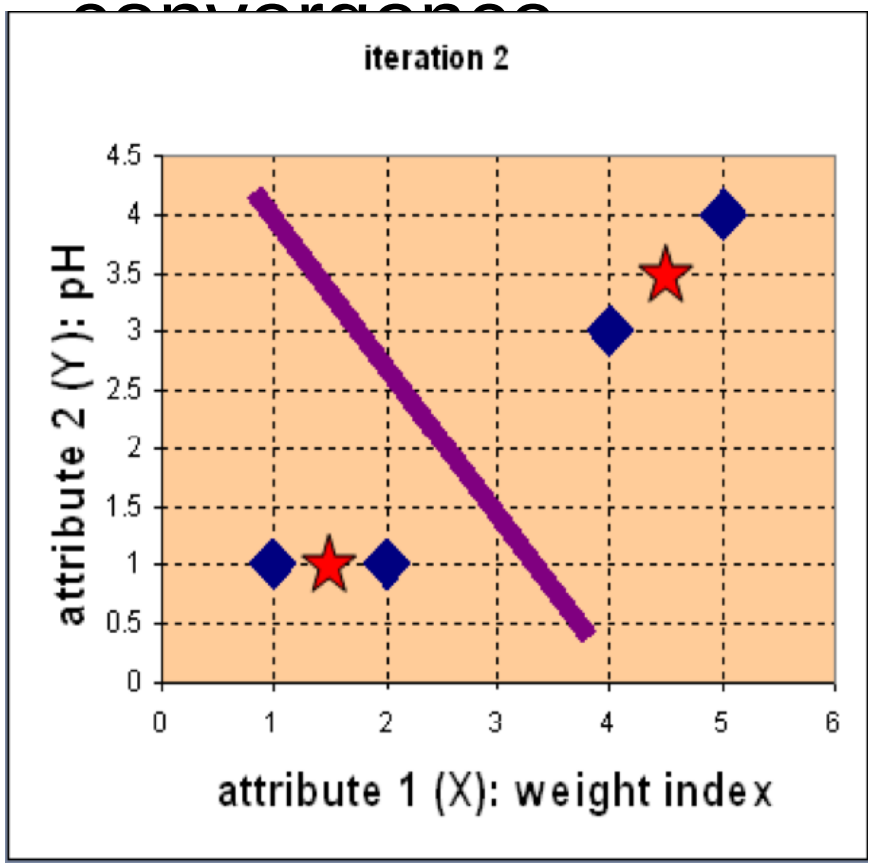
Knowing the members of each cluster, now we compute the new centroid of each group based on these new memberships.

$$c_1 = \left(\frac{1+2}{2}, \frac{1+1}{2}\right) = (1\frac{1}{2}, 1)$$

$$c_2 = \left(\frac{4+5}{2}, \frac{3+4}{2}\right) = (4\frac{1}{2}, 3\frac{1}{2})$$

Example

- Step 3: Repeat the first two steps until its convergence



Compute the distance of all objects to the new centroids

$$D^2 = \begin{bmatrix} 0.5 & 0.5 & 3.20 & 4.61 \\ 4.30 & 3.54 & 0.71 & 0.71 \end{bmatrix} \quad \begin{matrix} c_1 = (1\frac{1}{2}, 1) & group-1 \\ c_2 = (4\frac{1}{2}, 3\frac{1}{2}) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \begin{matrix} X \\ Y \end{matrix}$$

Stop due to **no new assignment**
Membership in each cluster no longer change

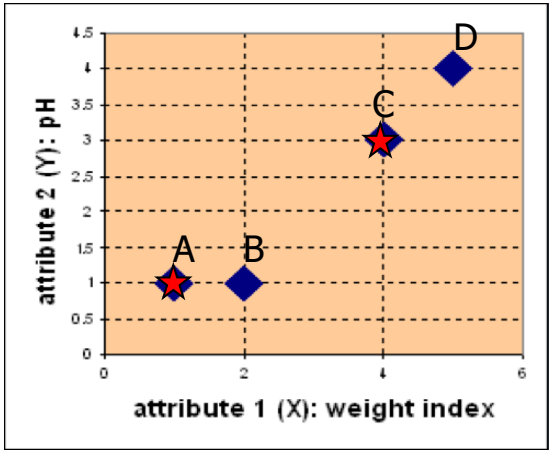
Exercise: Different Distance Metric

Use K-means with the Manhattan distance metric for clustering analysis by setting $K=2$ and initial seeds $C_1 = A$ and $C_2 = C$.

Answer three questions as follows:

1. How many steps are required for convergence?
2. What are memberships of two clusters after convergence?
3. What are centroids of two clusters after convergence?

