Week 9
Supervised learning

Artificial Intelligence
CSC 480
Rodrigo Canaan
rcanaan@calpoly.edu

Adapted with permission from
Julian Togelius
julian.togelius@nyu.edu

Contents

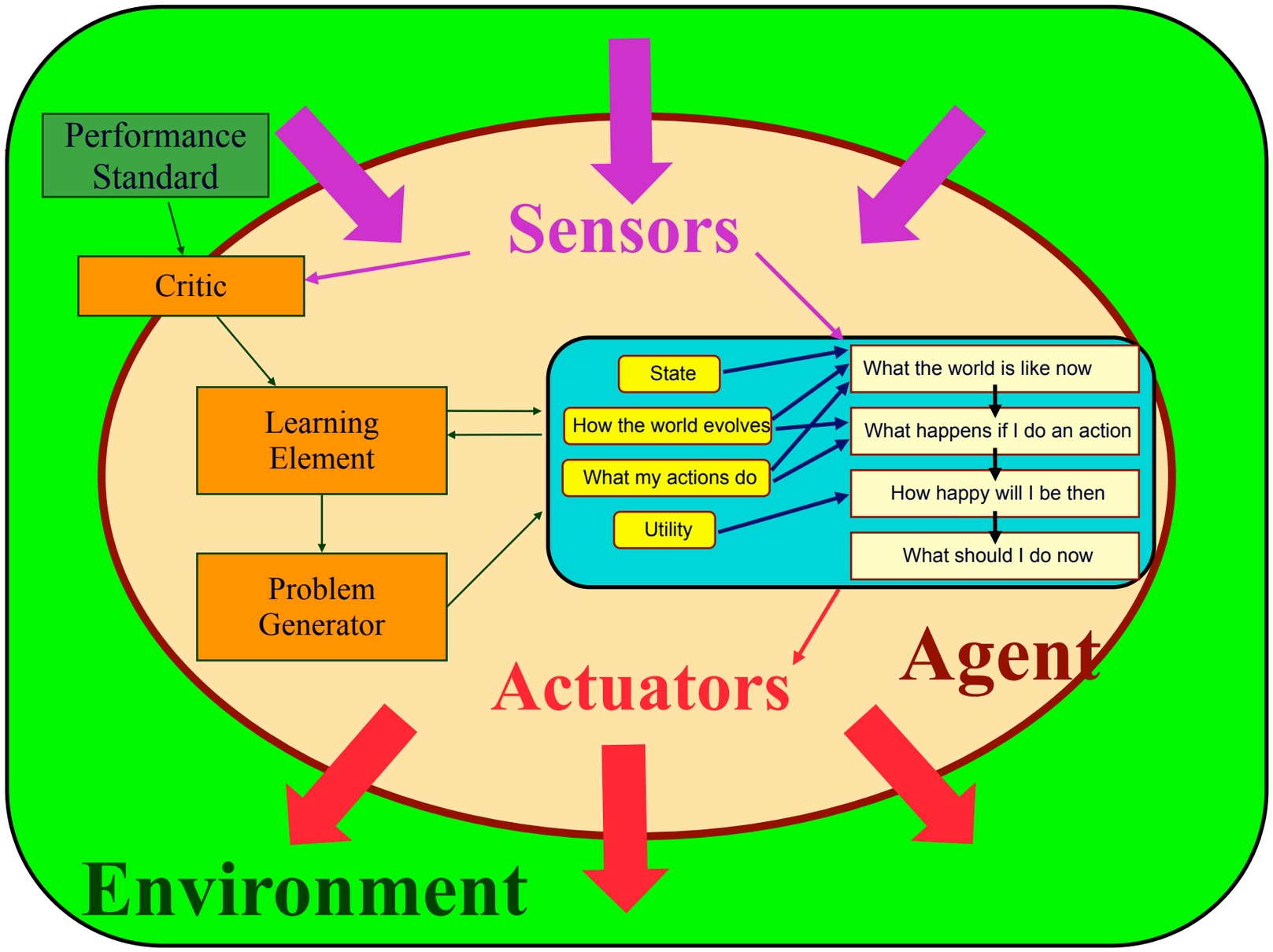
- Learning in intelligent agents
- Supervised learning and other forms of learning
- Eager and lazy learning
- k-Nearest Neighbor
- Decision Trees
- Perceptrons

Why learning?

Three main reasons:

- Designers might not anticipate all situations the agent might face (e.g. different mazes)
- Designers might not anticipate all changes over time to the environment (e.g. game patches, law changes)
- Designers might have no idea how to program a solution (e.g. face recognition)

A learning agent (AIMA ch.2)



What could be learned?

```
Welcome to
                EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                EE      LL       II       ZZ   AA   AA
                EEEEE   LL       II     ZZZ    AAAAAAA
                EE      LL       II    ZZ      AA   AA
                EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

Eliza is a mock Rogerian psychotherapist.
The original program was described by Joseph Weizenbaum in 1966.
This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

What could be learned?

- What words most likely follow other words
- How syntax works in a language
- How to detect emotion in a sentence
- How to map words from one language to another
- Speech recognition / synthesis
- How to look up basic facts online (bitcoin price)

What could be learned?

What could be learned?

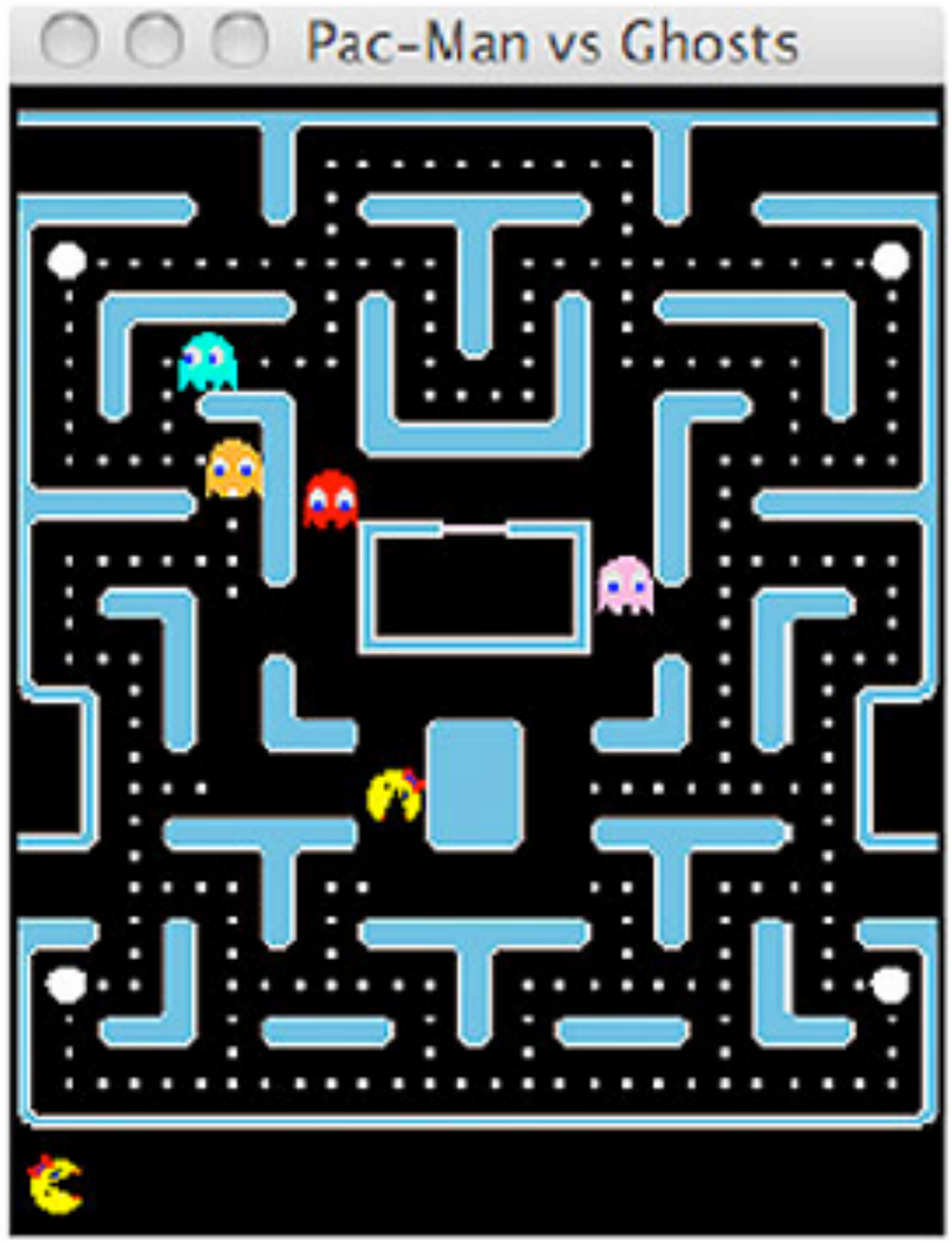
- How to drive from point A to point B, without hitting pedestrians or breaking laws
- What a human (or a cat, or bush) looks like
- The meaning of each traffic sign
- Which routes from A to B are actually fastest
- How far back you want your seat, temperature for the AC, favorite radio channel…
- Estimating distances

What could be learned?

What could be learned?

- 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?

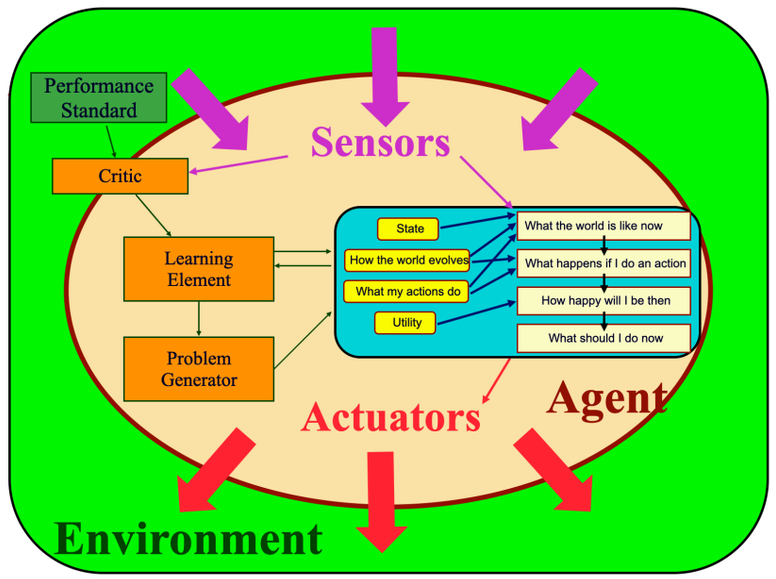


What could be learned?

- How ghosts move
- The value of a state
- Dangerous positions
- What action to take in a specific situation

What could be learned?

In general, anything in the blue box



- Mapping from states to actions
- Relevant properties of a partially observable world
- How the world evolves
- The desirability (utility) of each state
- The desirability of each action
- Goals to be pursued

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

Meme Time!

