Lecture 9: Supervised learning

Artificial Intelligence
CS-UY-4613-A / CS-GY-6613-I

Julian Togelius
julian.togelius@nyu.edu

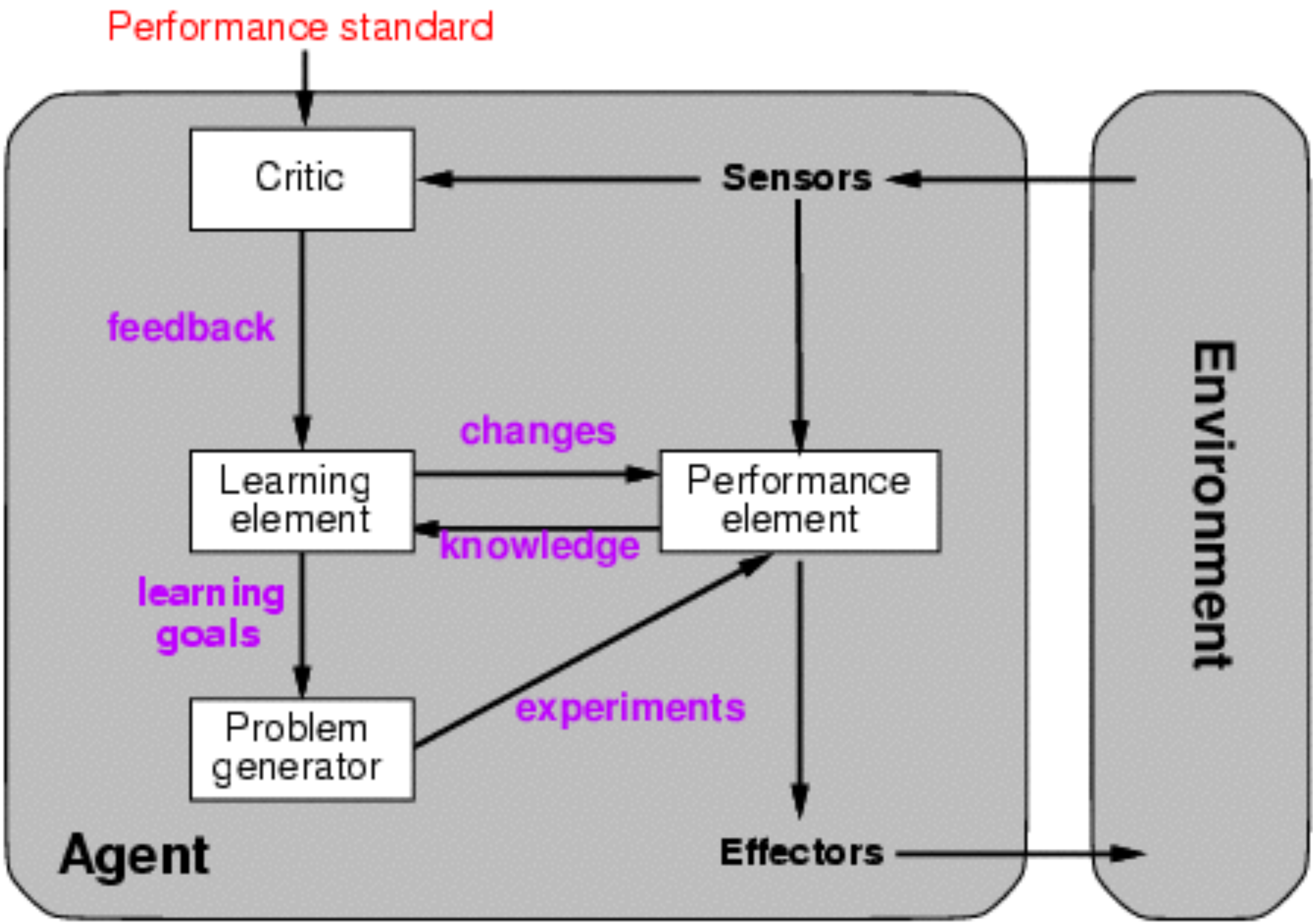
Contents

- Learning in intelligent agents
- Supervised learning and other forms of learning
- Eager and lazy learning
- k-Nearest Neighbor
- Perceptrons

Why learning?

- So far in the course: we have specified the mechanism by which the agent should decide how to act
- So far in the course: the world is completely known
- The agent might not know what the world is like, or what policies work well in the world
- The world may change
- You don't want to do all the programming

A (complex) learning agent

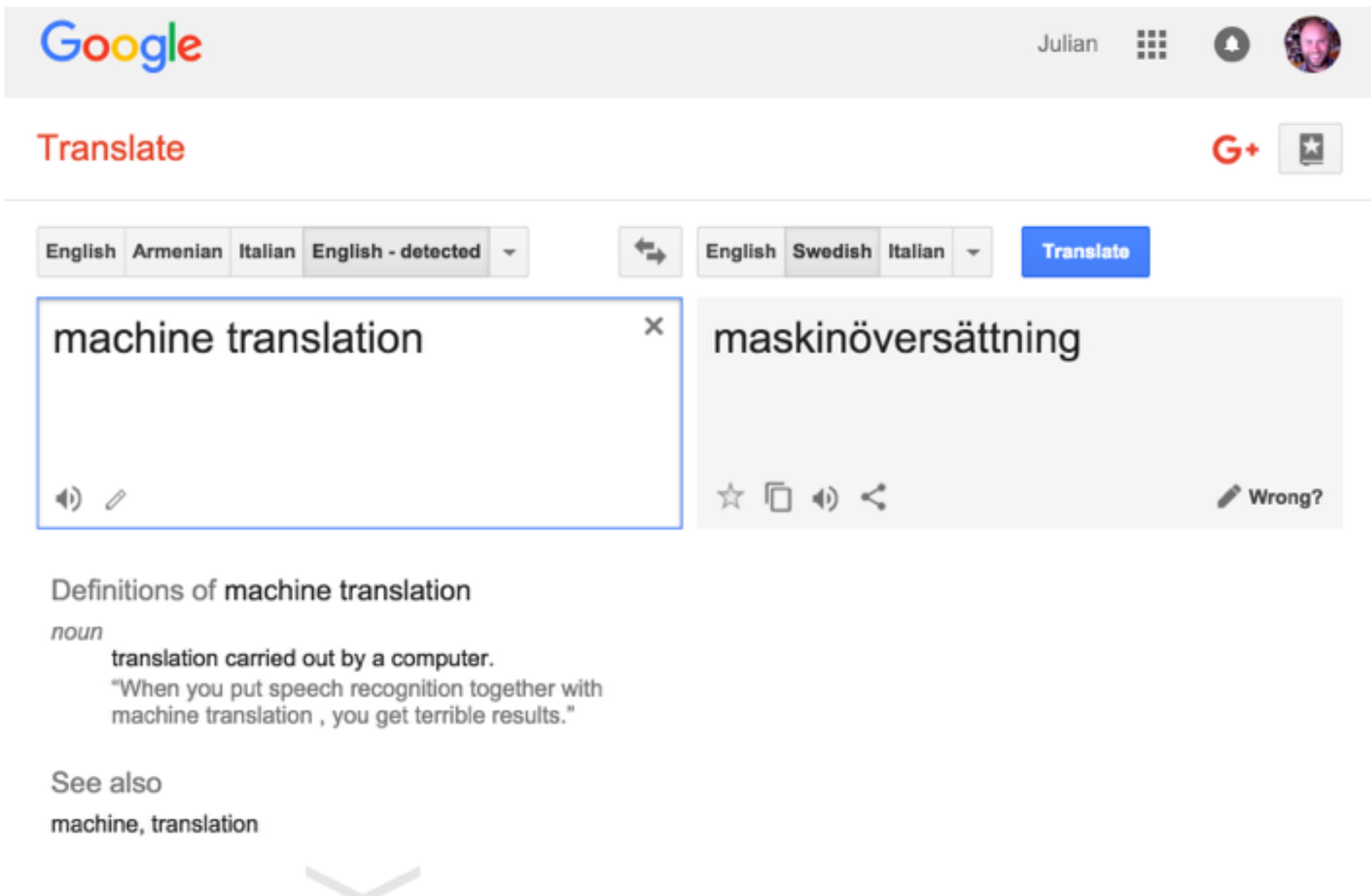


What could be learned?

What could be learned?

- How to drive from point A to point B, without hitting pedestrians
- What a human (or a cat, or bush) looks like
- Hand gestures
- How to drive in the style of a particular human (or according to that human's preferences)
- Which routes from A to B are actually fastest
- How far back you want your seat, temperature for the AC, favorite radio channel…
- Estimating distances

What could be learned?

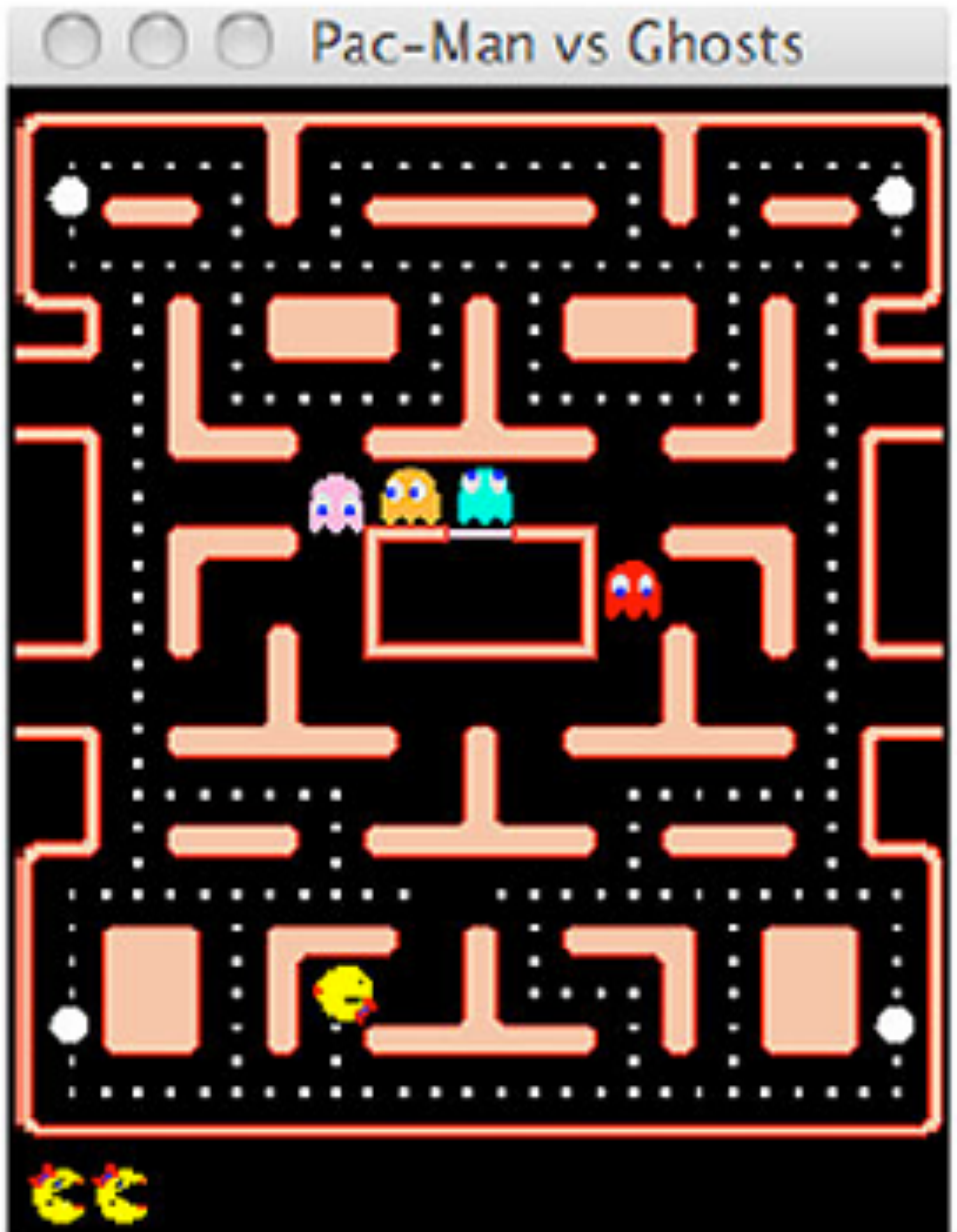


What could be learned?

What could be learned?

- How much a board position is worth
- What action to take in a specific situation
- What action a particular person would take in a specific situation
- Who is likely to win a game between two people, and how long it takes

What could be learned?



What could be learned?