Medicine	Weight	pH-Index
A	1	1
B	2	1
C	4	3
D	5	4



Another example: (using K=2)

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Step 1:

Initialization: Randomly we choose following two centroids m1[#1]=(1.0,1.0) and m2 [#4]=(5.0,7.0).

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

	Individual	Mean Vector
Group 1	1	(1.0, 1.0)
Group 2	4	(5.0, 7.0)

Step 2: From the distances:

- we obtain two clusters: {1,2,3} and {4,5,6,7}.
- Their new centroids are:

Individual	Centroid 1	Centroid 2
1	0	7.21
2 (1.5, 2.0)	1.12	6.10
3	3.61	3.61
4	7.21	0
5	4.72	2.5
6	5.31	2.06
7	4.30	2.92

$$m_1 = \left(\frac{1}{3}(1.0+1.5+3.0), \frac{1}{3}(1.0+2.0+4.0)\right) = (1.83, 2.33)$$

$$m_2 = \left(\frac{1}{4}(5.0+3.5+4.5+3.5), \frac{1}{4}(7.0+5.0+5.0+4.5)\right)$$

$$= (4.12, 5.38)$$

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

$$d(m_1, 2) = \sqrt{|1.0-1.5|^2 + |1.0-2.0|^2} = 1.12$$

$$d(m_2, 2) = \sqrt{|5.0-1.5|^2 + |7.0-2.0|^2} = 6.10$$

Step 3:

- Now using these **new** centroids we compute the Euclidean distance of each object, as in next table.
- m_1=(1.83,2.33), m_2=(4.12,5.33)
- The new clusters are:
  {1,2} and {**3**,4,5,6,7}
- Next centroids are: m1=(1.25,1.5) and m2 = (3.9,5.1) (**WHY?**)

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Individual	Centroid 1	Centroid 2
1	1.57	5.38
2	0.47	4.28
3	2.04	1.78
4	5.64	1.84
5	3.15	0.73
6	3.78	0.54
7	2.74	1.08

- Step 4 : Recall m1=(1.25,1.5) and m2 = (3.9,5.1)

  Now using the new centroids we compute the Euclidean distance of each object, as in next table.

The clusters obtained are:

{1,2} and {3,4,5,6,7}

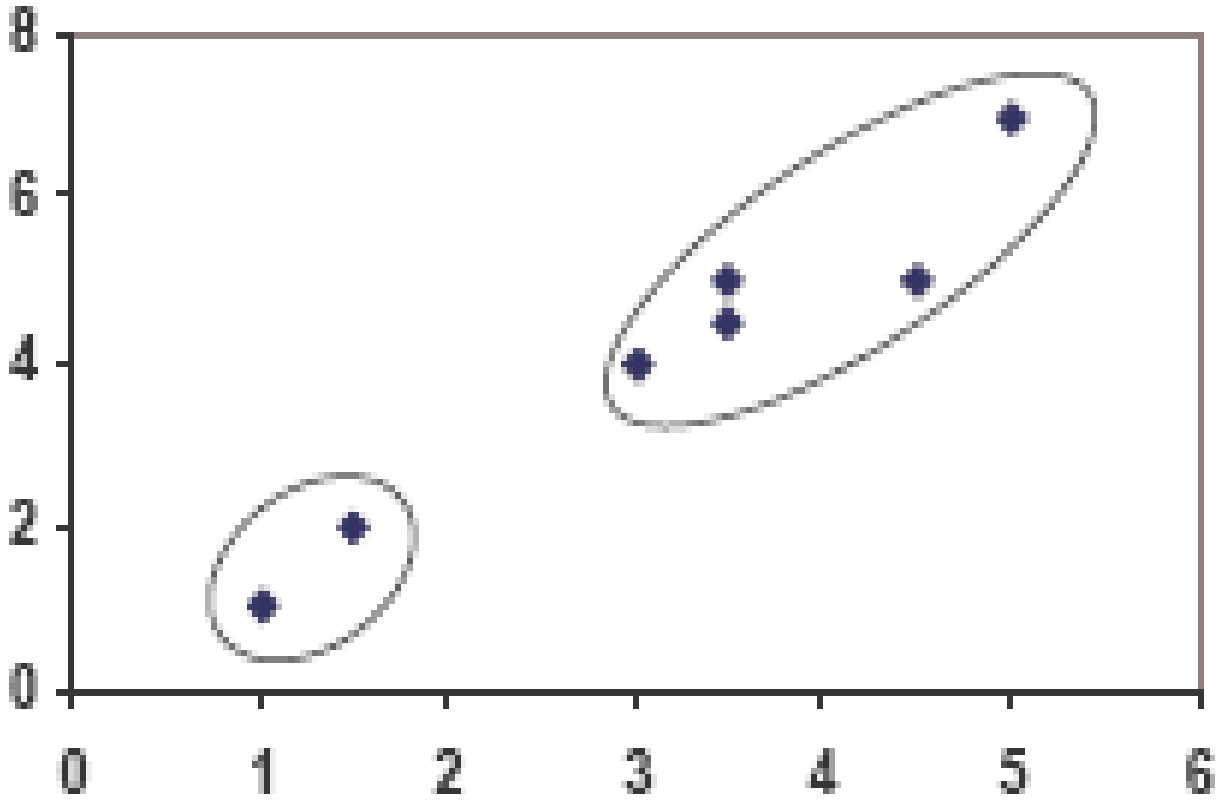
- There is no change in the cluster structure.
- Thus, the algorithm stops here and final result consist of 2 clusters