- **"Pure" Reinforcement Learning** (cherry)
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning** (icing)
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Unsupervised/Predictive Learning** (cake)
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**
- **(Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)**

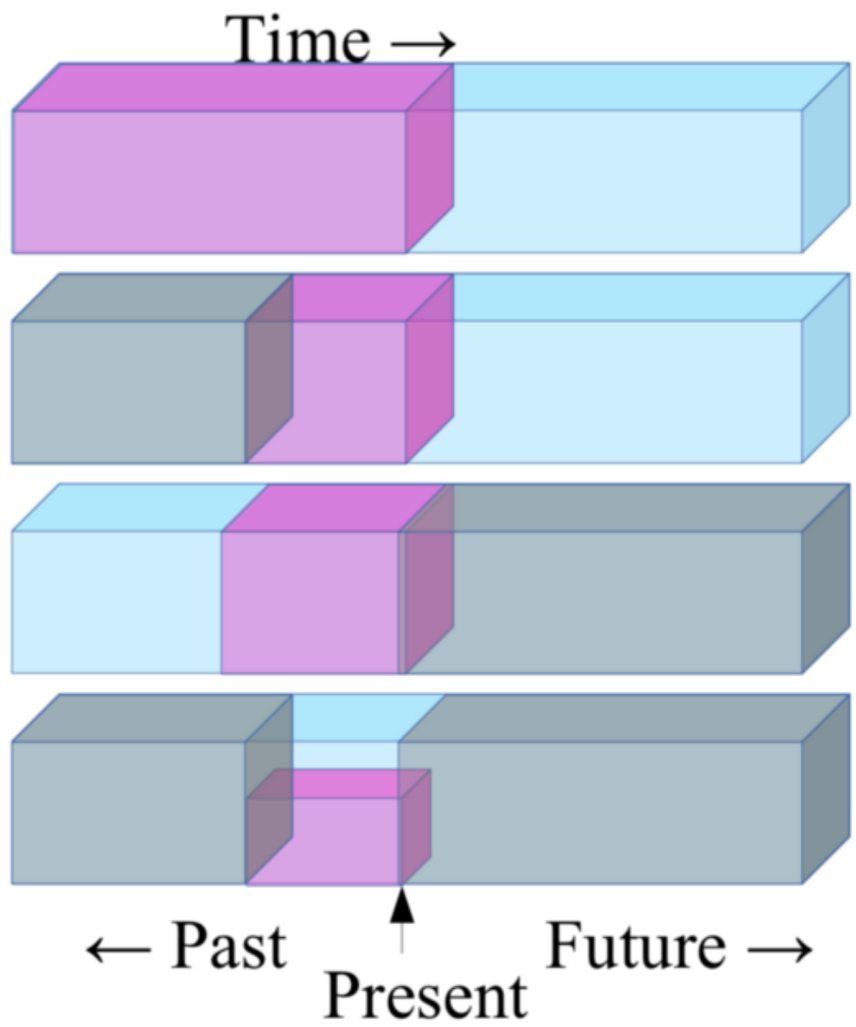
Source: Yann LeCun. Retrieved from https://medium.com/syncedreview/yann-lecun-cake-analogy-2-0-a361da560dae (access Nov 15 2021)

Self-Supervised Learning

Self-Supervised Learning

Y. LeCun

- **Predict any part of the input from any other part.**
- **Predict the future from the past.**
- **Predict the future from the recent past.**
- **Predict the past from the present.**
- **Predict the top from the bottom.**
- **Predict the occluded from the visible**
- **Pretend there is a part of the input you don't know and predict that.**



Source: Yann LeCun. Retrieved from https://medium.com/syncedreview/yann-lecun-cake-analogy-2-0-a361da560dae (access Nov 15 2021)

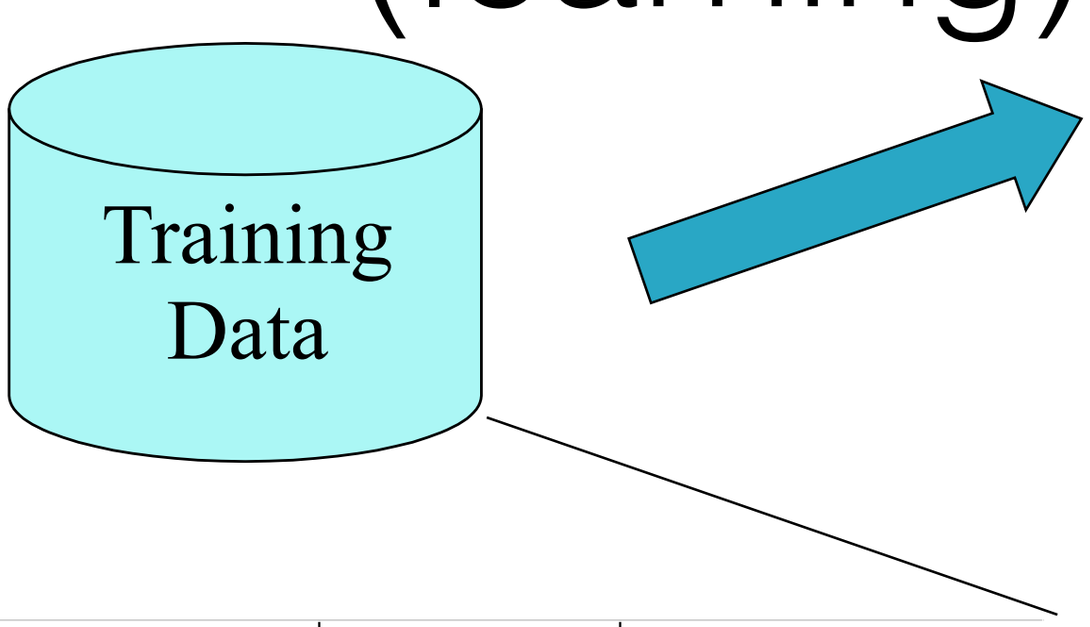
Types of learning

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

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

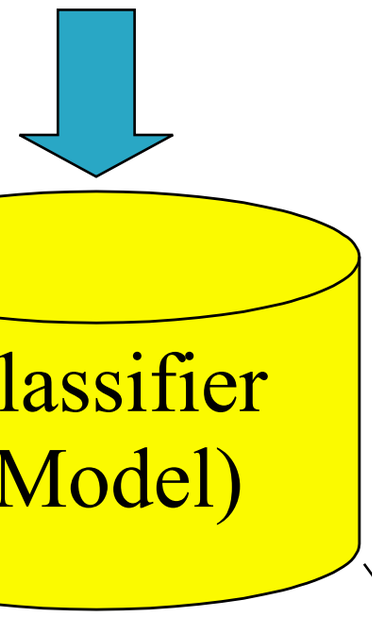
- **Reinforcement learning**
  Learning well-performing behavior from state observations and rewards

Model construction (learning)



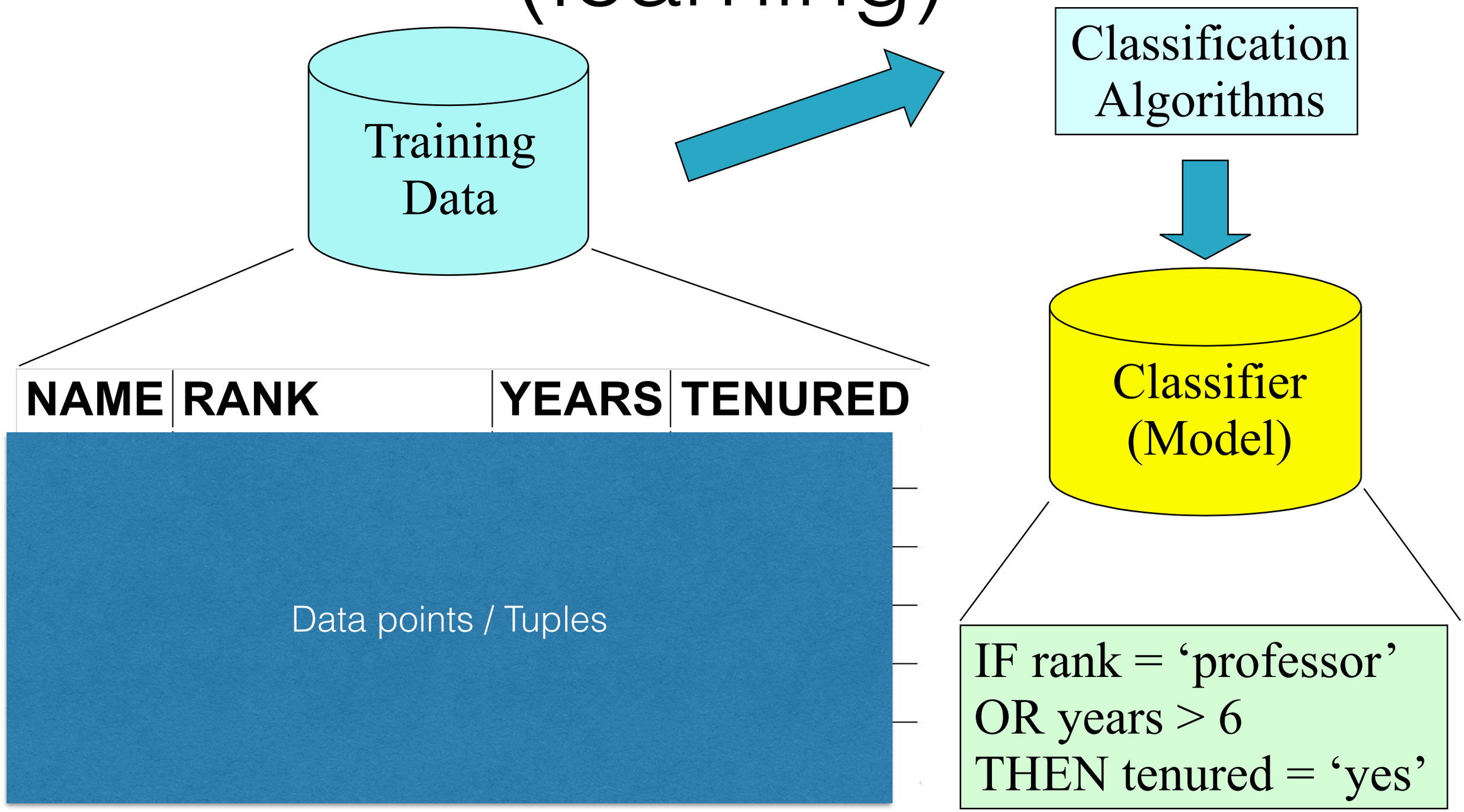
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Classification Algorithms



IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