- What actions a ghost would take
- The value of a state
- Dangerous positions
- What action to take in a specific situation

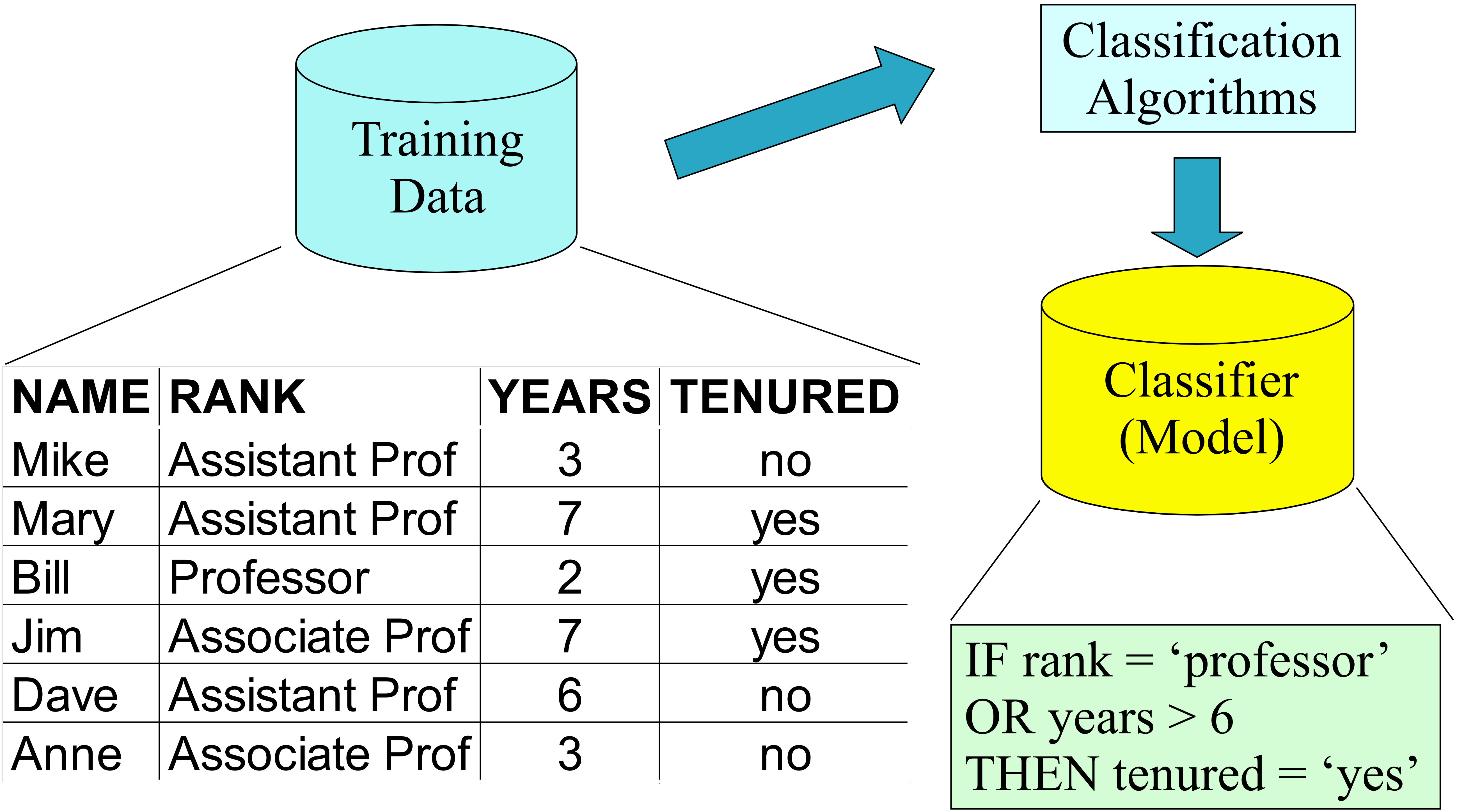
Types of learning

- **Supervised learning**
  Learning to predict or classify labels based on labeled input data

- **Unsupervised learning**
  Finding patterns in unlabeled data

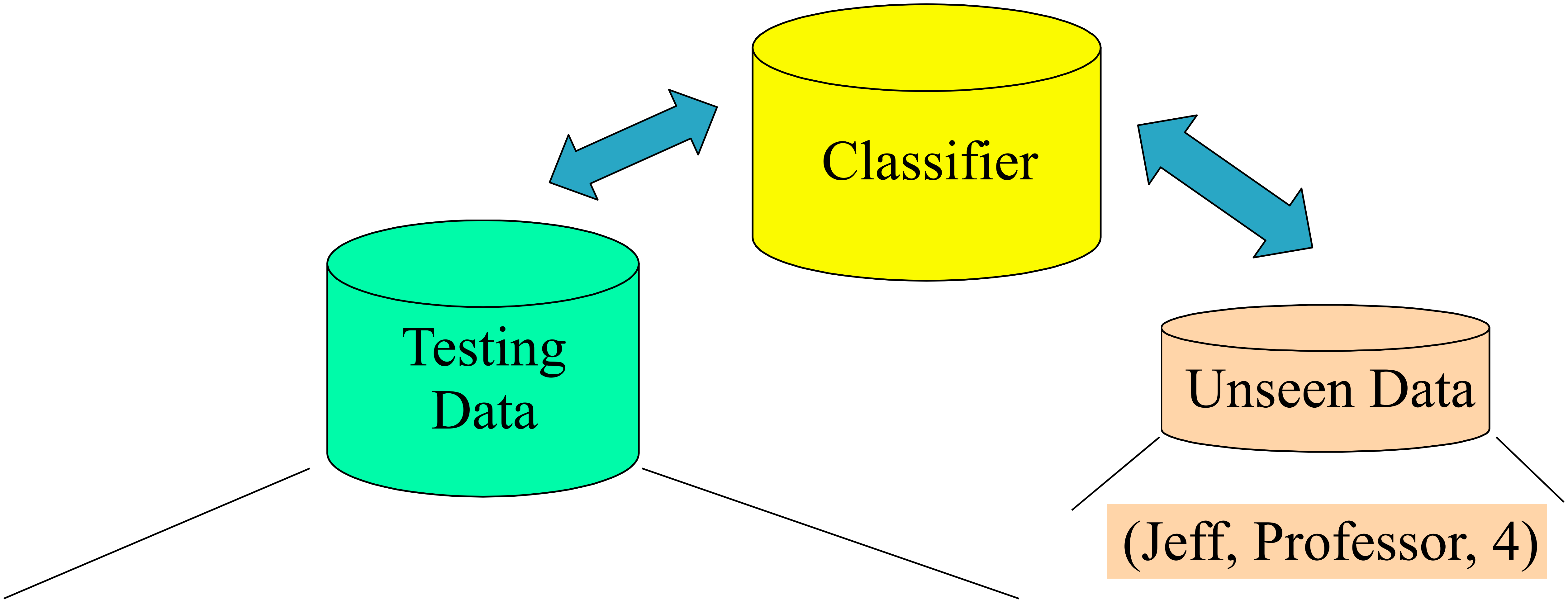
- **Reinforcement learning**
  Learning well-performing behavior from state observations and rewards

Model construction

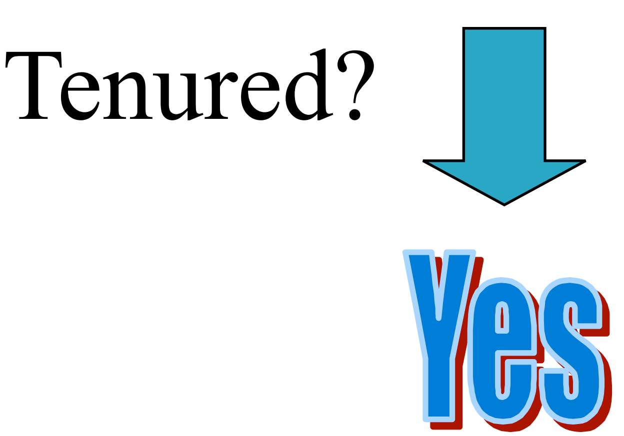


NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Using the model



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes



Classification vs prediction

- Classification: binary or nominal labels
  - Examples: pregnant or not, from which country, which type of road sign
- Prediction: continous labels
  - Examples: future stock price, life expectancy, distance to obstacle

Terminology (supervised learning)

- Each line of data: instance / data point / tuple
- The features of each instance: features / attributes
- That which should be learned: labels / targets
- Each instance has features and a label
- We train on the training set…
- …and test on the testing set

What's desirable?

- Accuracy
  classifier accuracy: predicting class label
  predictor accuracy: guessing value of predicted attributes

- Speed
  time to construct the model (training time)
  time to use the model (classification/prediction time)

- Robustness: handling noise and missing values

- Interpretability

- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Lazy vs Eager learning

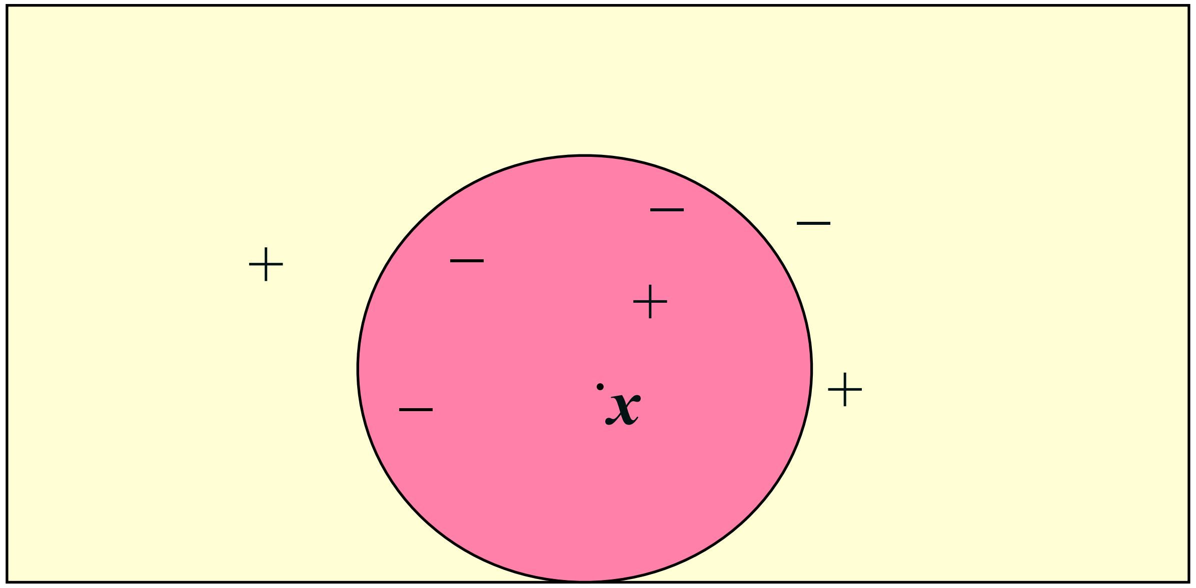
- Lazy learning: Simply stores training data (or only minor processing) and waits until it is given a test tuple
- Eager learning: Given a training set, constructs a classification model (smaller than the data) before receiving new data to classify
- Lazy: less time in training but more time in predicting

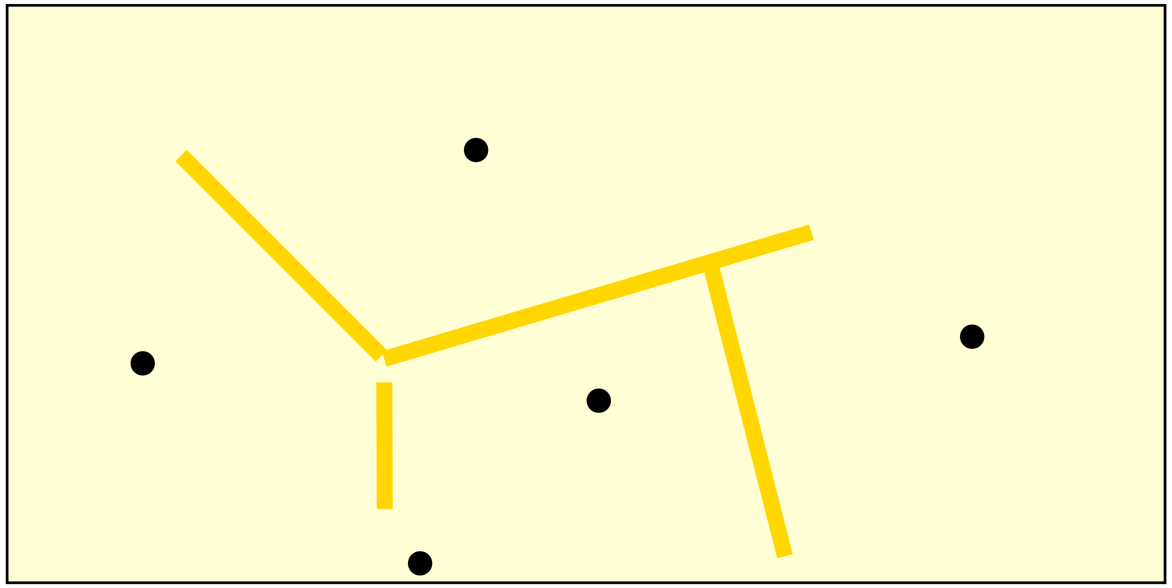
What's the simplest imaginable working classifier?