  {1,2} and {3,4,5,6,7}.

Individual	Variable 1	Variable 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Individual	Centroid 1	Centroid 2
1	0.56	5.02
2	0. 56	3.92
3	3.05	1.42
4	6.66	2.20
5	4.16	0.41
6	4.78	0.61
7	3.75	0.72

PLOT



(with K=3)

Individual	$m_1 = 1$	$m_2 = 2$	$m_3 = 3$	cluster
1	0	1.11	3.61	1
2	1.12	0	2.5	2
3	3.61	2.5	0	3
4	7.21	6.10	3.61	3
5	4.72	3.61	1.12	3
6	5.31	4.24	1.80	3
7	4.30	3.20	0.71	3

C_3

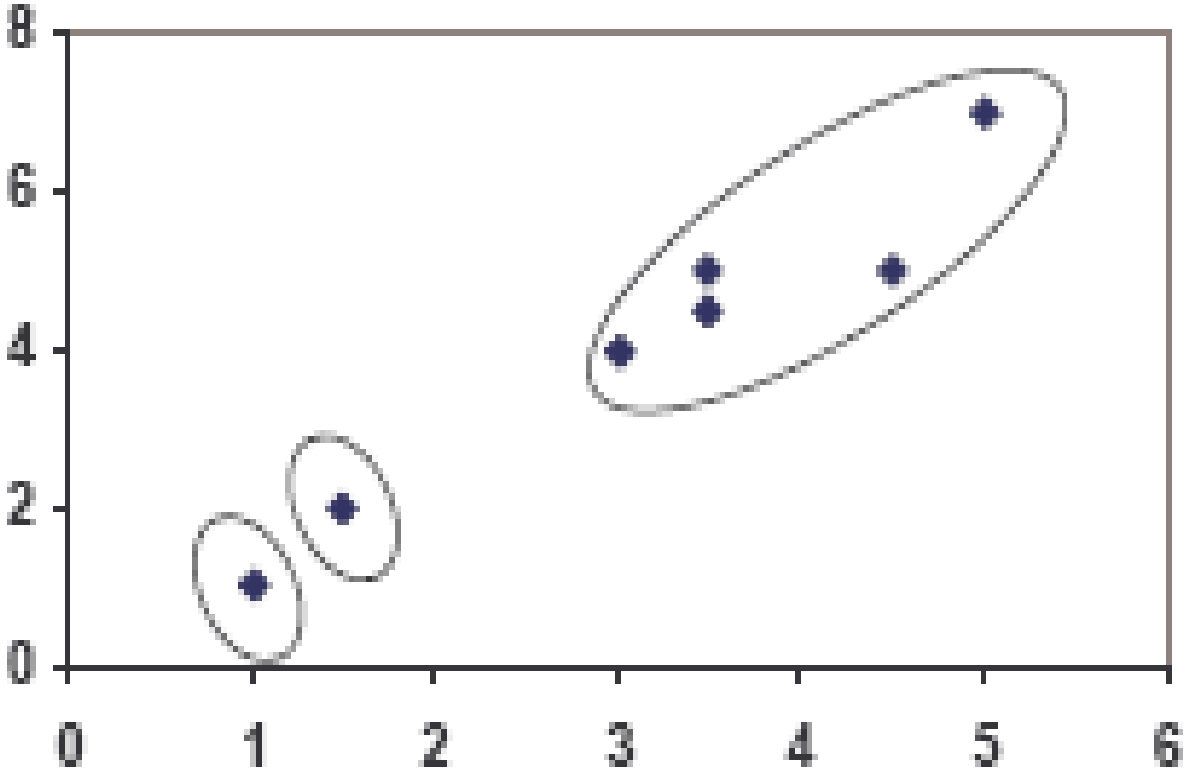
clustering with initial centroids (1, 2, 3)