Model construction (learning)



Model construction (learning)

Training Data

Classification Algorithms

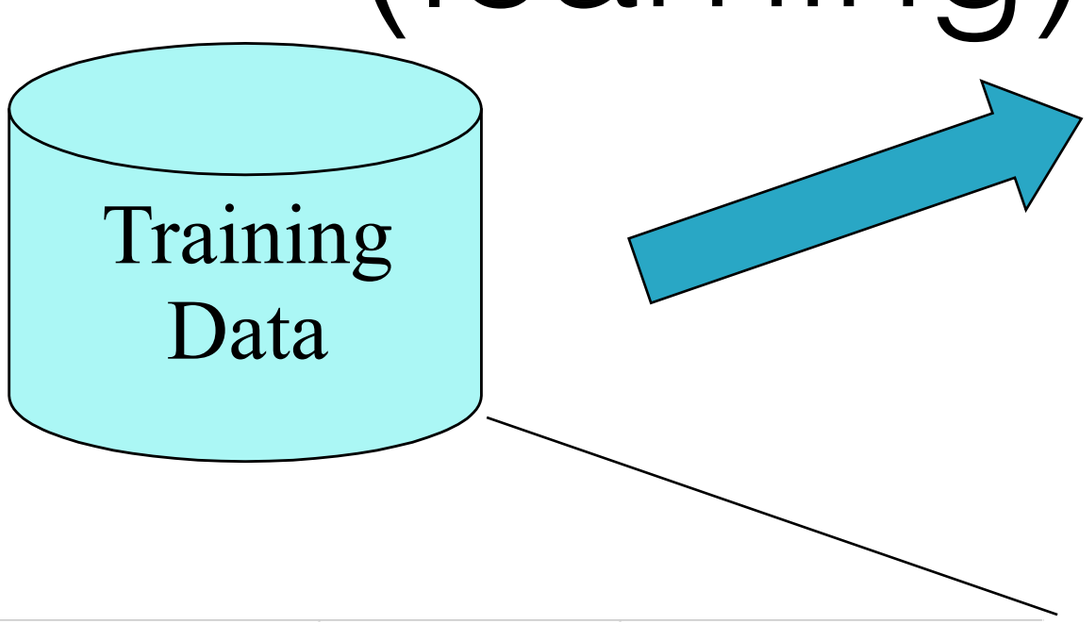
Classifier (Model)

NAME	RANK	YEARS	TENURED
			no
			yes
			yes
			yes
			no
			no

Features/ Attributes

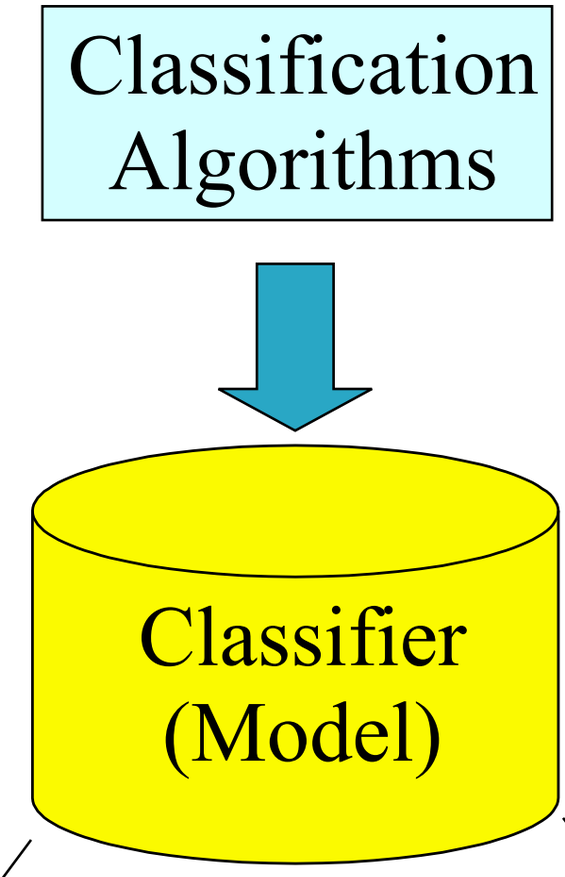
IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

Model construction (learning)



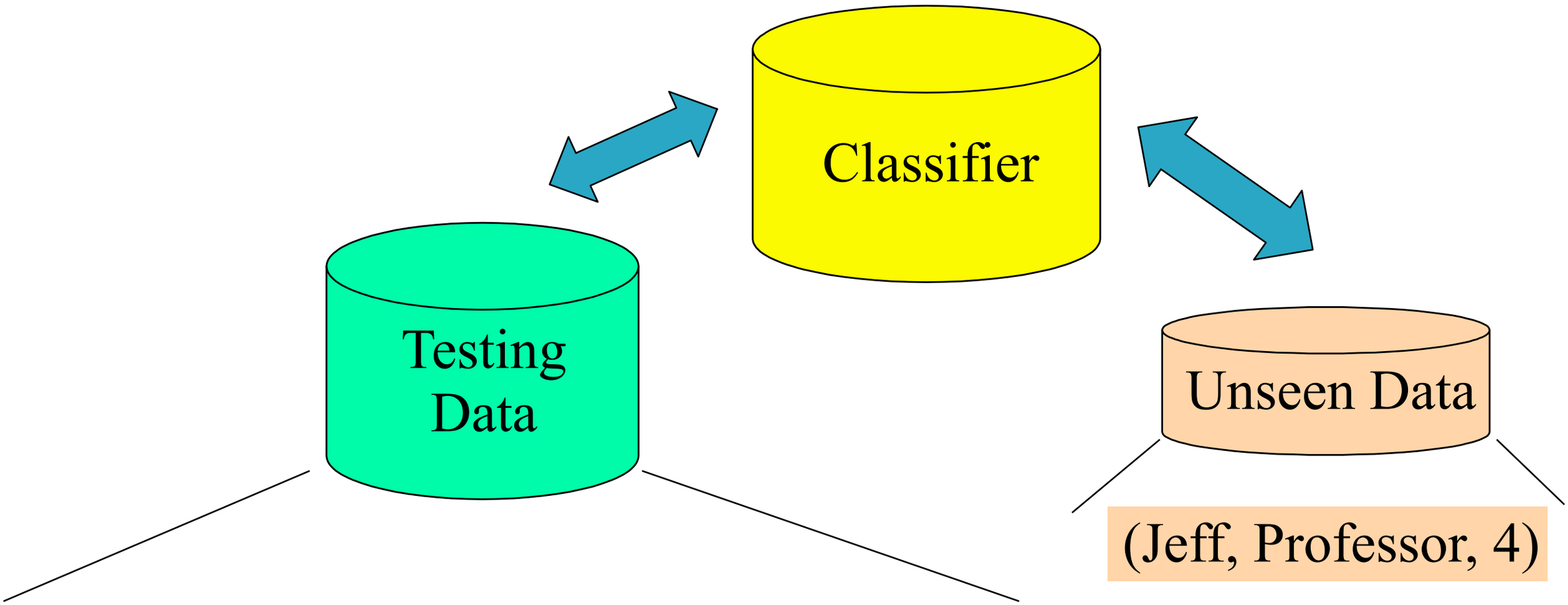
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	
Mary	Assistant Prof	7	
Bill	Professor	2	
Jim	Associate Prof	7	
Dave	Assistant Prof	6	
Anne	Associate Prof	3	

Labels / Targets



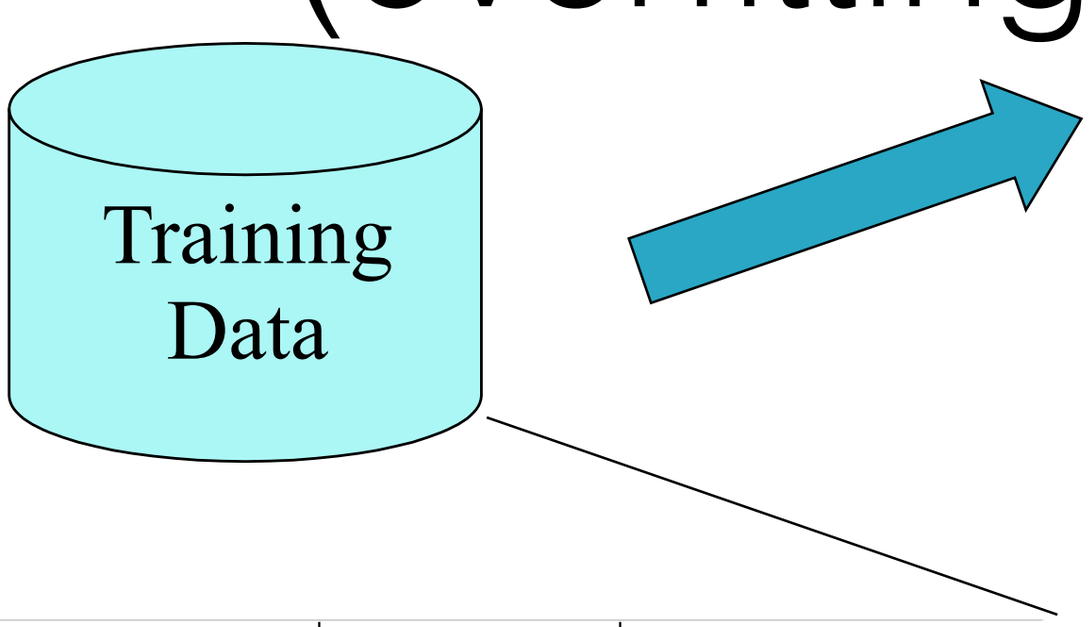
IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

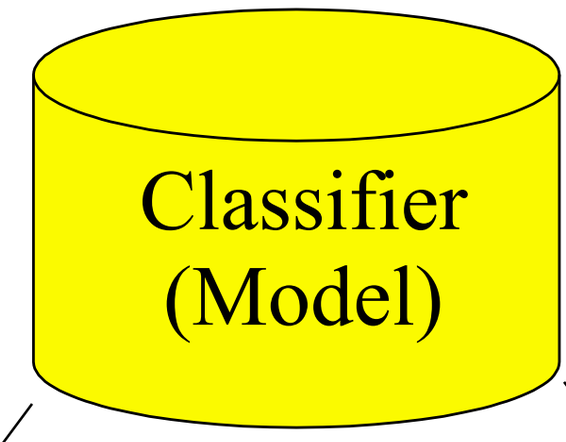
Using the model (inference)



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

Model construction (overfitting)





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

IF Name = “Mary”, “Bill” or”Jim” THEN Tenured = “yes”

Classification vs prediction

- Classification: binary or nominal labels
    - Examples: cat or dog, pregnant or not, from which country, which type of road sign
- Prediction: continuous labels
    - Examples: future stock price, life expectancy, distance to obstacle
    - Prediction is also called regression

Types of Datasets

- We train on the training set…
- …test while learning on the validation set…
- …and test after learning on the testing set (should be held-out until **after** learning. No peeking!)

What's desirable?