k-Nearest Neighbor Classification

- Simply look at the k instances in the training data which are closest to the instance you want to classify

- Choose the median/mean/mode of those values





k-Nearest Neighbor Classification

- All instances correspond to points in the n-D space
- The nearest neighbour is defined in terms of Euclidean distance, dist(X_1, X_2)
- Target function could be discrete- or real-valued
- For discrete-valued, k-NN returns the most common value among the k training examples nearest to xq
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples

k-Nearest Neighbor Classification

- Distance-weighted nearest neighbor algorithm: Weigh the contribution of each of the k neighbors according to their distance to the query x_q, and give greater weight to closer neighbors $w \equiv \frac{1}{d(x_q, x_i)^2}$

- Robust to noisy data by averaging k-nearest neighbors

- *Curse of dimensionality:* distance between neighbors could be dominated by irrelevant attributes

- To overcome it, stretch or shrink axes or eliminate the least relevant attributes

Distances

- Euclidean distance for continuous attributes

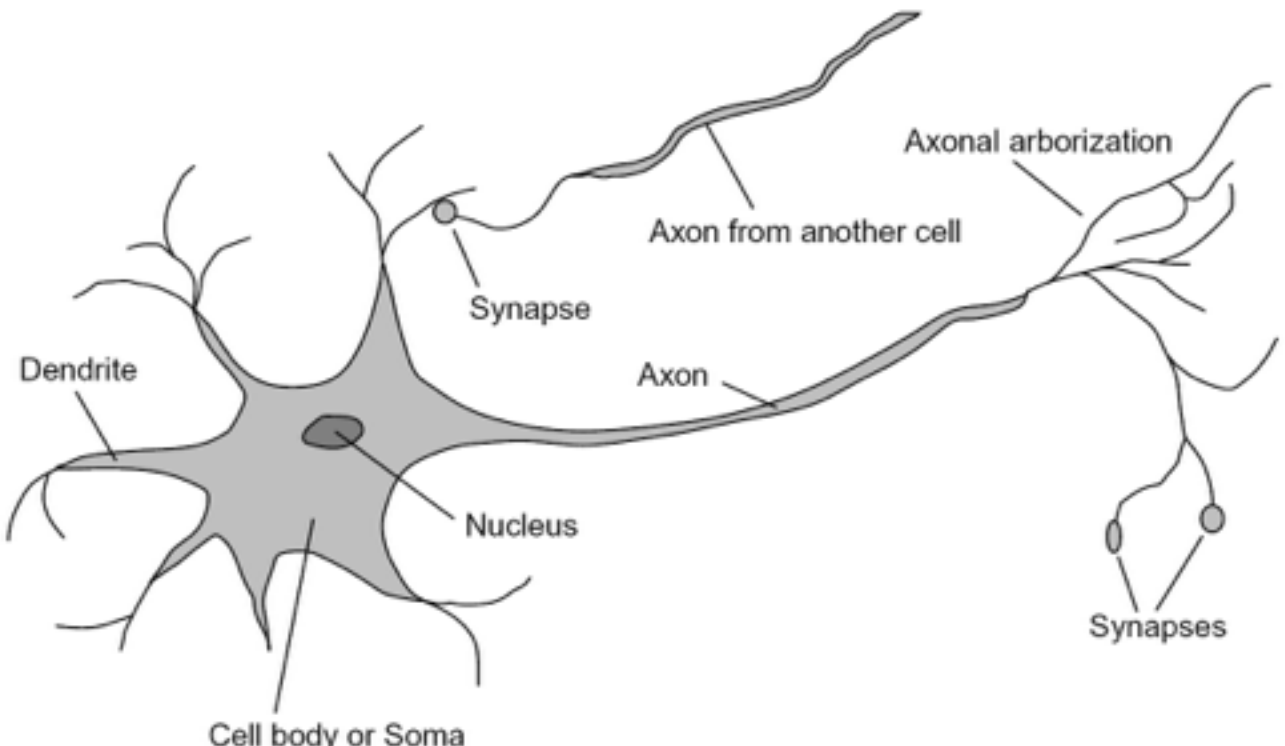
$$d(i,j)=\sqrt{(|x_{i_1}-x_{j_1}|^2+|x_{i_2}-x_{j_2}|^2+...+|x_{i_p}-x_{j_p}|^2)}$$

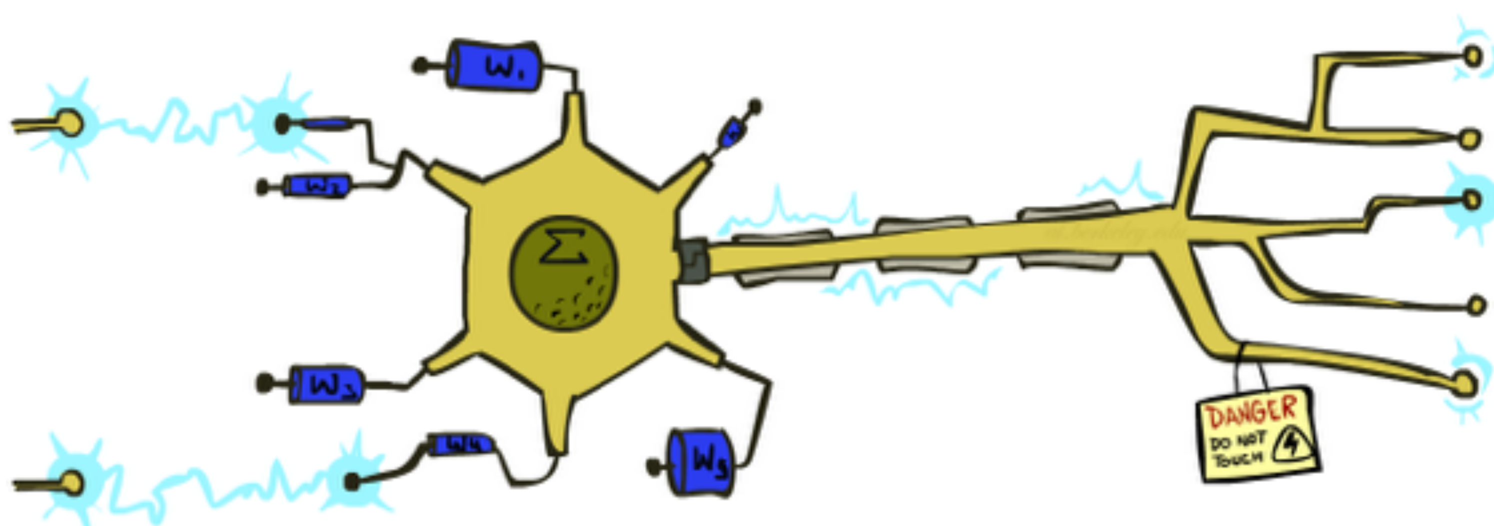
- Hamming distance for binary/nominal attributes: how many of the attributes differ

Perceptrons

- Early attempt at “neural networks” - copying neurophysiology to produce a learning machine
- Very simple and fast learning algorithm for linearly separable problems
- The basis for many more advanced neural network variants, such as Multilayer Perceptrons (and, by extension, deep networks)