Step 1

Individual	m_1 (1.0, 1.0)	m_2 (1.5, 2.0)	m_3 (3.9,5.1)	cluster
1	0	1.11	5.02	1
2	1.12	0	3.92	2
3	3.61	2.5	1.42	3
4	7.21	6.10	2.20	3
5	4.72	3.61	0.41	3
6	5.31	4.24	0.61	3
7	4.30	3.20	0.72	3

Step 2

PLOT



Strengths of k-means

- Strengths:
  - Simple: easy to understand and to implement
  - Efficient: Time complexity: $O(tkn)$,

    where n is the number of data points,

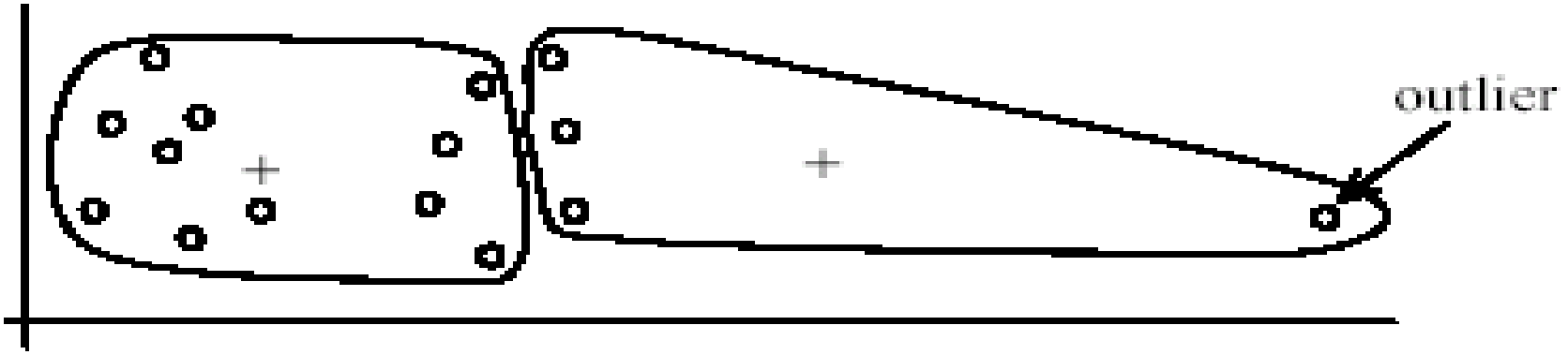
    k is the number of clusters, and

    t is the number of iterations (conversion can be slow!).
  - Since both k and t are usually small. k-means is considered a linear algorithm.
- K-means: most popular clustering algorithm.
- Note that: it terminates at a local optimum if SSE (Sum of Squared Errors) is used. The global optimum is hard to find due to complexity.

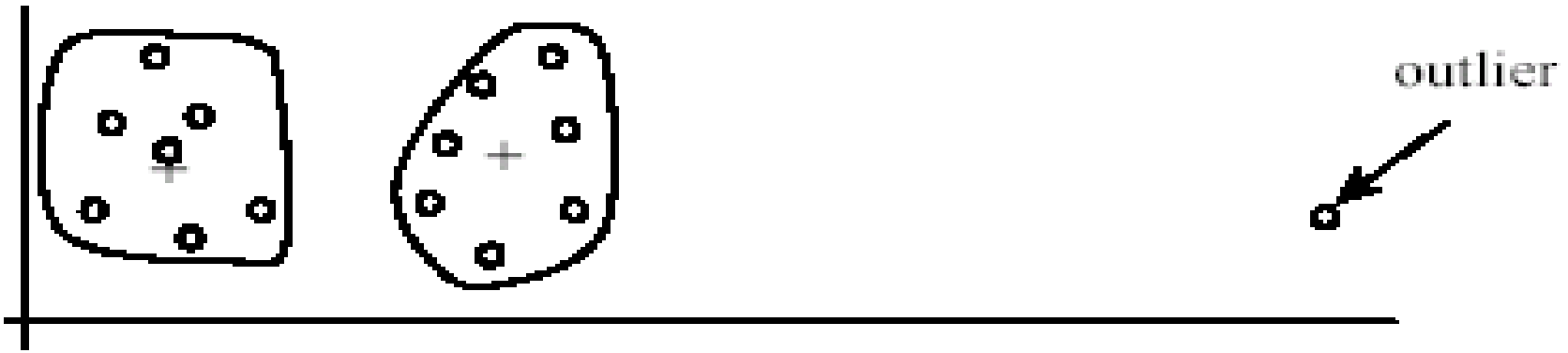
Weaknesses of k-means

- The algorithm is only applicable if the mean is defined.
  - For categorical data, *k*-mode - the centroid is represented by most frequent values.
- The user needs to specify *k*.
- The algorithm is sensitive to **outliers**
  - Outliers: data points very far away from other data points.
  - Outliers could be errors in data recording or special data points with very different values.