- Accuracy
  classifier accuracy: predicting class label
  regression 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: humans can understand the model

- Generality: model should work on novel examples

- 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

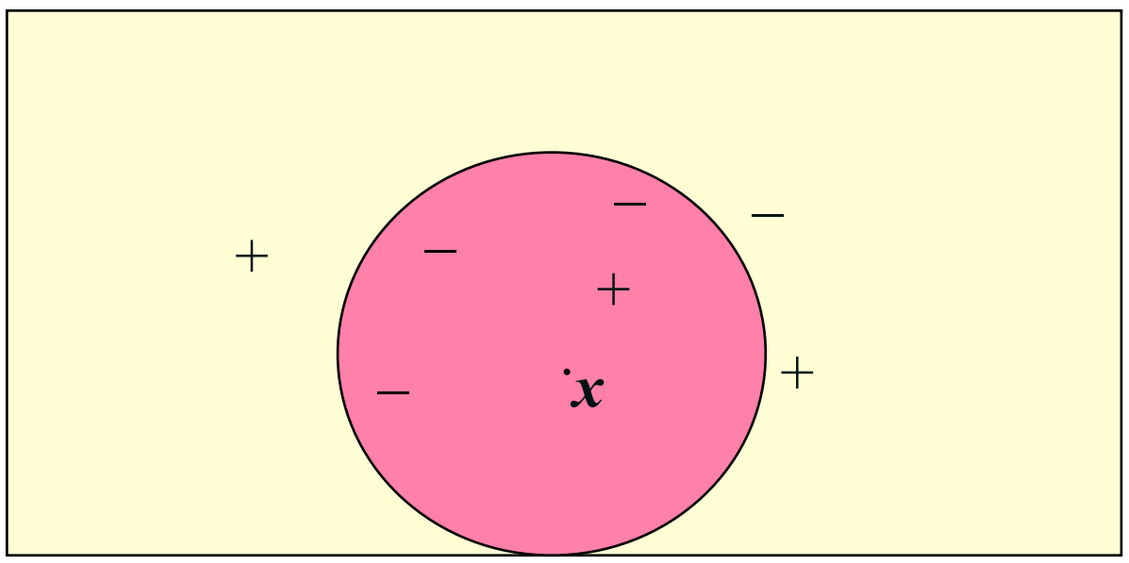
- Eager learning: Given a training set, constructs a classification model (smaller than the data) before receiving new data to classify

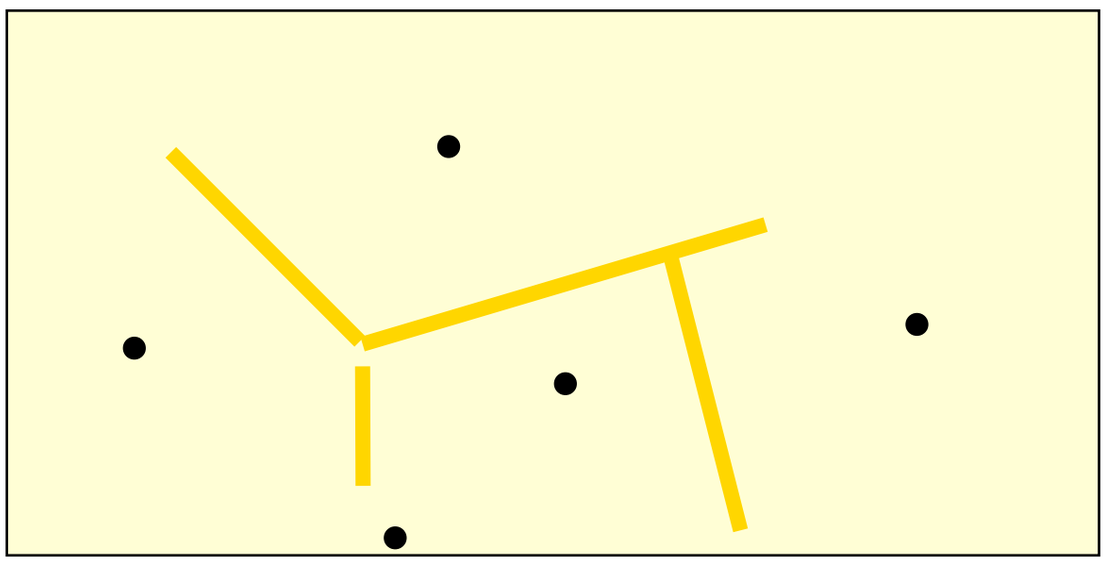
- Lazy: less time in training but more time in predicting

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 neighbor is defined in terms of Euclidean distance, dist(X_1, X_2)
- Target function could be discrete- or real-valued
- Use the mode (most common value) for discrete targets, mean/median for real-valued targets
- Voronoi 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

$$d(i,j)=\sqrt{(|x_{i1}-x_{j1}|^2+|x_{i2}-x_{j2}|^2+...+|x_{ip}-x_{jp}|^2)}$$

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

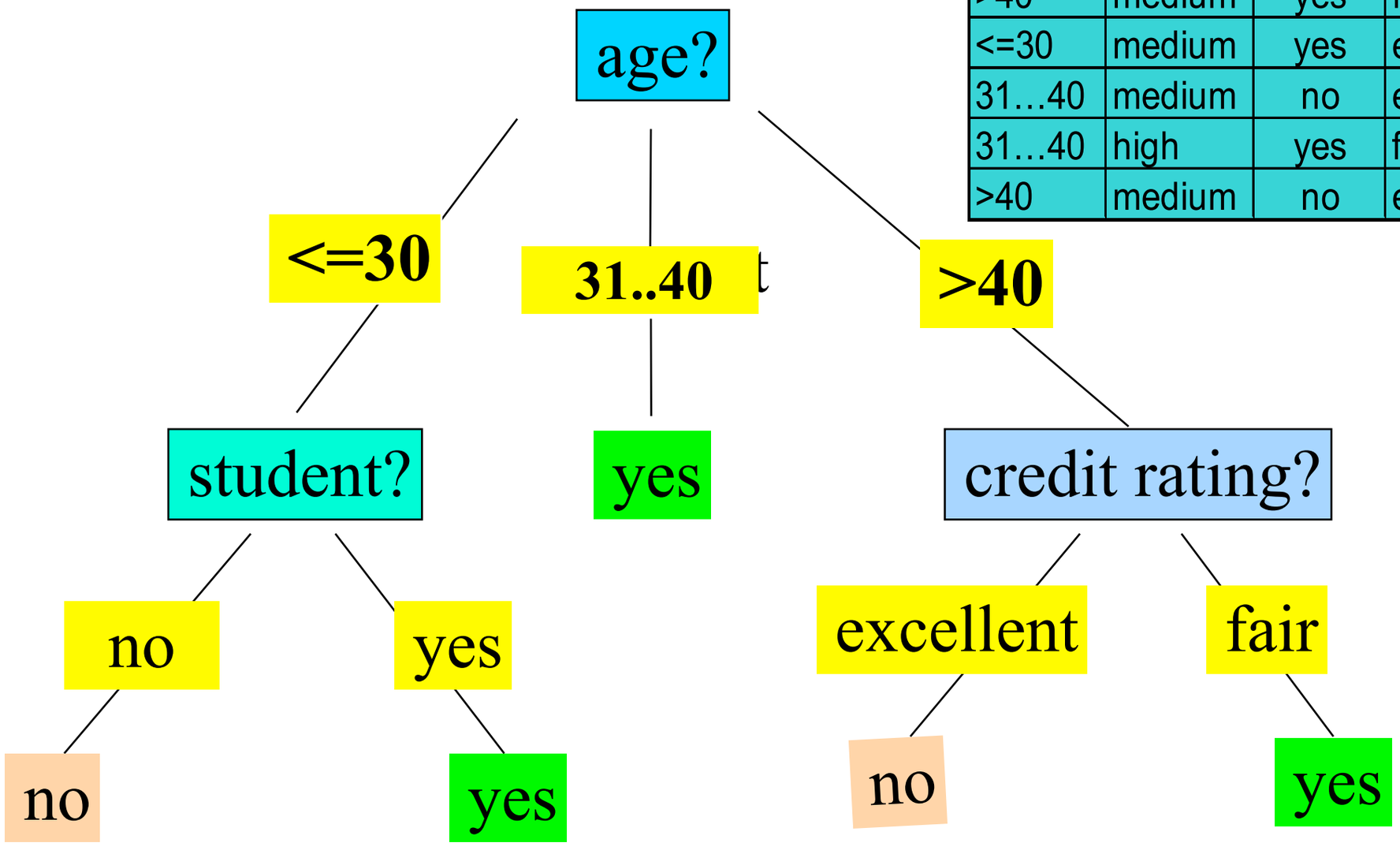
Decision trees

- A popular representation for classifiers
- Human-readable
- Can be learned (induced) efficiently using algorithms based on information theory
  - An eager learning method
- Often yields high-accuracy classifiers

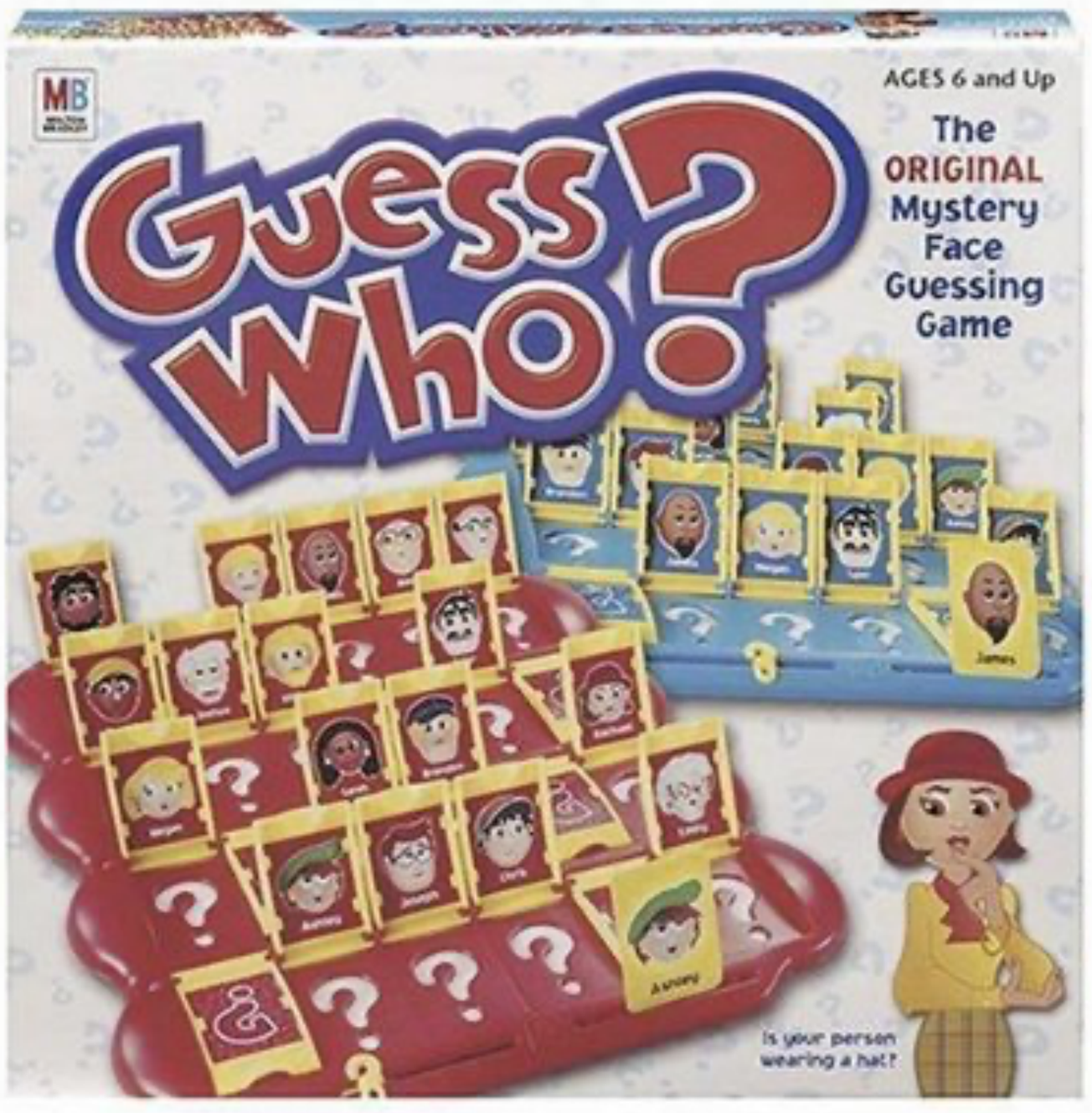
An example

Classify: buys_computer

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no



Analogy: “Guess Who?” Game



ID3 Algorithm for Decision Tree Induction

- Tree is constructed in a top-down recursive divide-and-conquer manner

- At start, all the training examples are at the root

- Attributes are categorical (if continuous-valued, they are discretized in advance)

- Examples are partitioned recursively based on selected attributes

- Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

ID3 Algorithm for Decision Tree Induction

- Stop partitioning when...