Very loose biological analogy



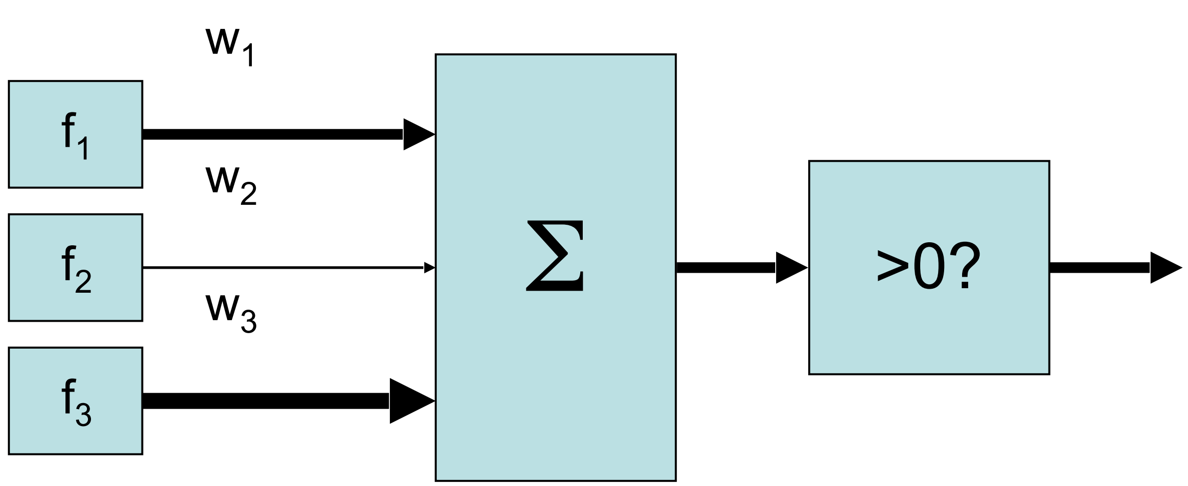


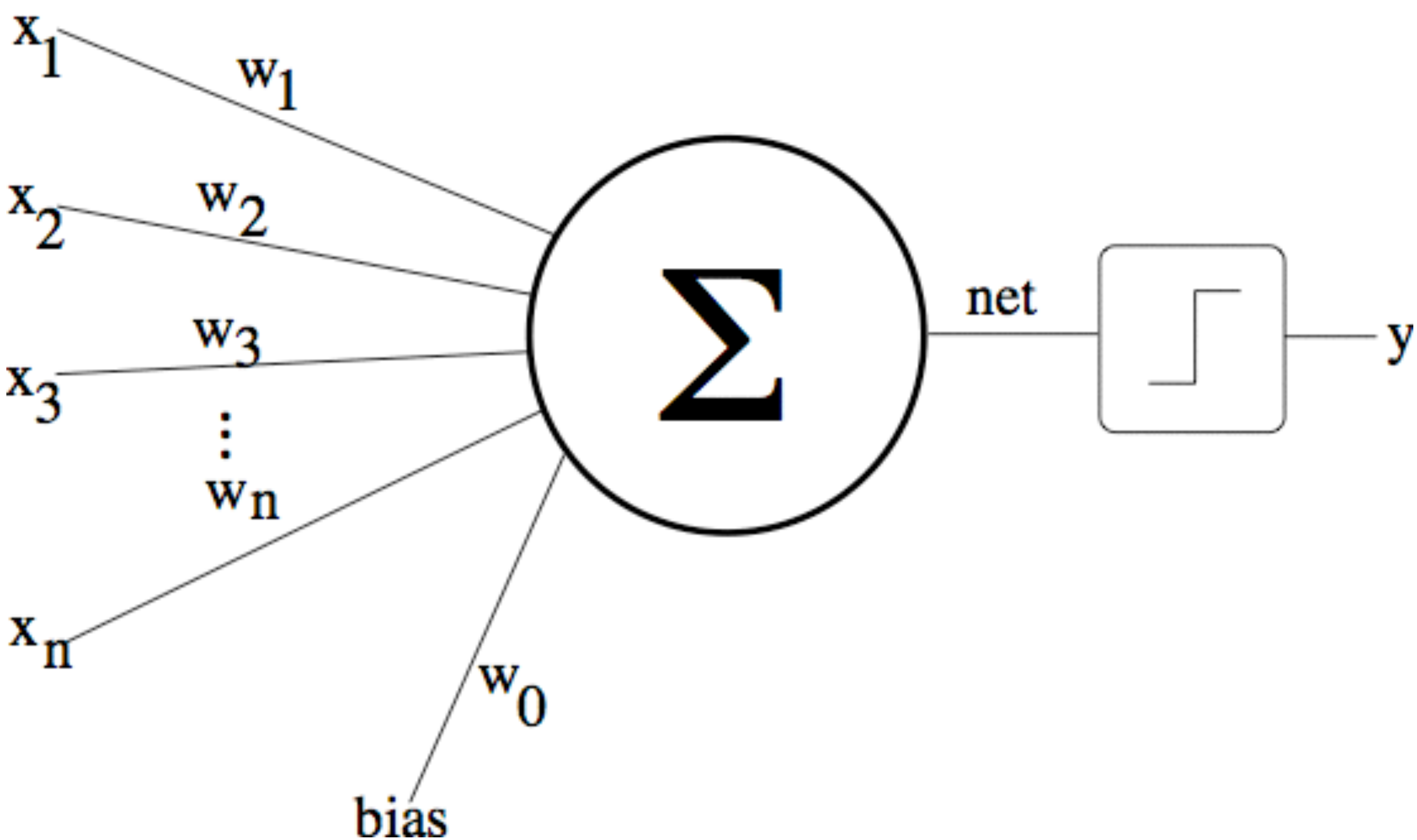
Perceptrons are linear classifiers

- Inputs are feature values
- Each feature has a weight
- Sum is the activation

$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

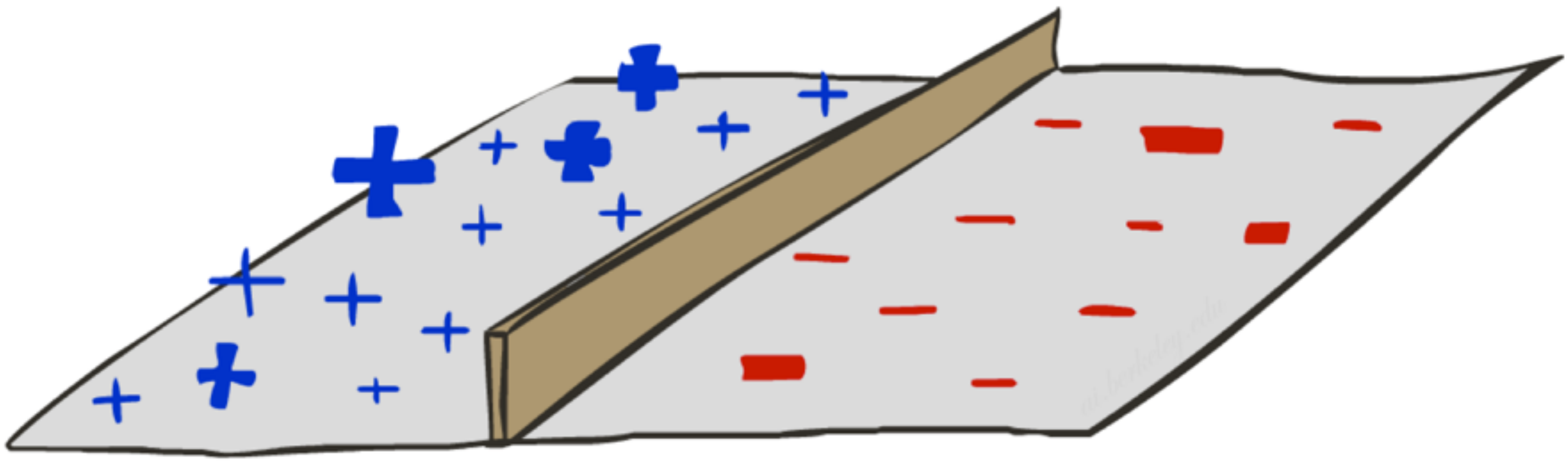
- If the activation is:
  - Positive, output +1
  - Negative, output -1





$$f(x)=\begin{cases}1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i + b > 0\\ 0 & \text{else}\end{cases}$$

Decision surface

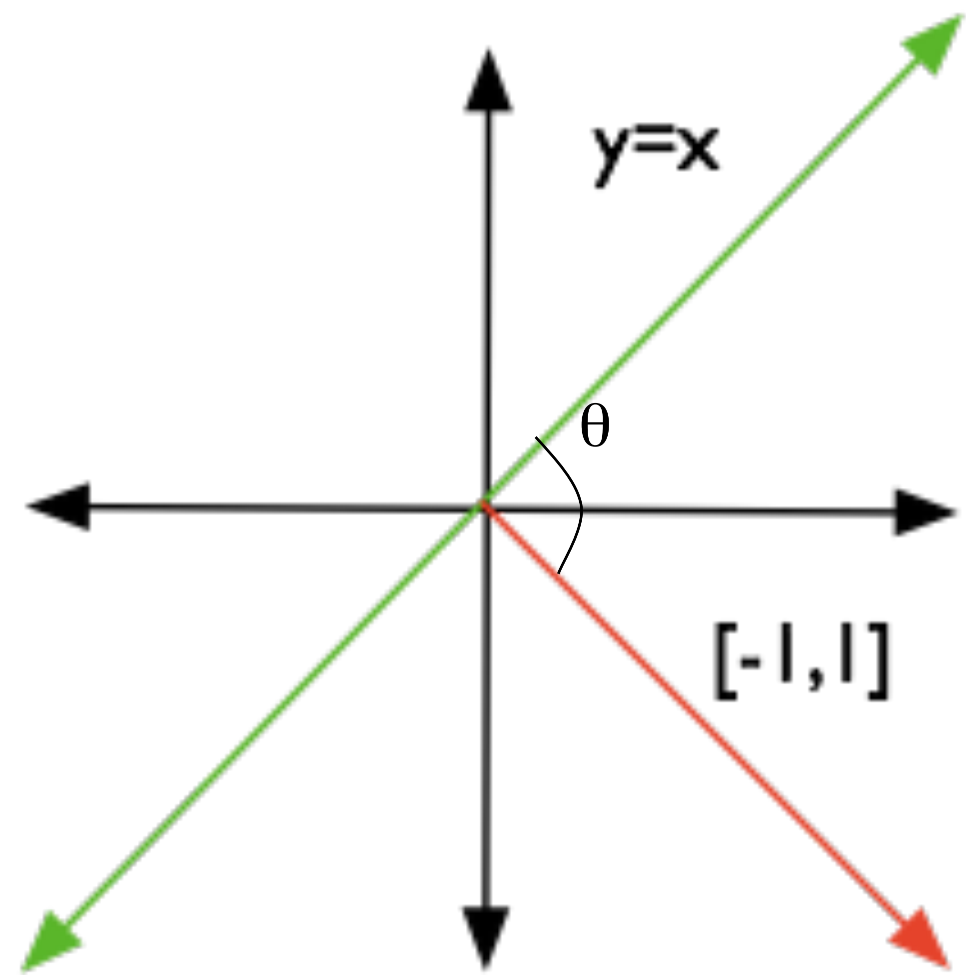


Decision surface

$$y = x$$
$$0 = x - y$$
$$0 = [1, -1] \left[\begin{array}{c} x \\ y \end{array} \right]$$

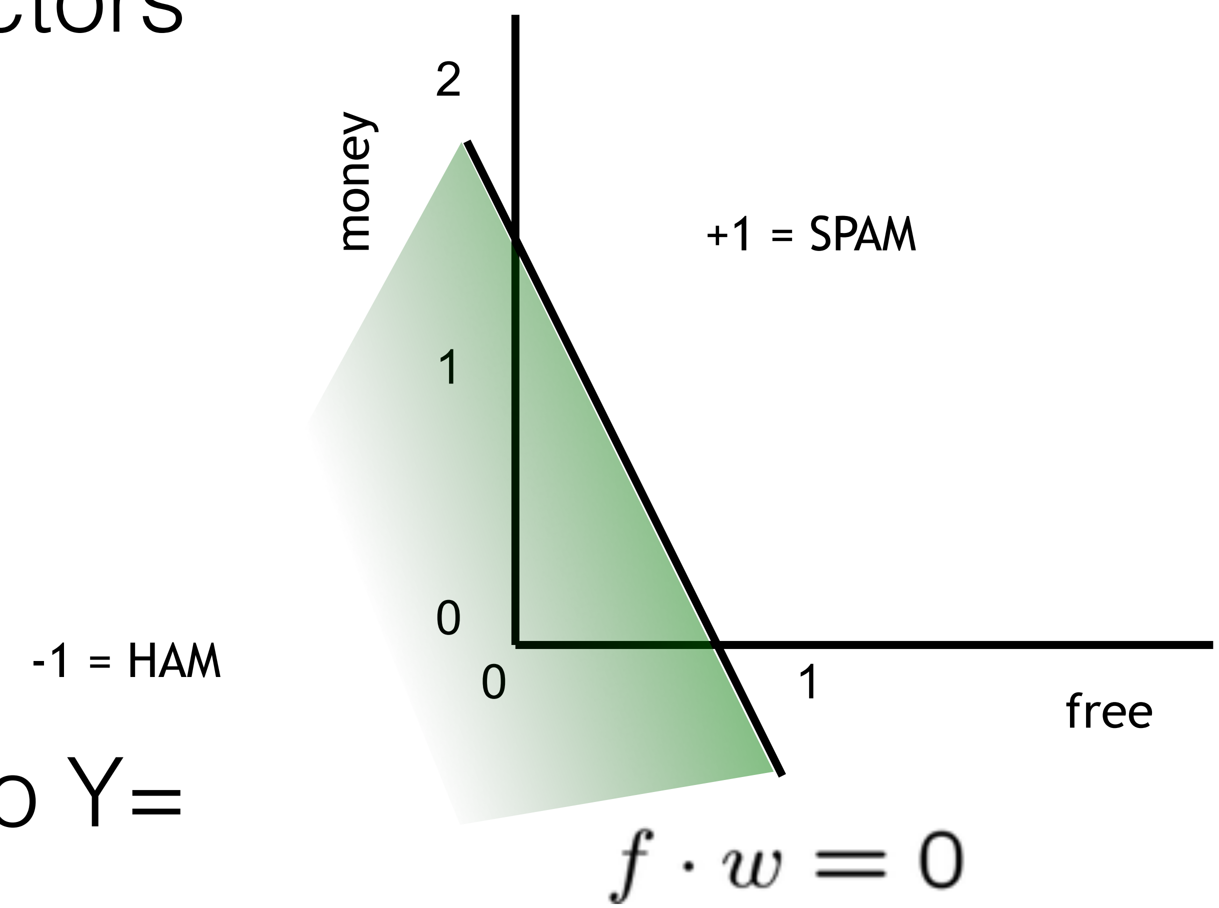
In general a **hyperplane** is defined by

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



Model usage

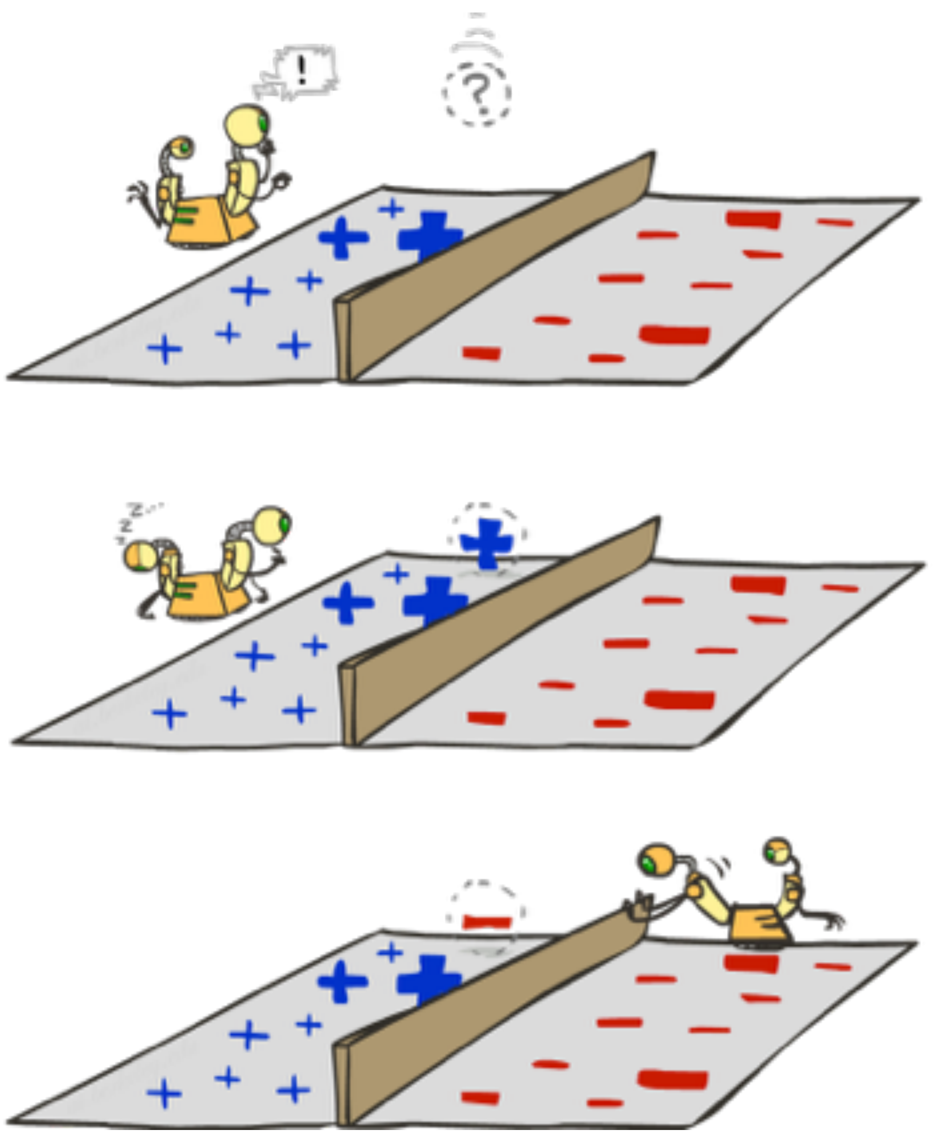
- In the space of feature vectors
  - Examples are points
  - Any weight vector is a hyperplane
  - One side corresponds to Y= +1
  - Other corresponds to Y=-1