Weaknesses of k-means: Problems with outliers



(A): Undesirable clusters



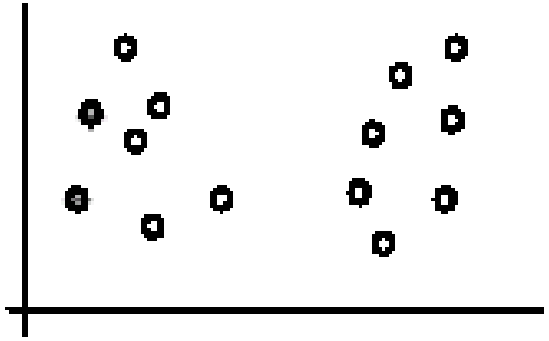
(B): Ideal clusters

Weaknesses of k-means: **Outliers**

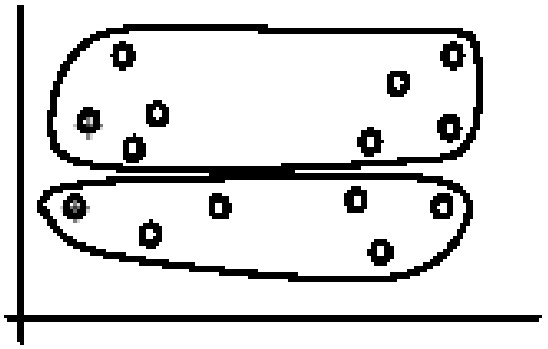
- Remove some data points in the clustering process that are much further away from the centroids than other data points.
  - To be safe, we may want to monitor these possible outliers over a few iterations and then decide to remove them.
- Or perform random sampling. Since in sampling we only choose a small subset of the data points, the chance of selecting an outlier is very small (*larger data sets*).
  - Assign the rest of the data points to the clusters by distance or similarity comparison, or classification

Weaknesses of k-means (cont.)

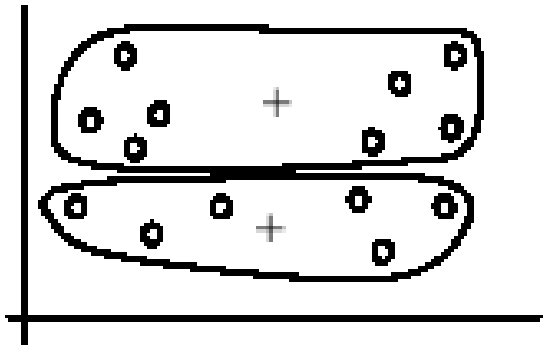
- The algorithm is sensitive to initial seeds.



(A). Random selection of seeds (centroids)



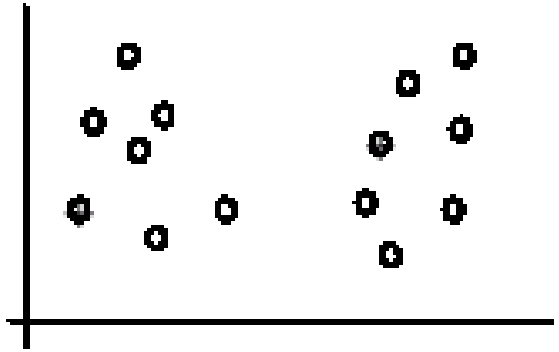
(B). Iteration 1



(C). Iteration 2

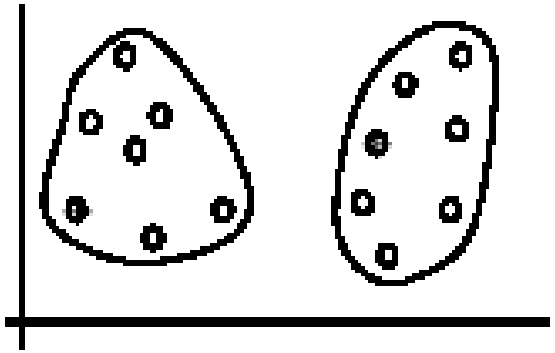
Weaknesses of k-means (cont.)