  - All samples for a given node belong to the same class, or…
  - There are no remaining attributes for further partitioning – (vote on the leaf) or…
  - There are no samples left

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Assumptions:
  - Our features are $A, B, C, \ldots$ etc, with $v_A, v_B, v_C, \ldots$ possible values in each feature's domain
  - Our table is Y with m possible values $y_1, \ldots, y_m$, indexed by i
- Let p_i be the probability that an arbitrary tuple in D belongs to class y_i, $P(Y = y_i)$ estimated by $|D_{Y=y_i}|/|D||$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i * log_2(p_i)$$ over target classes i

- Can be interpreted as the number of bits of information needed for classification

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Information needed (after using A to split D into v_a partitions $D_{A=j}$) to classify D

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_{A=j}|}{|D|} Info(D_{A=j})$$

-

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Information needed (after using A to split D into v_a partitions $D_{A=j}$) to classify D

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_{A=j}|}{|D|} Info(D_{A=j})$$

- In other words, it's the sum of information of each subset of D filtered by $A = j$,

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Information needed (after using A to split D into v_a partitions $D_{A=j}$) to classify D

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_{A=j}|}{|D|} \times Info(D_{A=j})$$

- In other words, it's the sum of information of each subset of D restricted by $A = j$, weighted by the relative size of each subset

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Information needed (after using A to split D into v_a partitions $D_{A=j}$) to classify D

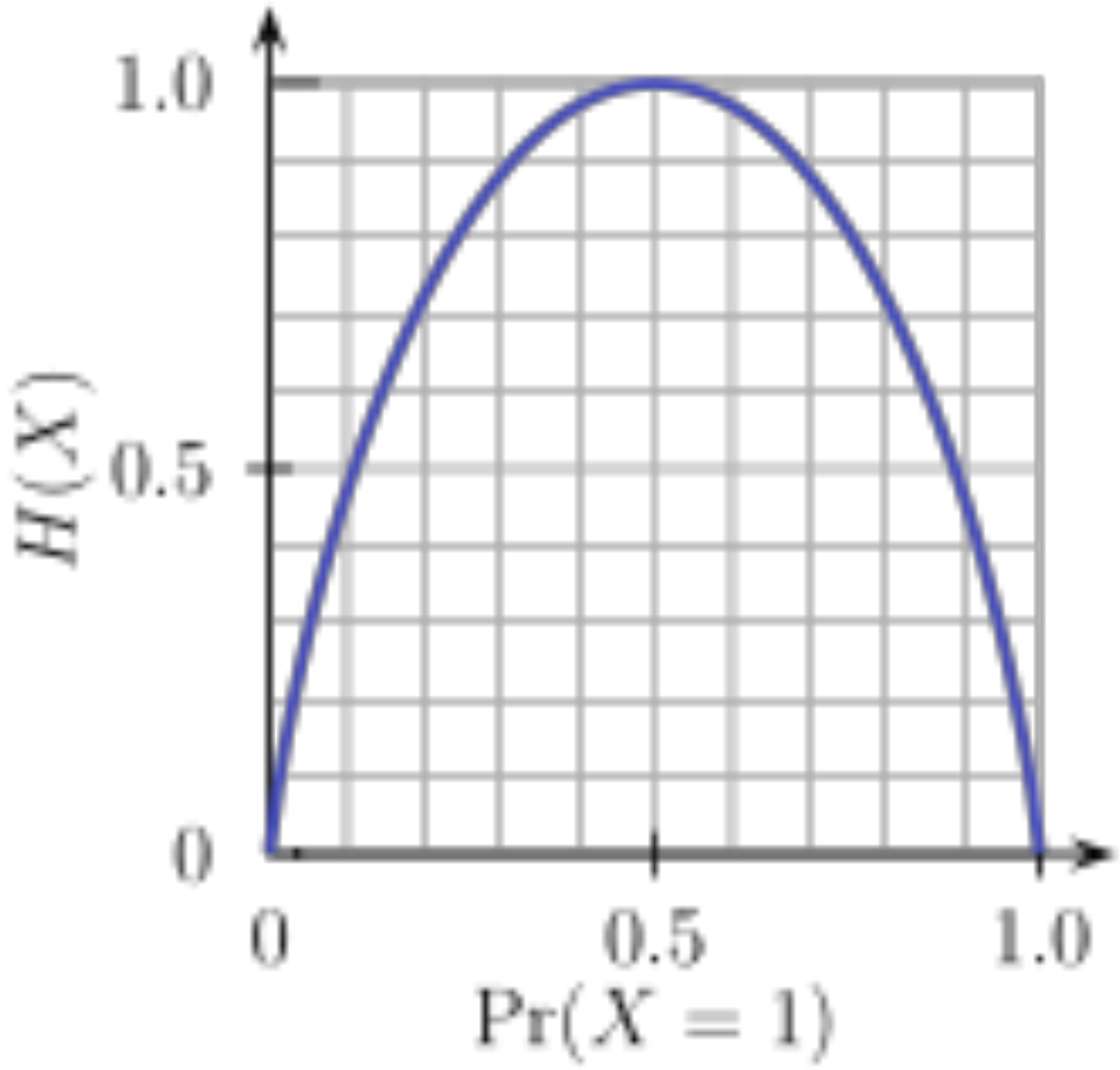
$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_{A=j}|}{|D|} Info(D_{A=j})$$

- Information gained by branching on attribute A

$$Gain(A, D) = Info(D) - Info_A(D)$$

- We want to select the attribute with highest gain (biggest *decrease* in entropy)

Binary entropy function



Attribute Selection: Information Gain

Class P: buys_computer = “yes”

Class N: buys_computer = “no”

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

means “age <=30” has 5 out of 14 samples, with 2 yes’es and 3 no’s. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

- Input: A data set, S
  Output: A decision tree

    - If all the instances have the same label for the target attribute then return a decision tree that is simply this label

- Else

    - Compute Gain values (see above) for all attributes and select an attribute with the highest value and create a node for that attribute.

    - Make a branch from this node for every value of the attribute

    - Follow each branch by partitioning the dataset to be only instances whereby the value of the branch is present and then go back to 1.