Model training

Algorithm

- Start with random weights. For each training instance:
- Classify with current weights
- If correct (i.e., y=y*), no change!
- If wrong: adjust the weight vector



Algorithm

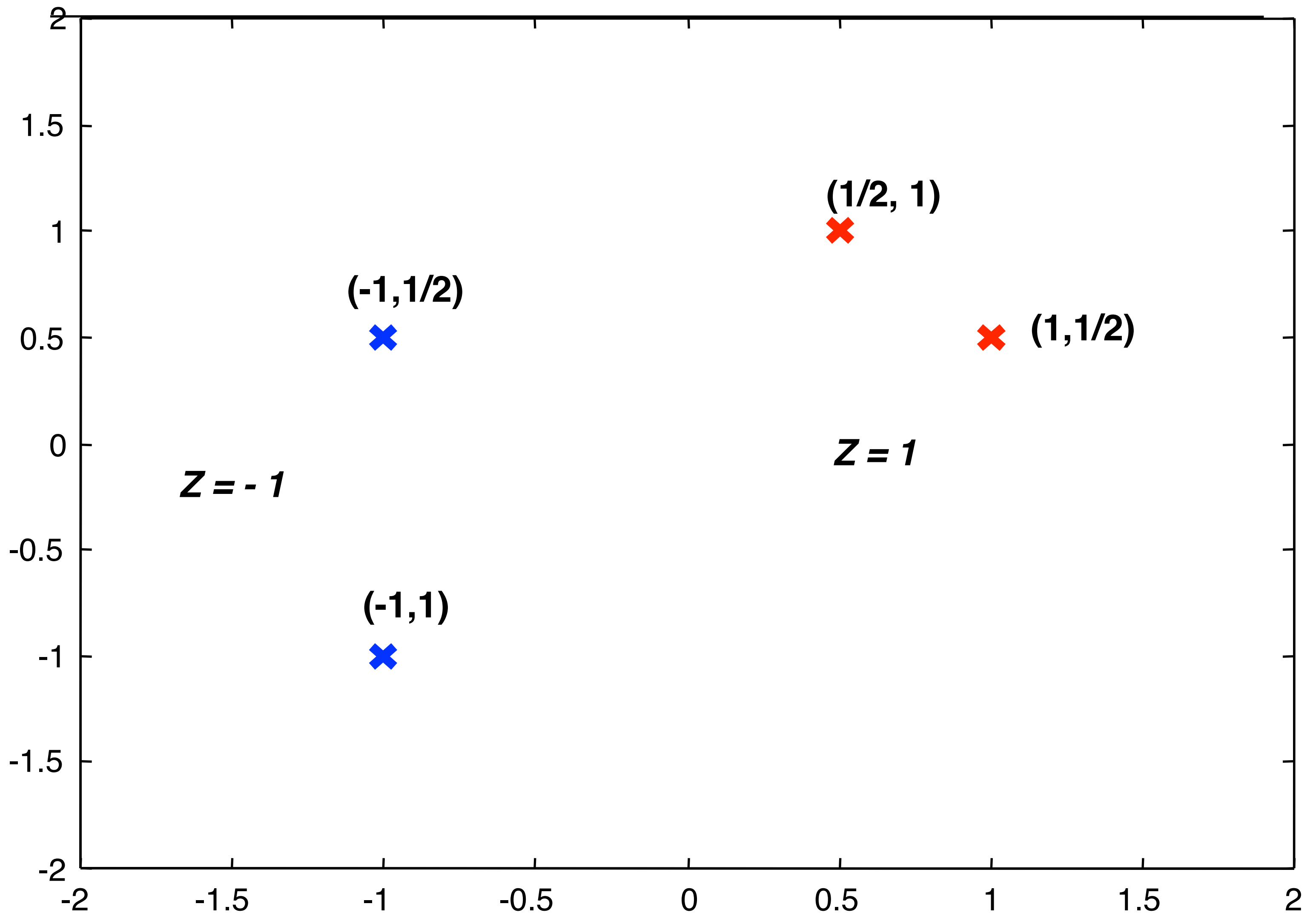
Input : list of n training examples $(x_0, d_0) \ldots (x_n, d_n)$
where $\forall i : d_i \in \{+1, -1\}$
Output : classifying hyperplane w

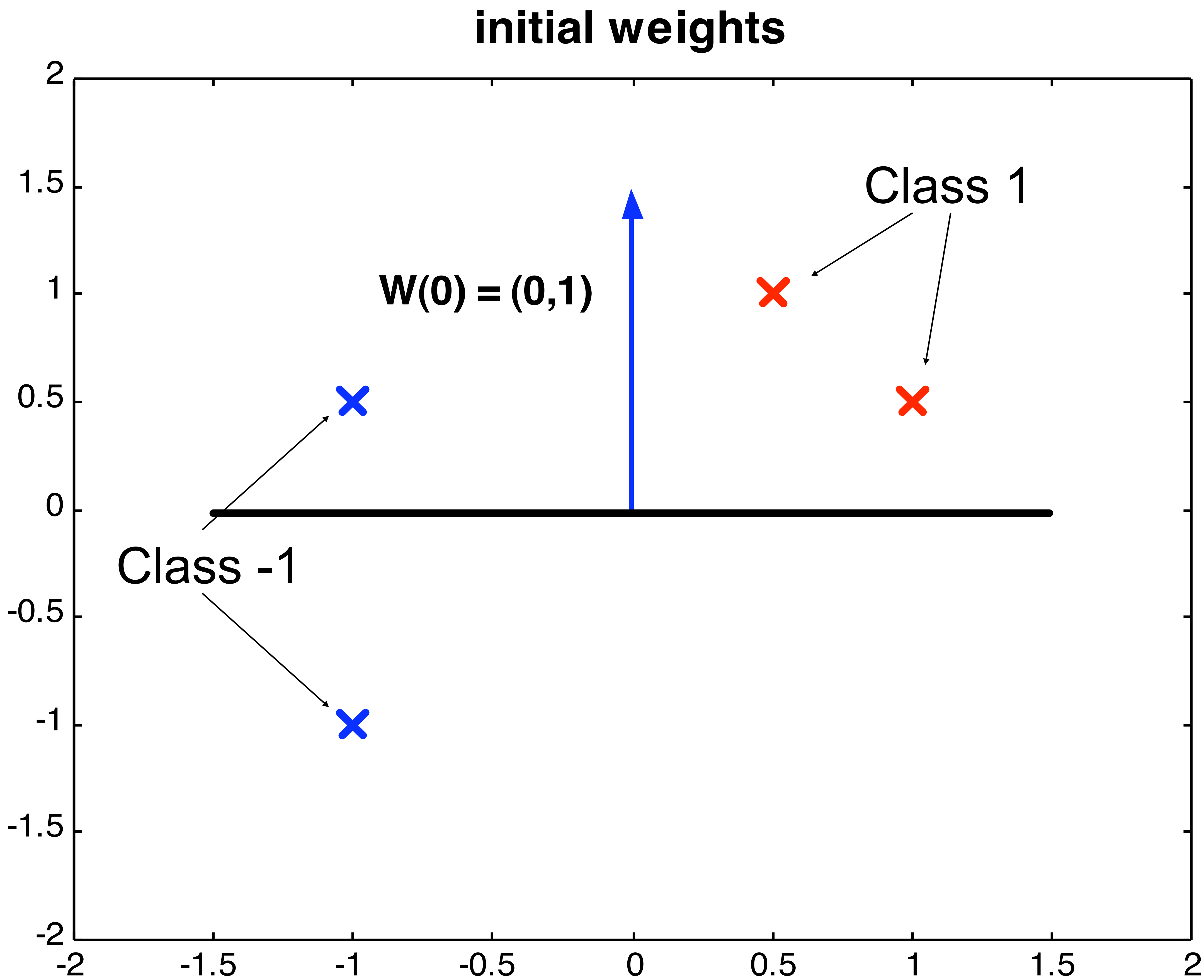
Algorithm :
Randomly initialize w;
While makes errors on training set **do**
  for $(x_i d_i)$ **do**
    let $y_i = sign(w \cdot x_i)$;
    if $y_i \neq d_i$ **then**
      $w \leftarrow w + \eta d_i x_i$;
    end
  end
end

x and w are vectors;
i is the instance index

A simple example

4 linearly separable points



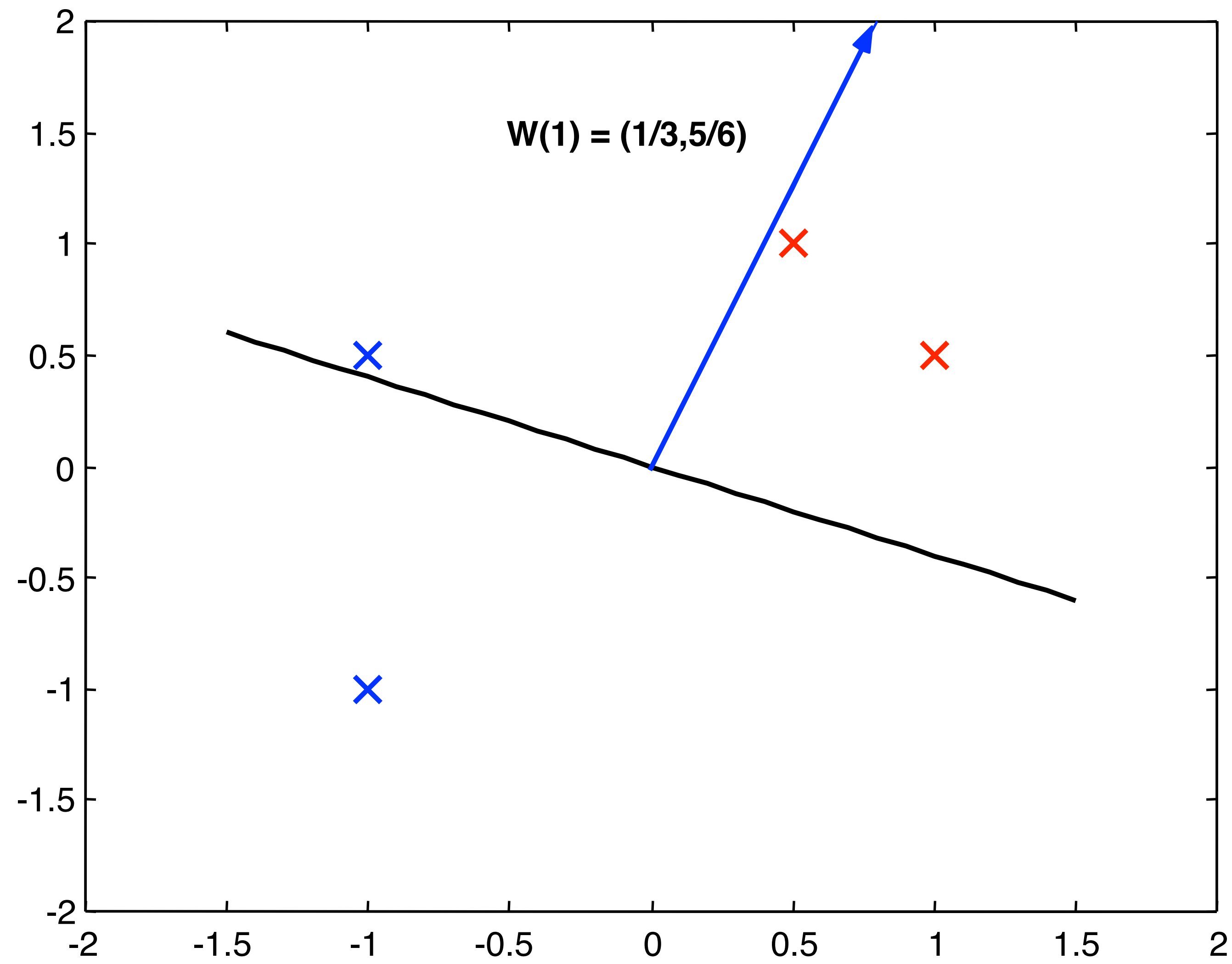

initial weights
W(0) = (0,1)
Class 1
Class -1
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Updating Weights