- If we use different seeds: good

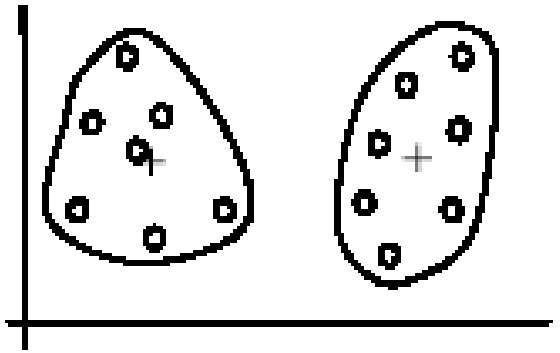


There are some methods to help choose good seeds

(A). Random selection of k seeds (centroids)



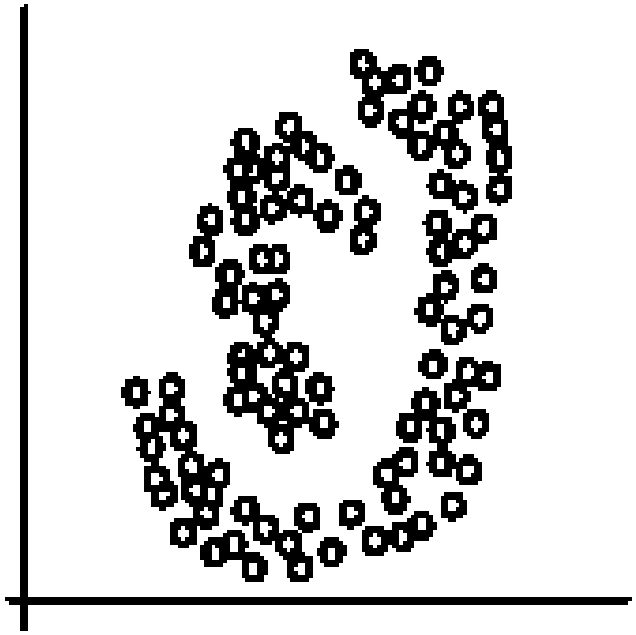
(B). Iteration 1



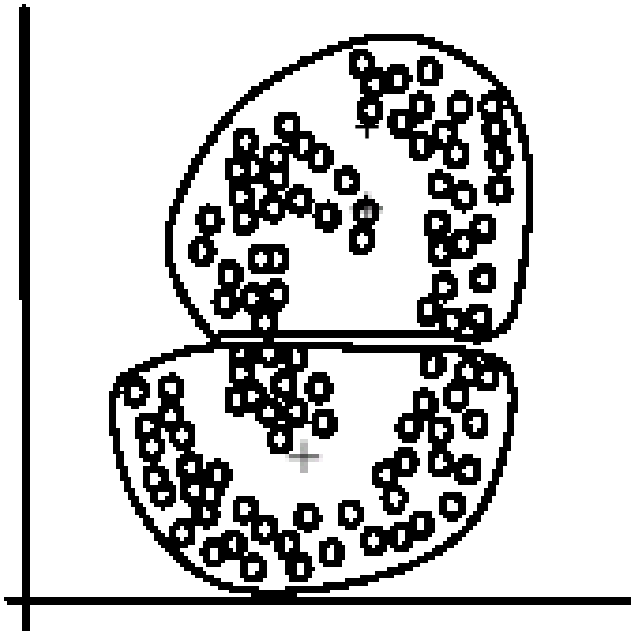
(C). Iteration 2

Weaknesses of k-means (cont.)

- The *k*-means algorithm is not suitable for discovering clusters that are not hyper-ellipsoids (or hyper-spheres).



(A): Two natural clusters



(B): *k*-means clusters

Cluster Evaluation: hard problem

- The quality of clustering is very hard to evaluate because
  - We do not know the correct clusters
- Some methods, however, are used:
  - User inspection
    - Study centroids, and spreads
    - Rules from a decision tree.
    - For text documents, one can read some documents in clusters.

Cluster evaluation: ground truth

- We use some labeled data (for classification)
- Assumption: Each class is a cluster.
- After clustering, a confusion matrix is constructed. From the matrix, we compute various measurements, entropy, purity, precision, recall and F-score.
  - Let the classes in the data D be $C = (c_1, c_2, \ldots, c_k)$. The clustering method produces k clusters, which divides D into k disjoint subsets, $D_1, D_2, \ldots, D_k$.

What Is A Good Clustering?