function DECISION-TREE-LEARNING($examples$, $attributes$, $parent\_examples$) **returns** tree

if $examples$ is empty **then return** PLURALITY-VALUE($parent\_examples$)
else if all $examples$ have the same classification **then return** the classification
else if $attributes$ is empty **then return** PLURALITY-VALUE($examples$)
else
$A \leftarrow \text{argmax}_{a \in attributes}$ IMPORTANCE(a, $examples$)
$tree \leftarrow$ a new decision tree with root test A
for each value v_k of A **do**
$exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
$subtree \leftarrow$ DECISION-TREE-LEARNING(exs, $attributes - A$, $examples$)
add a branch to $tree$ with label $(A = v_k)$ and subtree $subtree$
return $tree$

Extending decision trees

- **Regression Trees**
  - If prediction target is real-valued, each leaf node returns average value of datapoints
  - Select features (and thresholds, if features are real-valued) as to minimize error (typically mean squared error) of the node

Extending decision trees

- **Decision Forests (e.g. Random Forests)**
  - Train an ensemble of diverse decision trees
  - Take the average or mode of each tree's prediction
  - Diversity can be achieved by training each tree on a random subset of data points and features
  - Each tree can overfit, but the ensemble tends to minimize overfitting in the joint prediction

Extending decision trees

- **Boosting trees**
  - Another ensemble method
  - Each successive tree predicts (and corrects) the error (loss) of the previous tree
    - **Adaboost**: Assign weights to datapoints focusing on points with large loss
    - **XGBoost:** Directly estimate the gradient of the loss

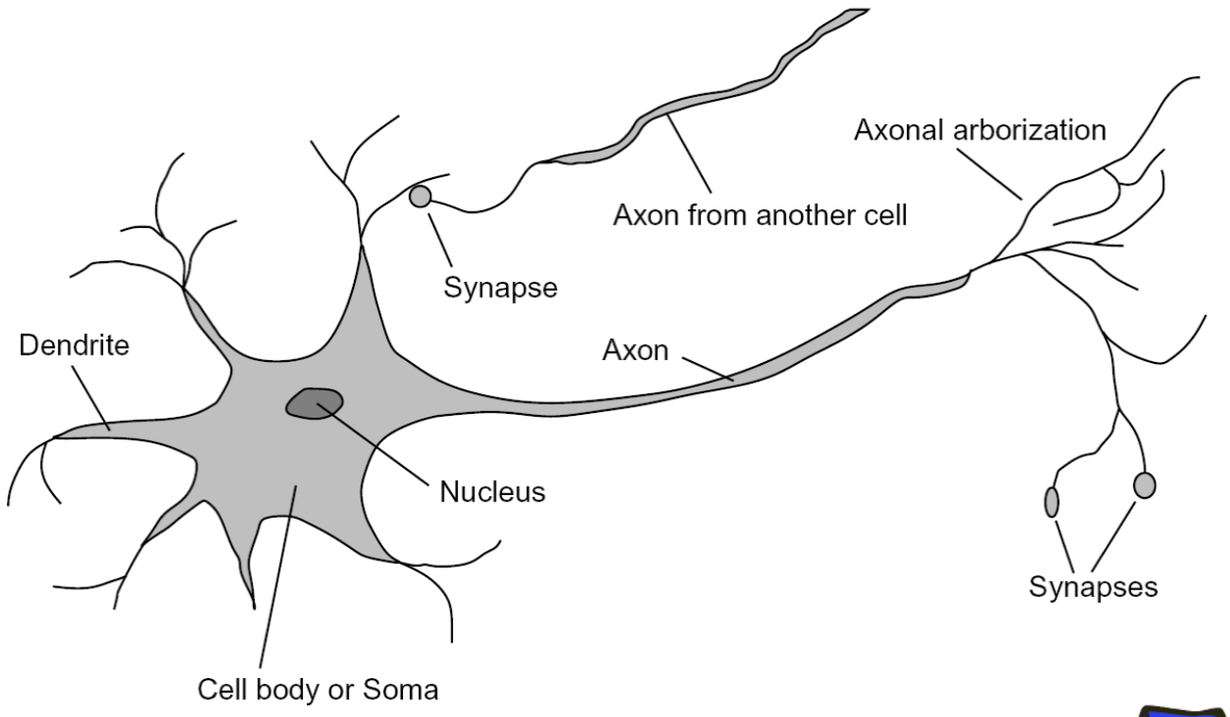
Perceptrons

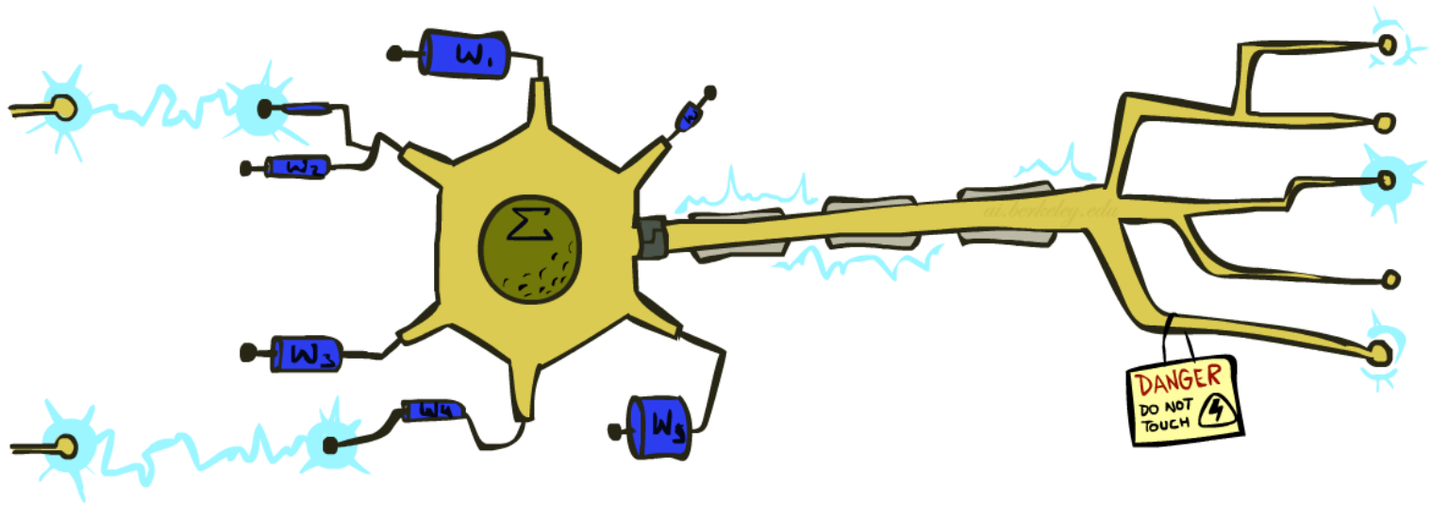
- Early attempt at “neural networks” - copying neurophysiology to produce a learning machine

- 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



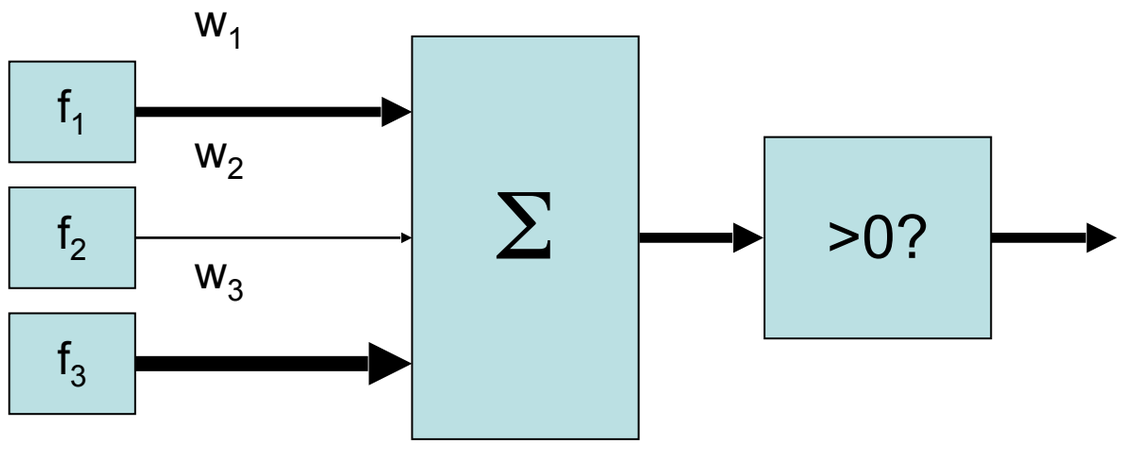


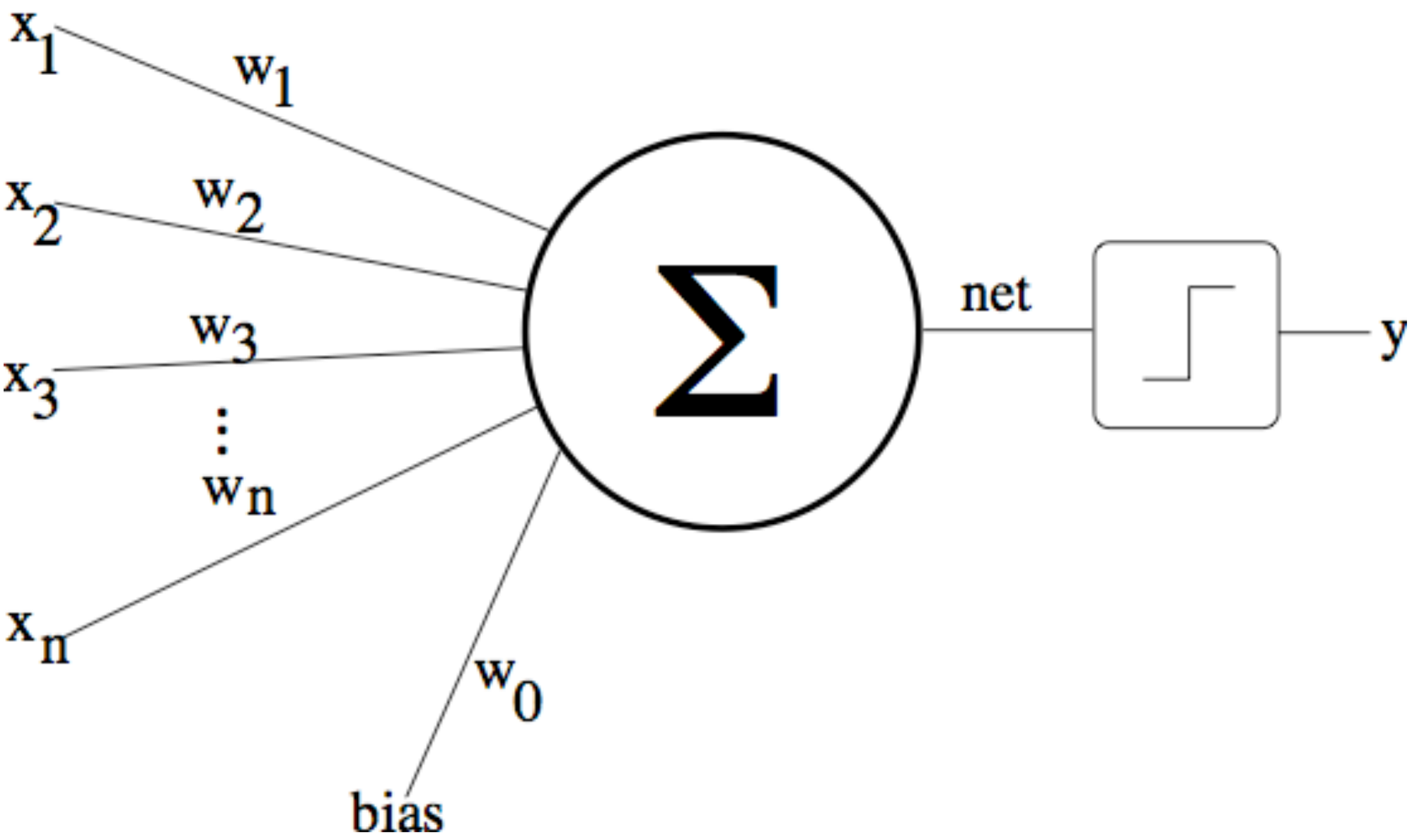
Perceptrons are linear classifiers

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

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

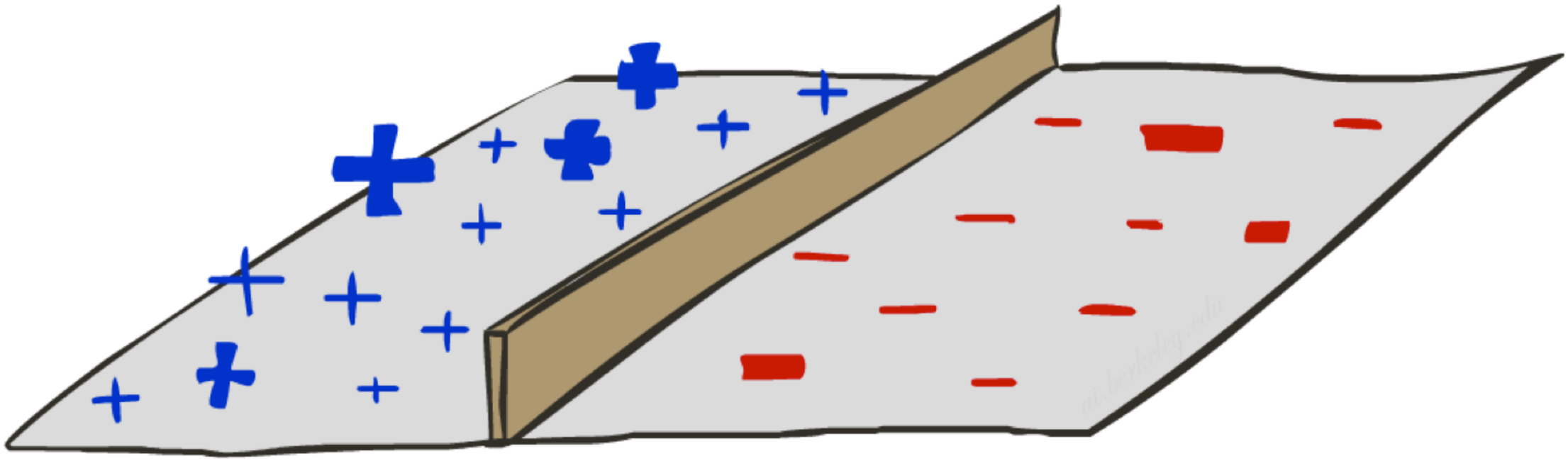
- If the activation is:
  - Positive, output +1
  - Negative, output 0





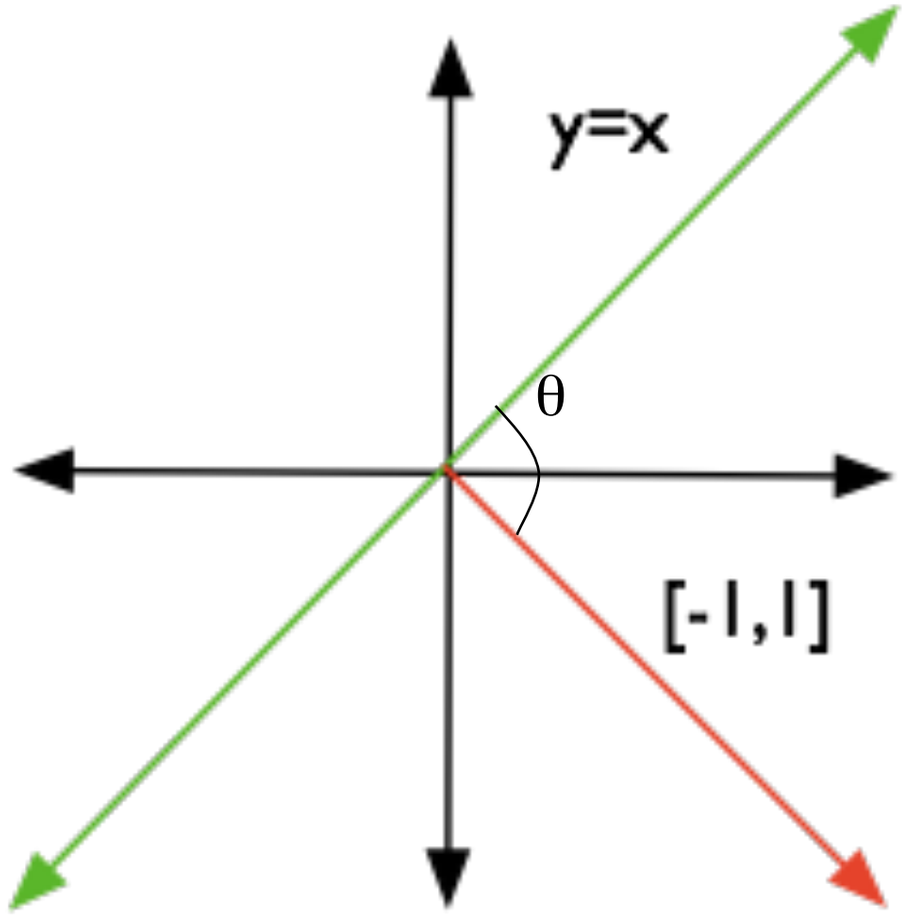
$$f(x) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0 & \text{else} \end{cases}$$

Decision surface



Decision surface

$$y = x$$
$$0 = x - y$$
$$0 = [1, -1] \begin{bmatrix} x \\ y \end{bmatrix}$$

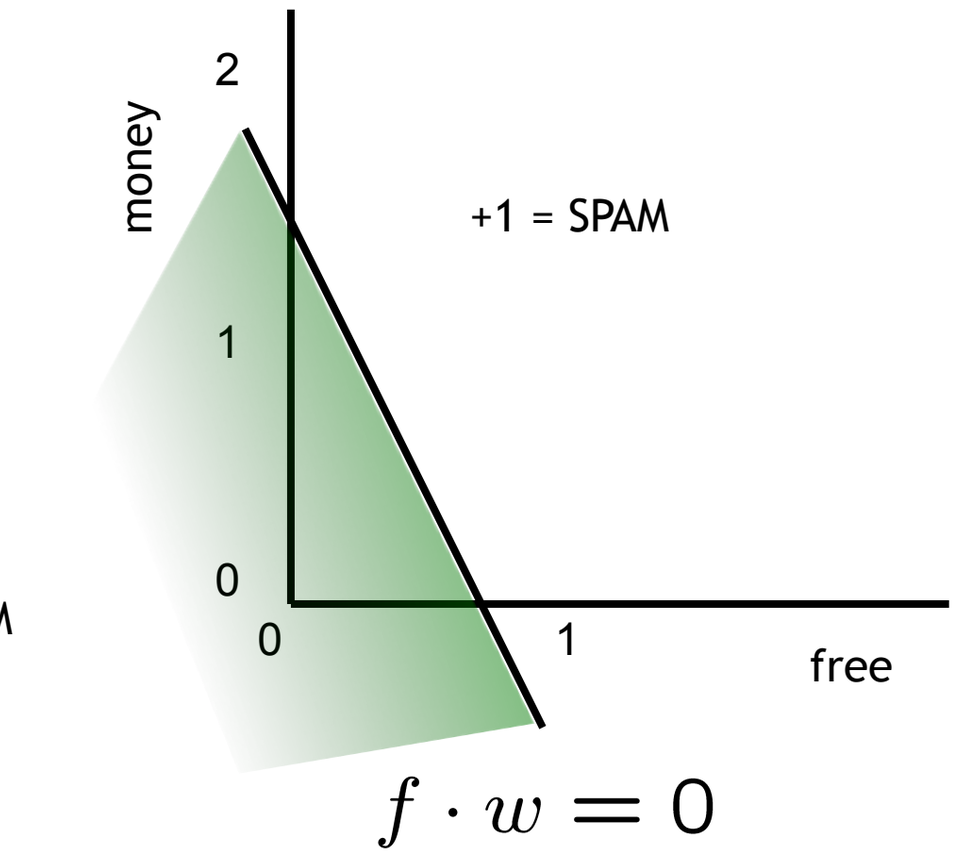


In general a **hyperplane** is defined by

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

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

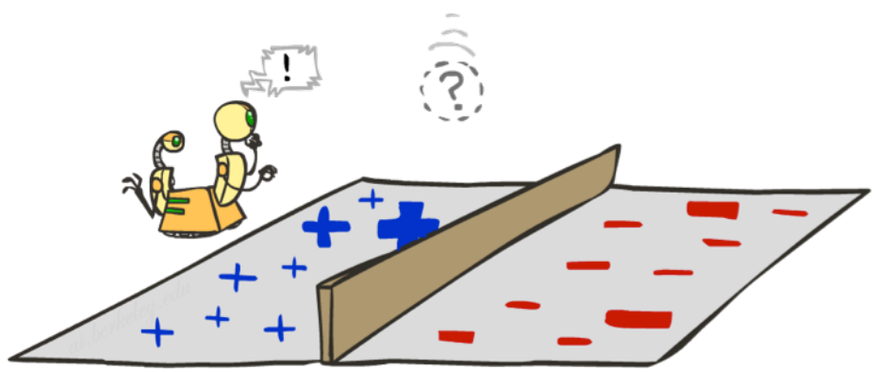
Algorithm

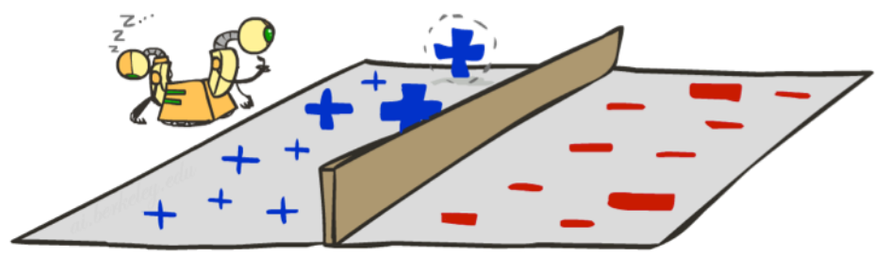
- Start with random weights.