Upper left point (-1,1/2) is wrongly classified

$$
\begin{aligned}
&x = (-1, 1/2) \\
&d = -1 \\
&\eta = 1/3, w(0) = (0,1) \\
&w(1) \leftarrow w(0) + \eta d x \\
&w(1) = (0,1) + 1/3 * (-1) * (-1, 1/2) \\
&\qquad = (0,1) + 1/3 * (1, -1/2) \\
&\qquad = (1/3, 5/6)
\end{aligned}
$$

first correction

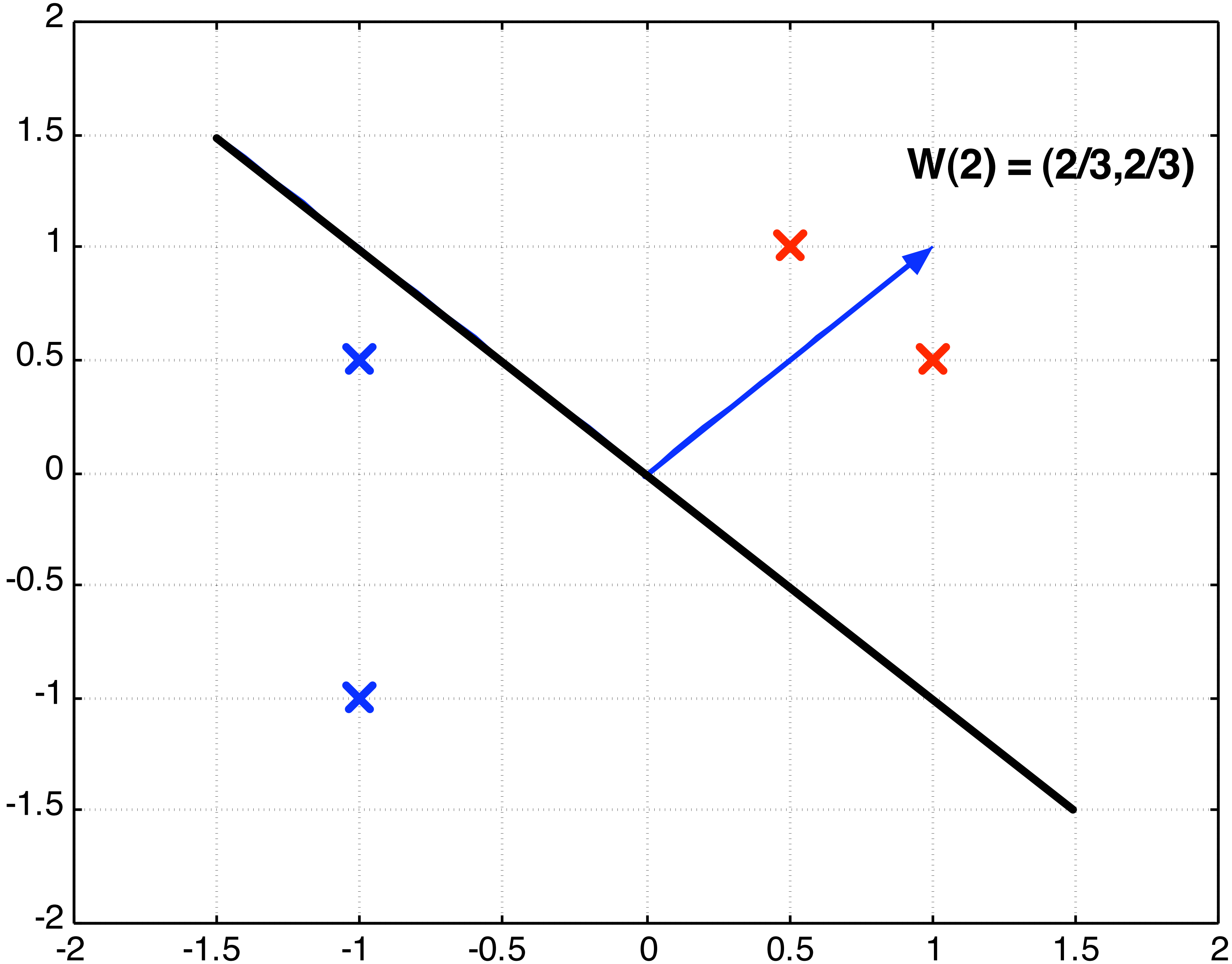


Updating Weights, Ctd

Upper left point is still wrongly classified

$$
\begin{aligned}
&x = (-1, 1/2) \\
&d = -1 \\
&w(2) \leftarrow w(1) + \eta d x \\
&w(2) = (1/3, 5/6) + 1/3 * (-1) * (-1, 1/2) \\
&\qquad = (1/3, 5/6) + 1/3 * (1, -1/2) \\
&\qquad = (2/3, 2/3)
\end{aligned}
$$

second correction



If we have multiple classes

- A weight vector for each class:

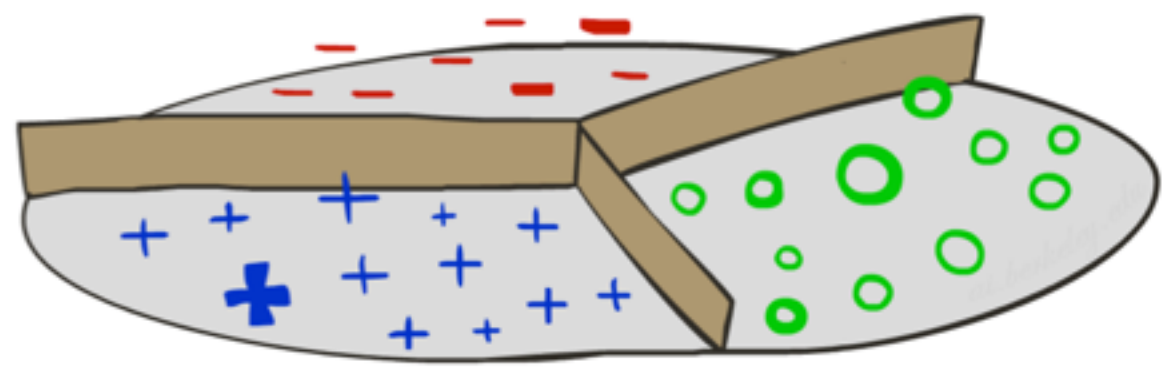
$$w_y$$

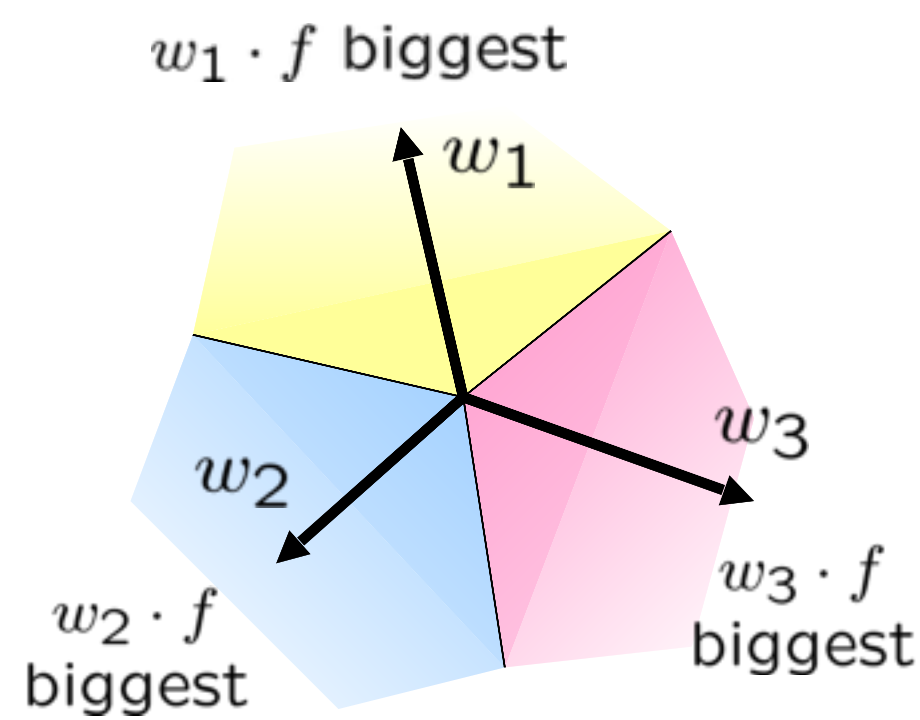
- Score (activation) of a class y:

$$w_y \cdot f(x)$$

- Prediction highest score wins

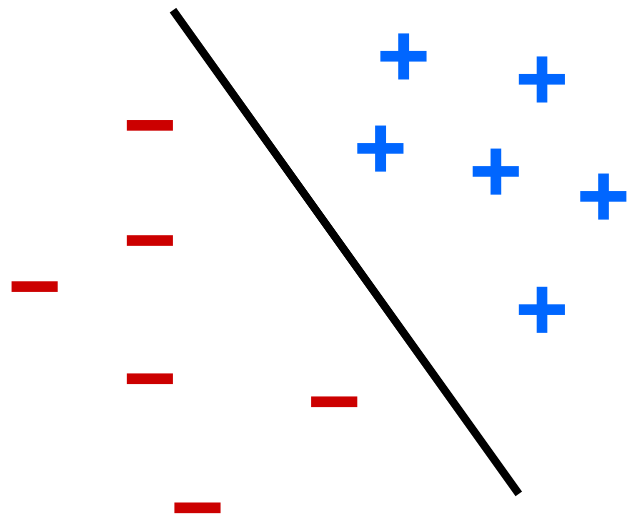
$$y = \arg\max_y \; w_y \cdot f(x)$$





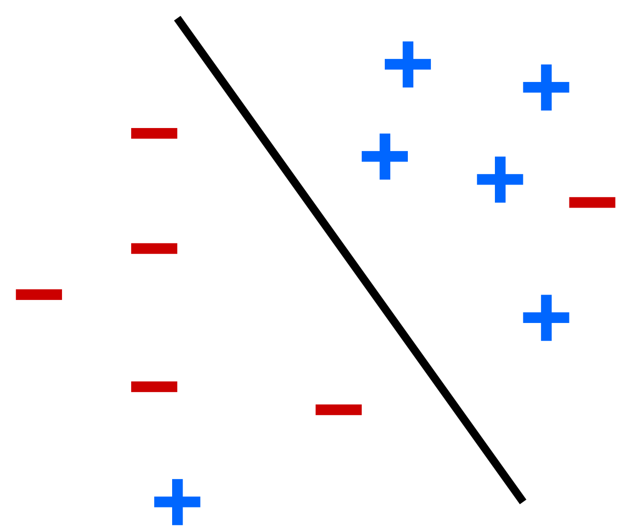
Properties of Perceptrons

- Separability: true if some parameters get the training set perfectly correct

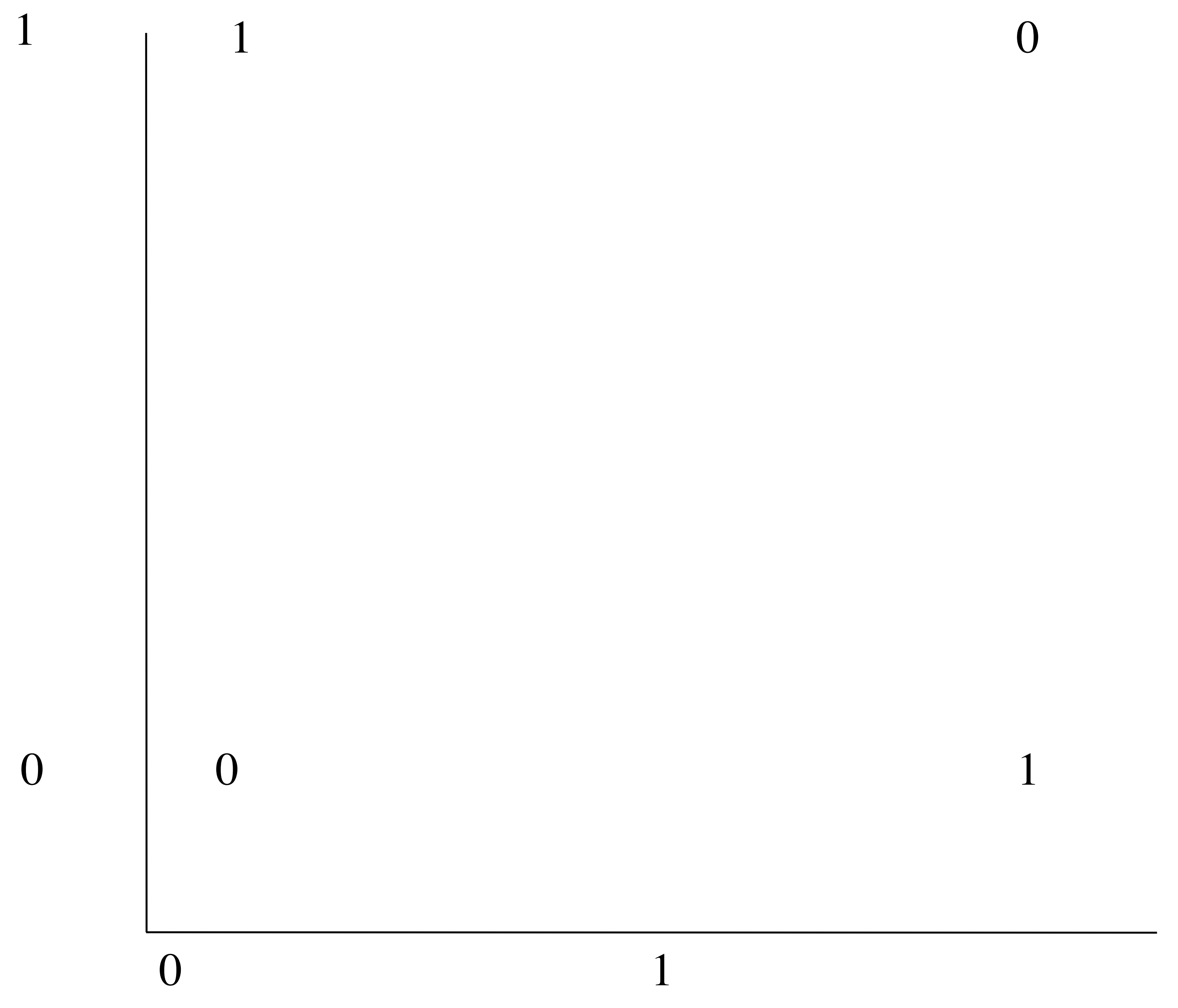


- Convergence: if the training is separable, perceptron will eventually converge (binary case)
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability

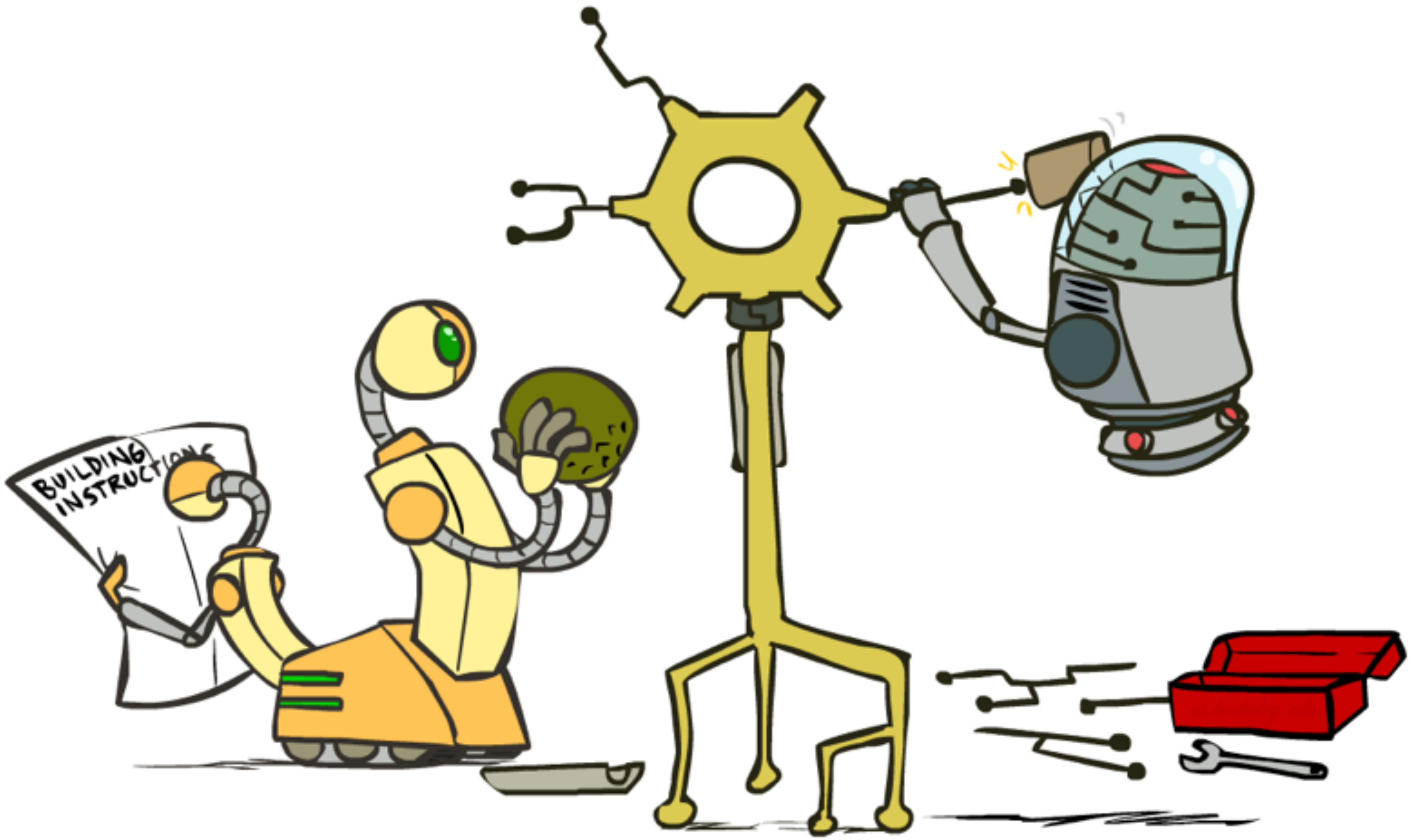
$$\text{mistakes} < \frac{k}{\delta^2}$$



Problem: learn this!

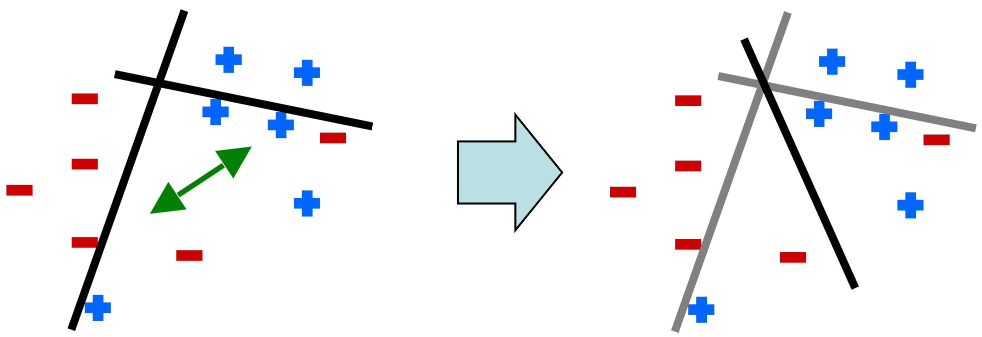


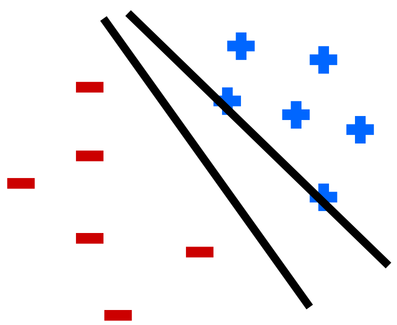
Improving the perceptron

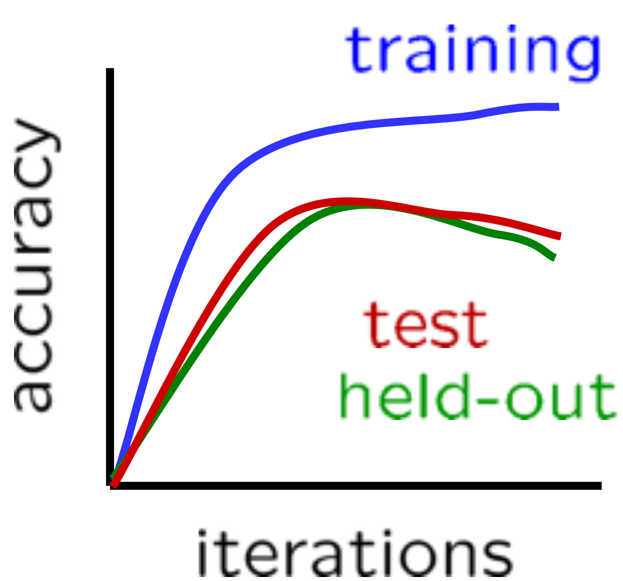


Perceptron problems

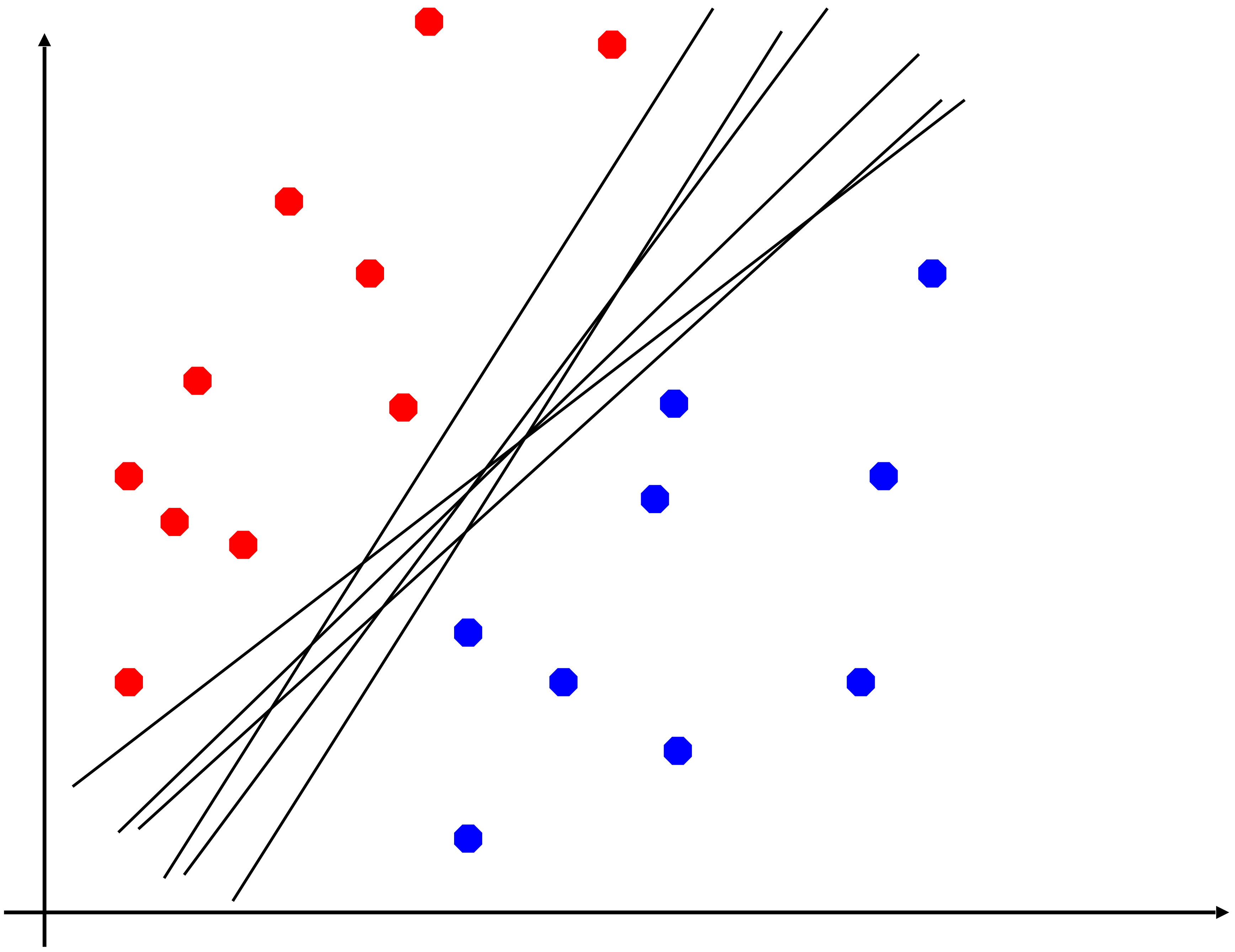
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)
- Mediocre generalization: finds a "barely" separating solution
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