Internal criterion: A good clustering will produce high quality clusters in which:

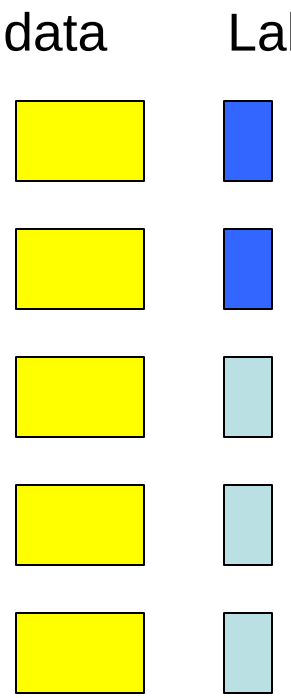
- the intra-class (intra-cluster) similarity is high
- the inter-class similarity is low

How would you evaluate clustering?

Common approach: use labeled data

Use data with known classes

- For example, document classification data



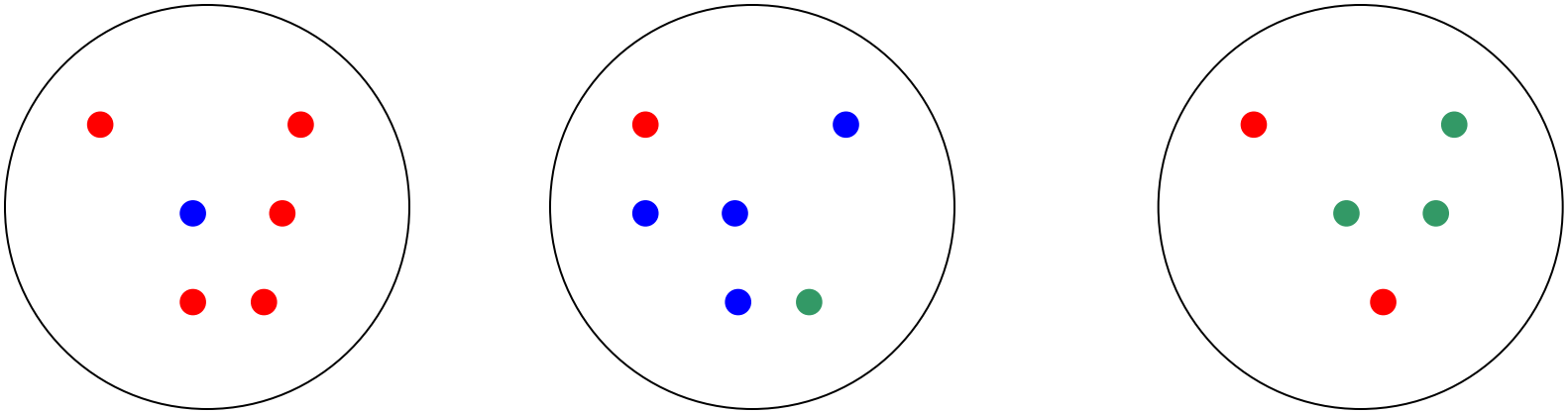
If we clustered this data (ignoring labels) what would we like to see?

Reproduces class partitions

How can we quantify this?

Common approach: use labeled data

Purity, the proportion of the dominant class in the cluster



Cluster I Cluster II Cluster III

Cluster I: Purity = 1/6 (max(5, 1, 0)) = 5/6=.83

Cluster II: Purity = 1/6 (max(1, 4, 1)) = 4/6=.67

Cluster III: Purity = 1/5 (max(2, 0, 3)) = 3/5=.6

Average Purity=1/3 (.83+.67+.6)=.70
=1/total_Points(sum of Majority points)
=1/17(5+4+3)=12/17=.70

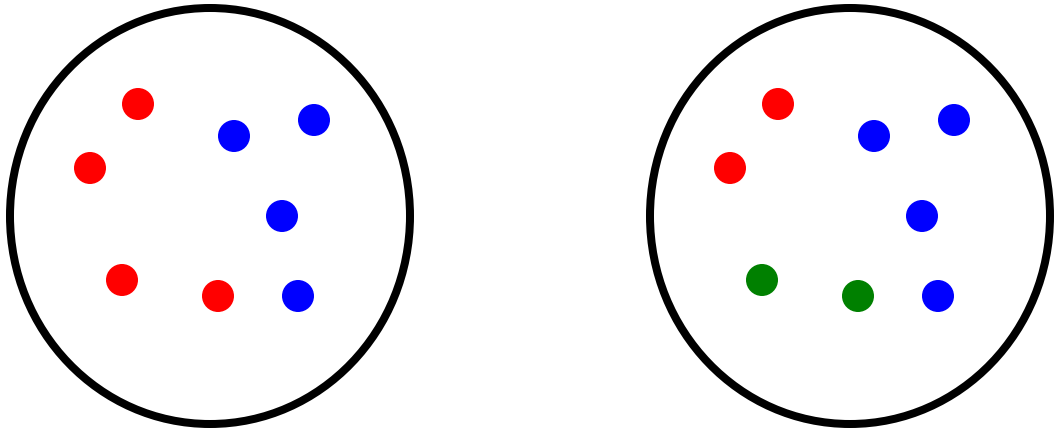
Other purity issues…

Purity, the proportion of the dominant class in the cluster

Good for comparing two algorithms, but not understanding how well a single algorithm is doing, why?