  For each training instance:

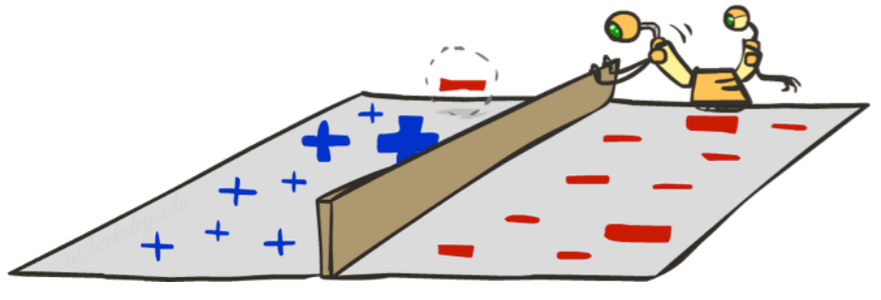
- Classify with current weights

- If correct (i.e., y=y*), no change!

- If wrong: adjust the weight vector







Algorithm - Linear Classifier

Input : list of n training examples $(x_0, y_0) \ldots (x_n, y_n)$
where $\forall i : y_i \in \{+1, -1\}$, learning rate η

Output : classifying hyperplane w, bias b

Algorithm :

Randomly initialize w and b;
While makes errors on training set **do**
  for (x_i, y_i) **do**
    $h = \text{sign}(w \cdot x_i + b)$ *// output of the model*
    $w = w + \eta*(y_i - h)*x_i$
    $b = b + \eta*(y_i - h)$
  end
end

This works for the linear classifier. For the lab, you'll need the more general formula (next slide)

x and w are vectors;
i is the instance index

Algorithm - Linear Classifier

Input : list of n training examples $(x_0, y_0) \ldots . (x_n, y_n)$
where $\forall i : y_i \in \{+1, -1\}$, learning rate η

Output : classifying hyperplane w, bias b

Algorithm :

Randomly initialize w and b;

While makes errors on training set **do**

for (x_i, y_i) **do**

$$h = w \cdot x_i + b$$

$$L = \text{loss}(y_i, h)$$

$$w = w - \eta * \frac{\partial L}{\partial w}, \quad b = b - \eta * \frac{\partial L}{\partial b}$$

end

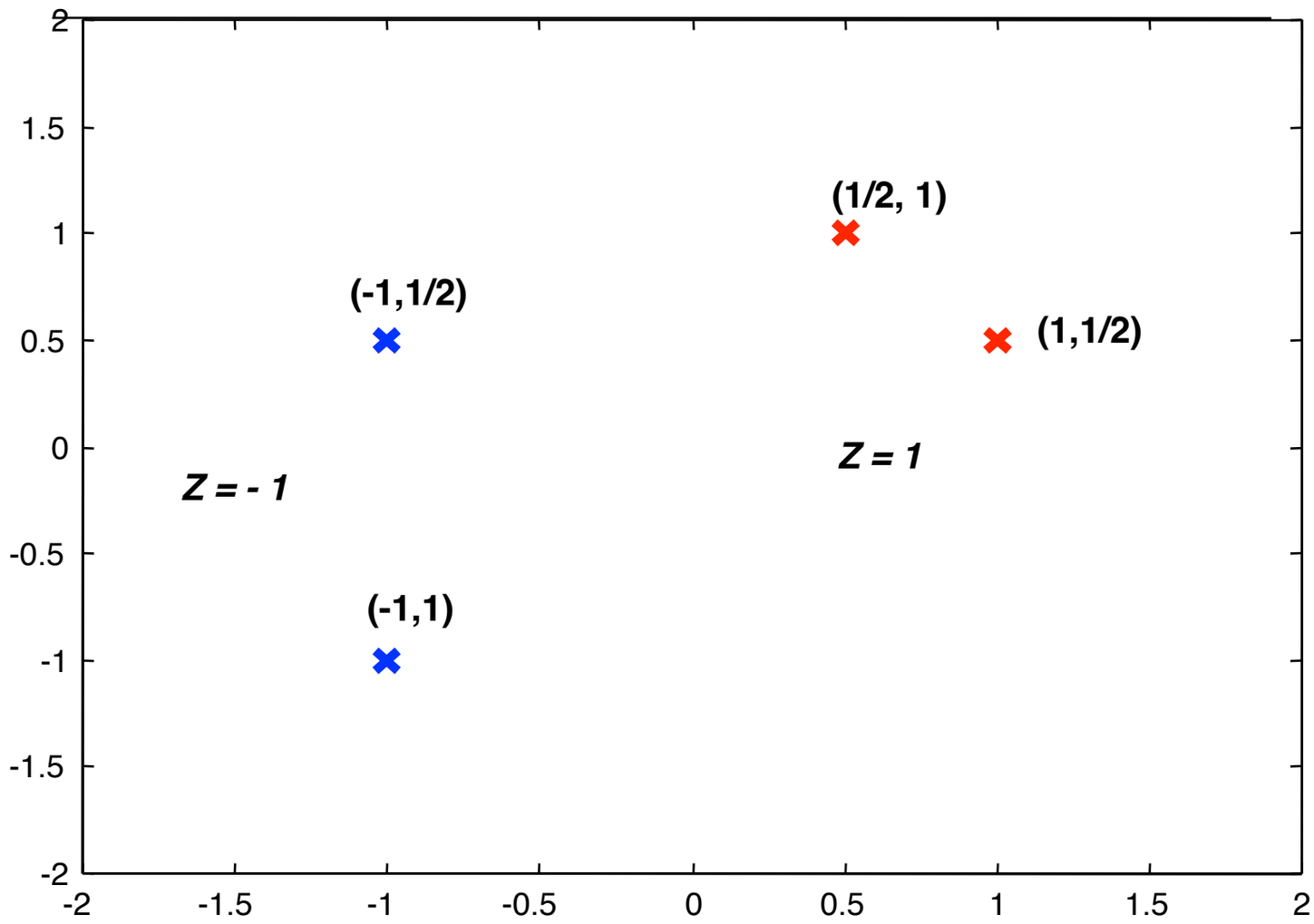
end

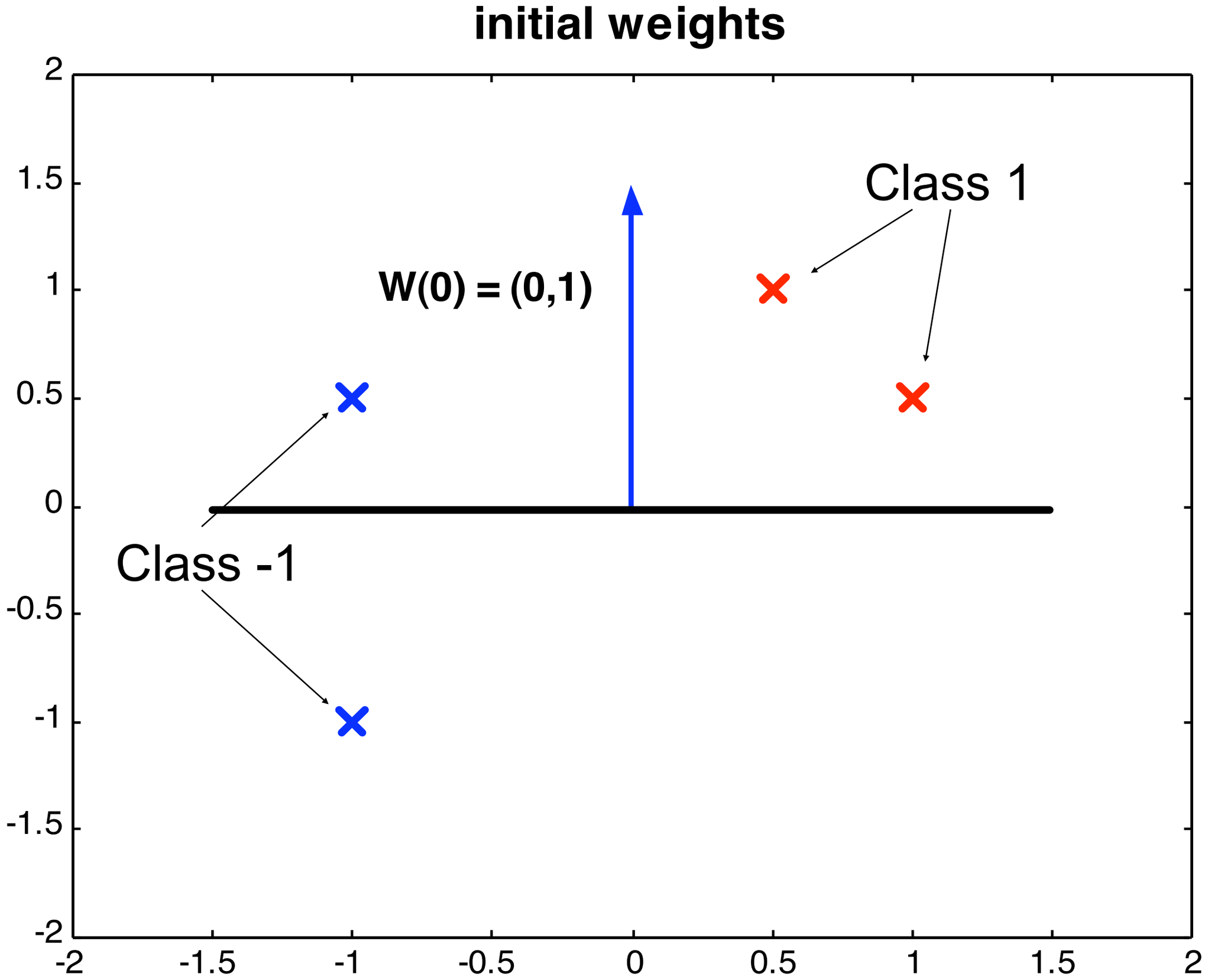
Note the NEGATIVE sign on the gradient term!