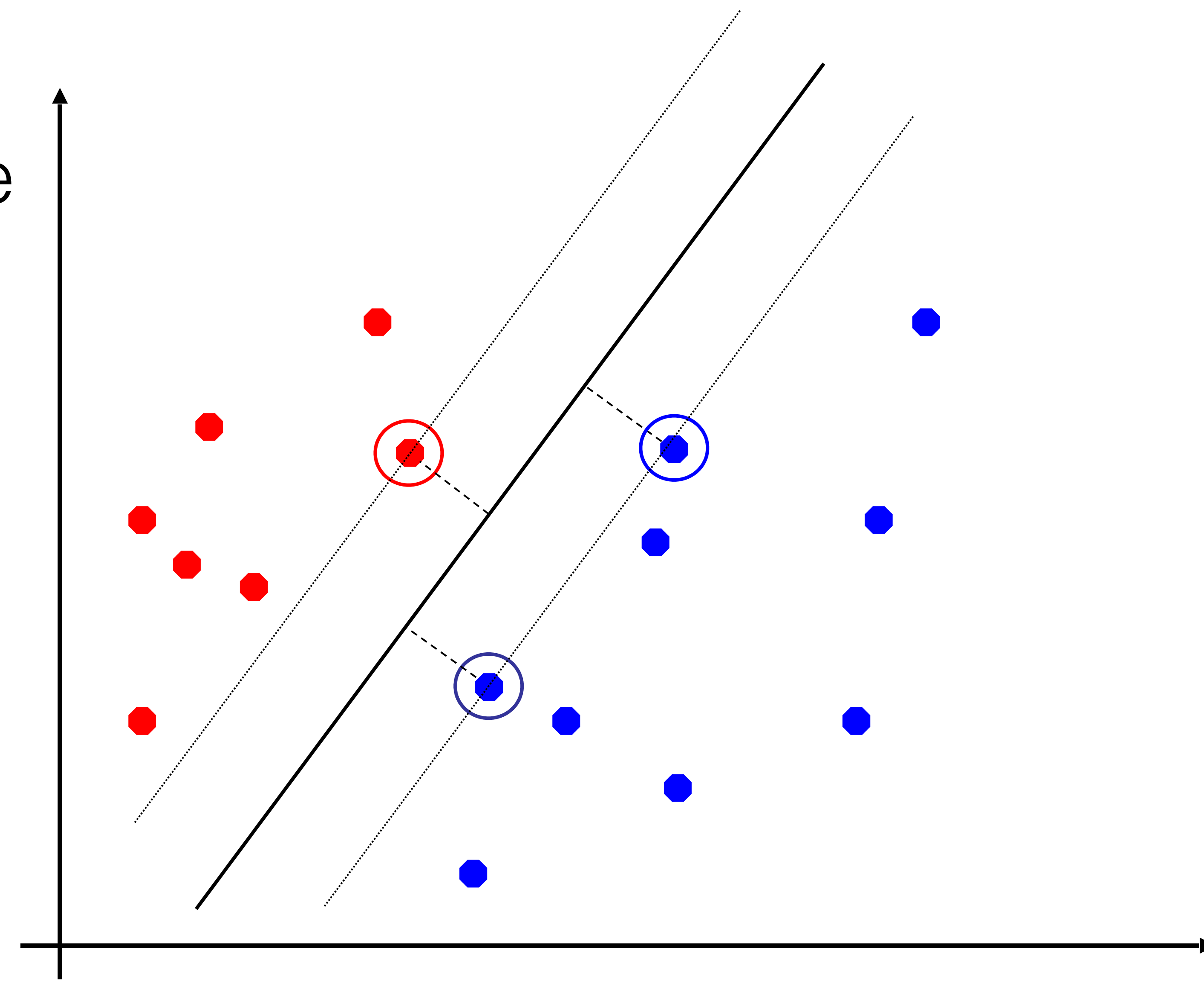


Which of these separators is optimal?

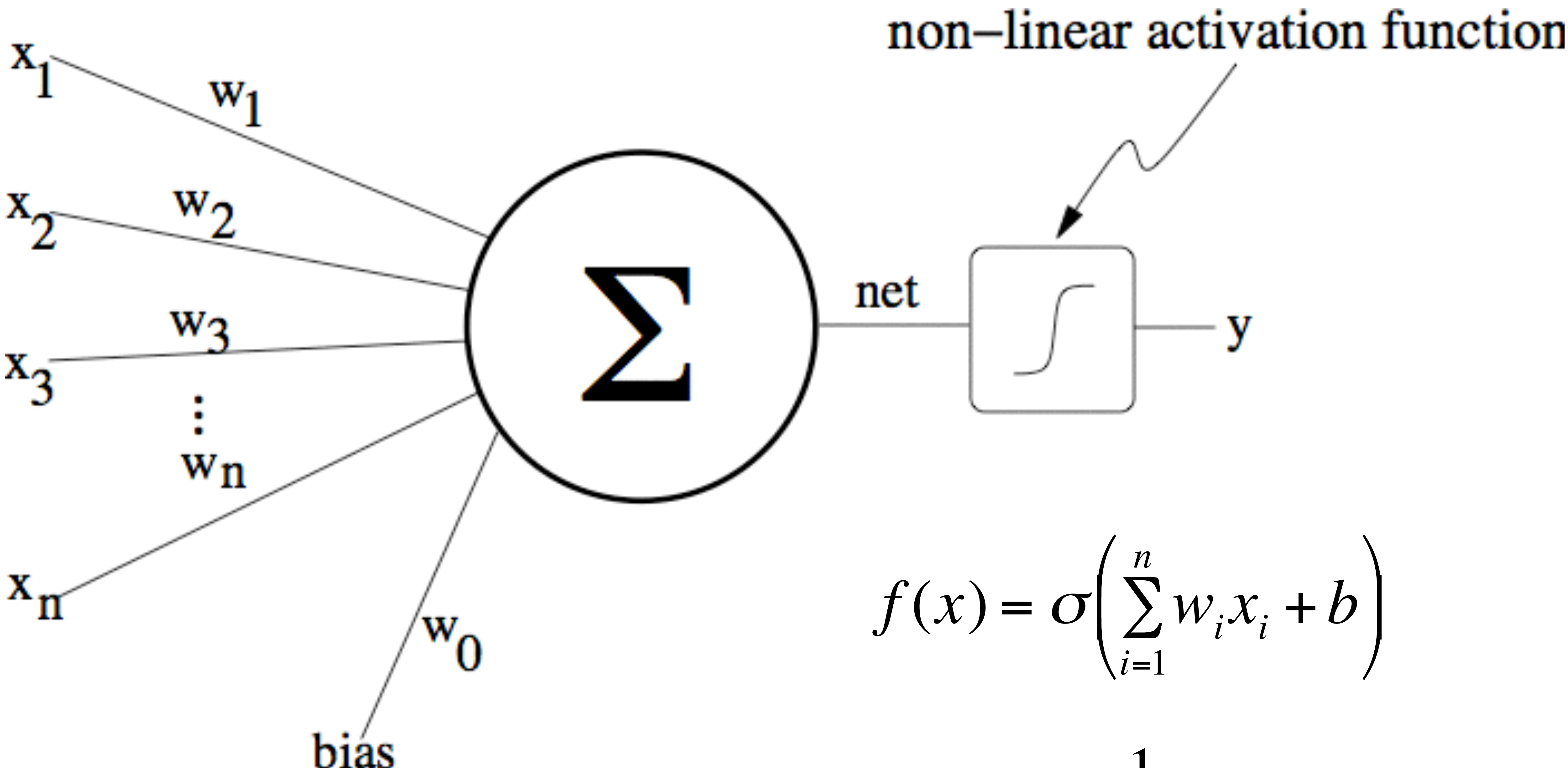


Support vector machines

- Maximizing the margin: good according to intuition, theory, practice
- Only support vectors matter; other training examples are ignorable
- Support vector machines (SVMs) find the separator with max margin



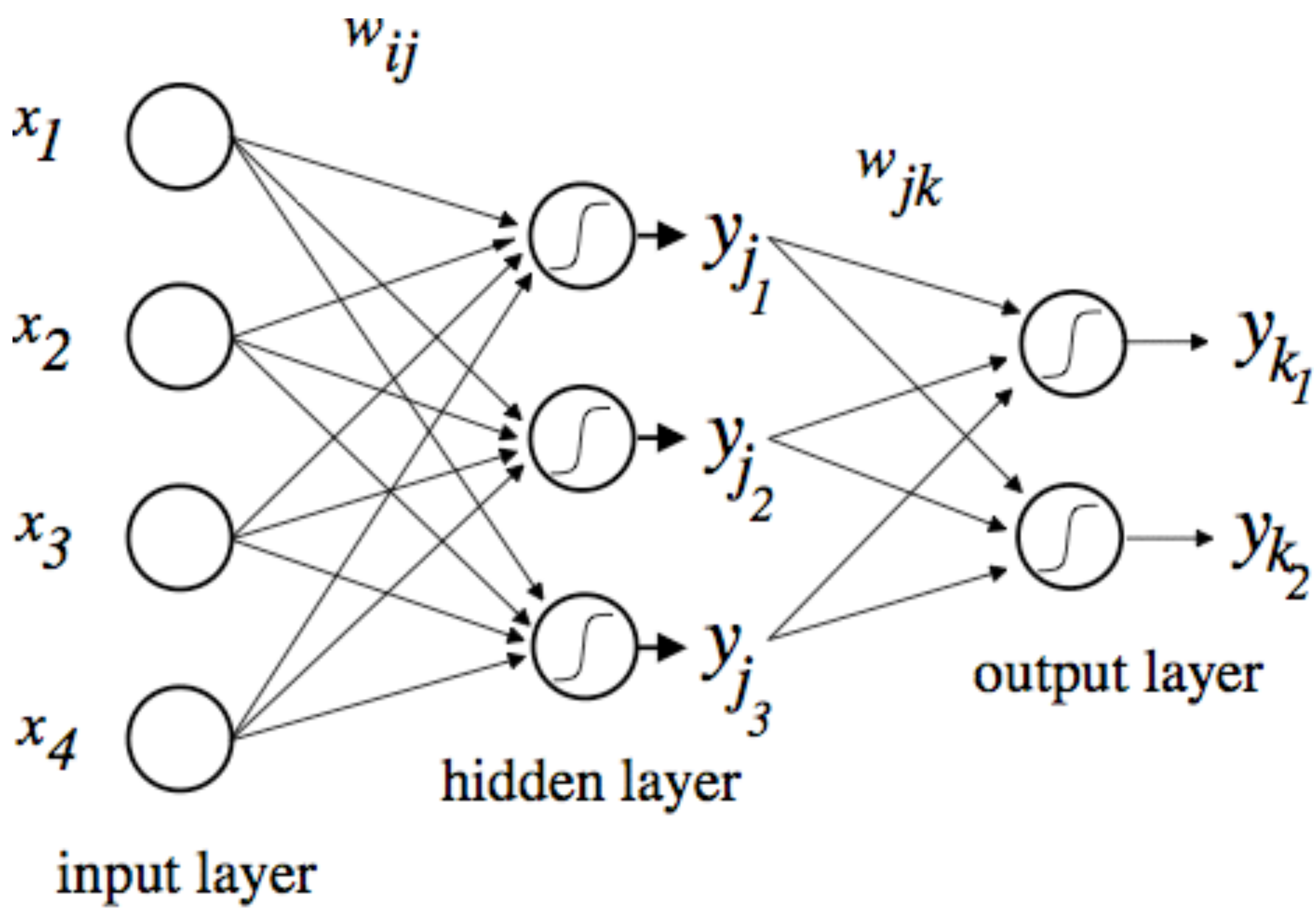
Non-Linear Neuron



$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

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

Multi-layer Perceptron (MLP)



Backpropagation

- Forward Pass: present training input pattern to network and activate network to produce output (can also do in batch: present all patterns in succession)
- Backward Pass: calculate error gradient and update weights starting at output layer and then going back