- Increasing the number of clusters increases purity

Purity isn’t perfect



Which is better based on purity?

Which do you think is better?

Ideas?

Common approach: use labeled data

Average entropy of classes in clusters

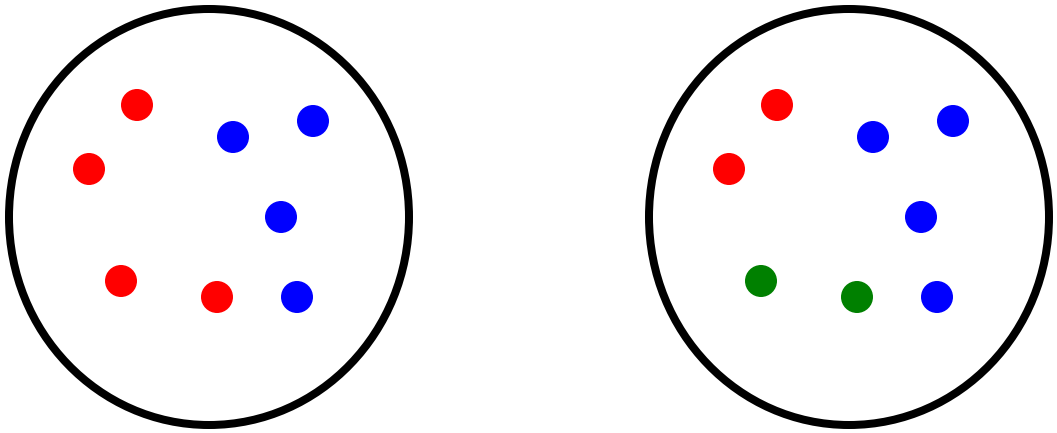
$$entropy(cluster) = -\frac{1}{num\_classes}\sum_i p(class_i)\log p(class_i)$$

where $p(class_i)$ is proportion of class *i* in cluster

Common approach: use labeled data

Average entropy of classes in clusters

$$entropy(cluster) = -\sum_i p(class_i)\log p(class_i)$$

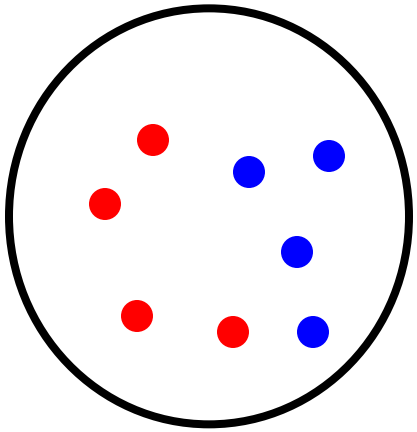


entropy?

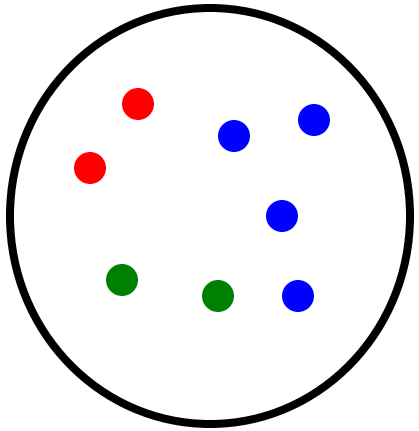
Common approach: use labeled data

Average entropy of classes in clusters

$$entropy(cluster) = -\sum_i p(class_i)\log p(class_i)$$



$-0.5\log 0.5 - 0.5\log 0.5 = 1$



$-0.5\log 0.5 - 0.25\log 0.25 - 0.25\log 0.25 = 1.5$

Entropy Computation: https://stackoverflow.com/questions/35709562/how-to-calculate-clustering-entropy-a-working-example-or-software-code

K-means summary

- Despite weaknesses, *k*-means is still the most popular algorithm due to its simplicity, efficiency and
  - other clustering algorithms have their own lists of weaknesses.
- No clear evidence that any other clustering algorithm performs better in general
  - although they may be more suitable for some specific types of data or applications.
- Comparing different clustering algorithms is a difficult task. No one knows the correct clusters!