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.

Let's use η = 1/6, and let's ignore the bias for now

X = (-1,1/2)

Y = -1

w = (0,1)

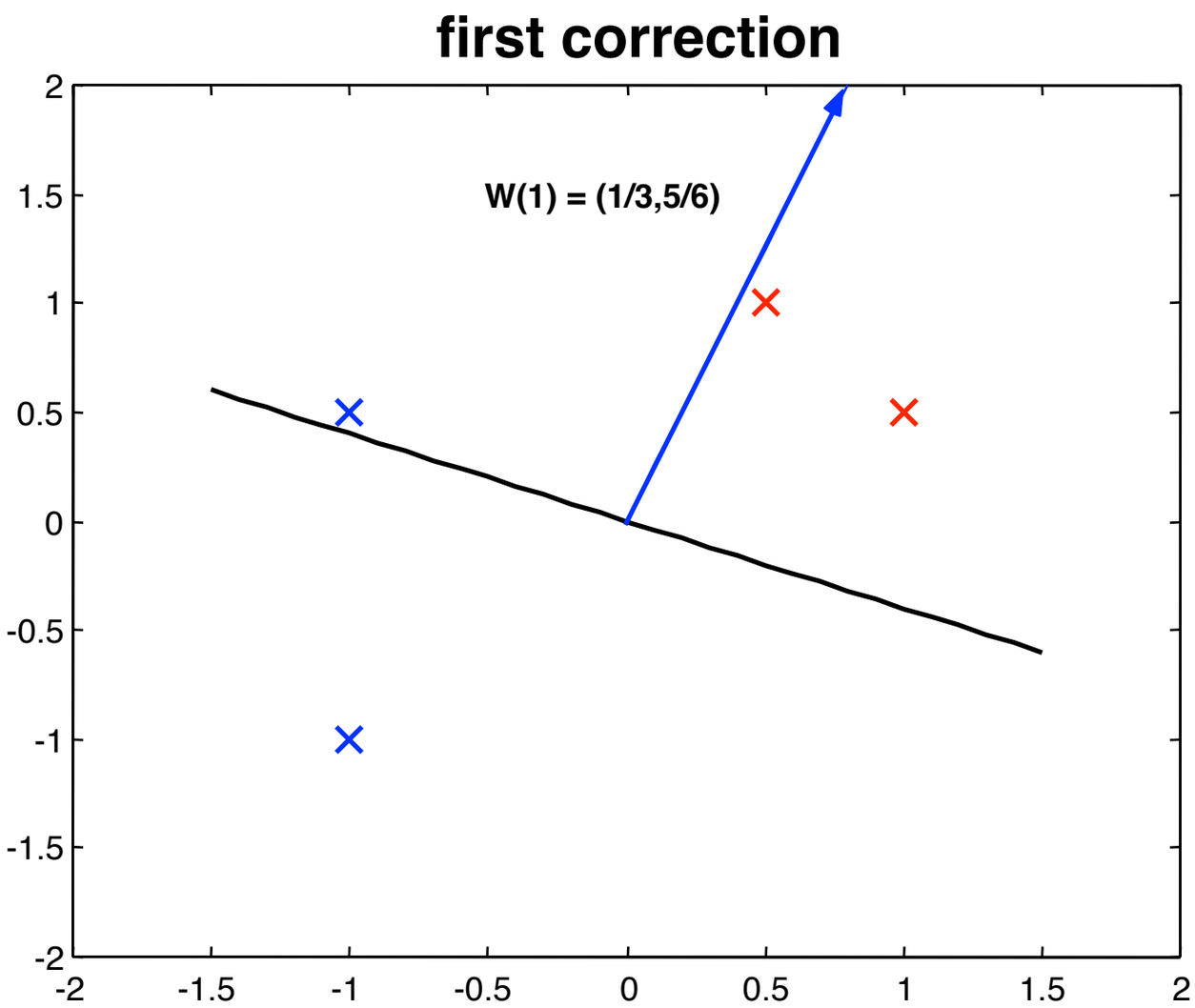
h = sign(0*(-1) + 1*(1/2)) = sign(1/2) = 1

w = w + η*(*y* -h)*x

= (0,1) + 1/6*(-1-1)*(-1,1/2)

= (0,1) - 1/3 (-1,1/2)

= (1/3, 5/6)


first correction
W(1) = (1/3,5/6)
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, Ctd

Upper left point is still wrongly classified

X = (-1,1/2)

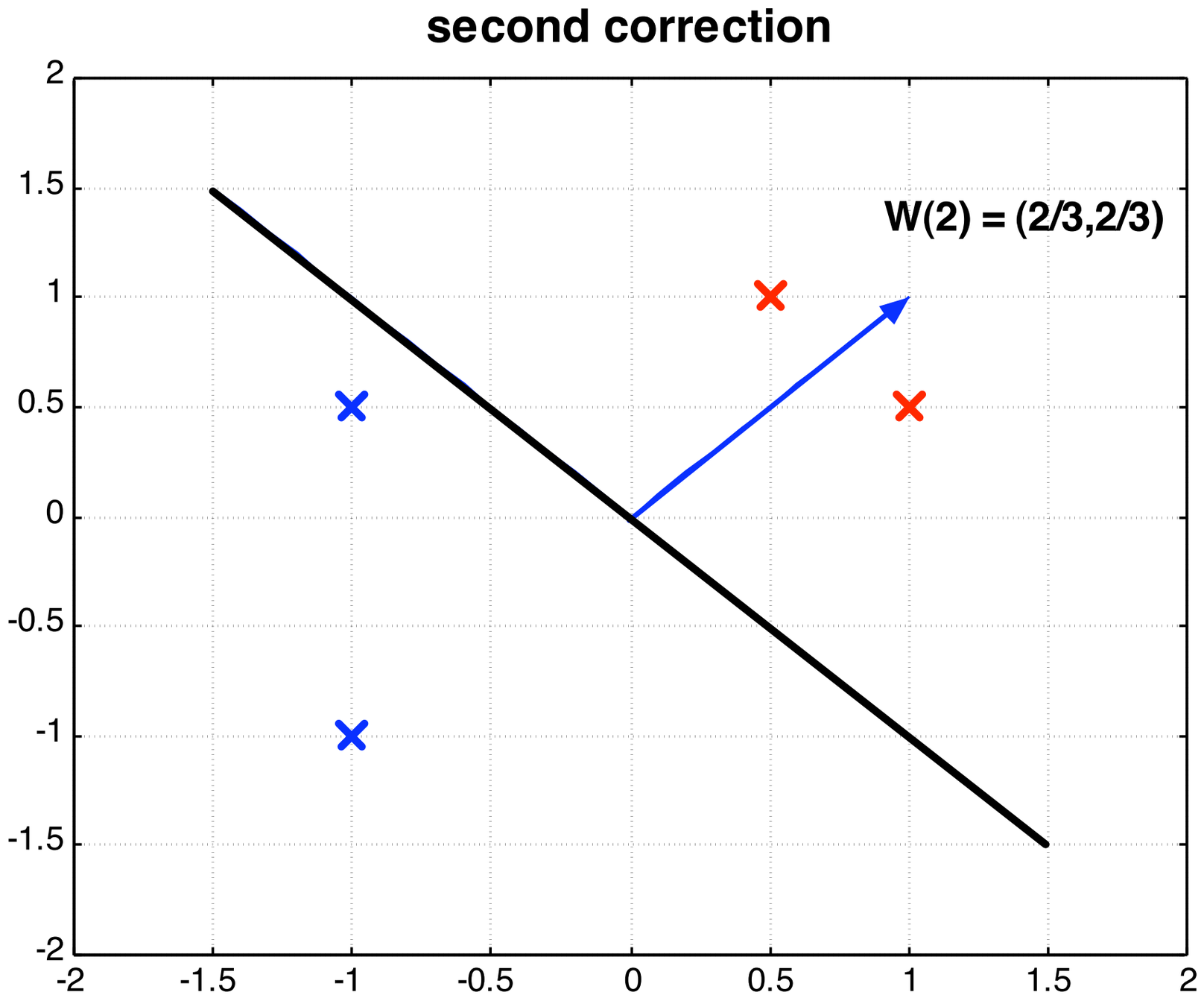
Y = -1, h = 1

w = w + η*(y -h)*x

= (1/3,5/6) + 1/6*(-2)*(-1,1/2)

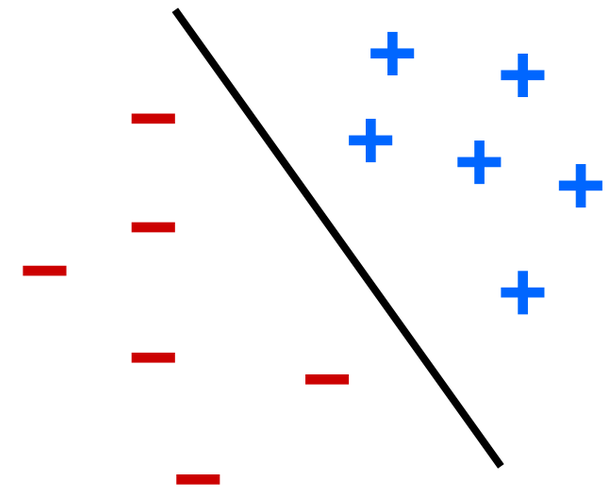
= (1/3,5/6) - 1/3 (-1,1/2)

= (2/3, 2/3)

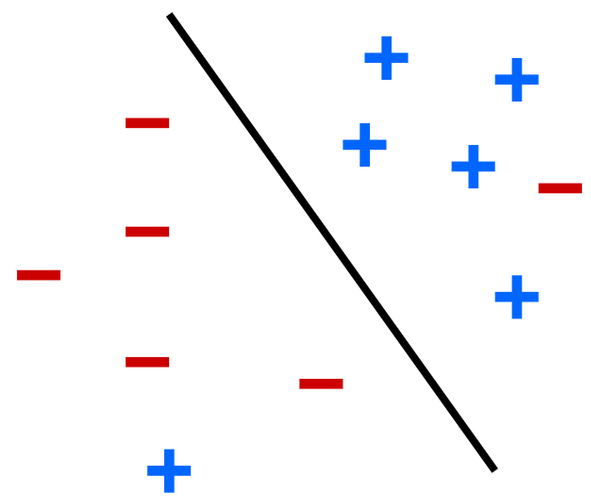

second correction
W(2) = (2/3,2/3)
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Properties of Perceptrons

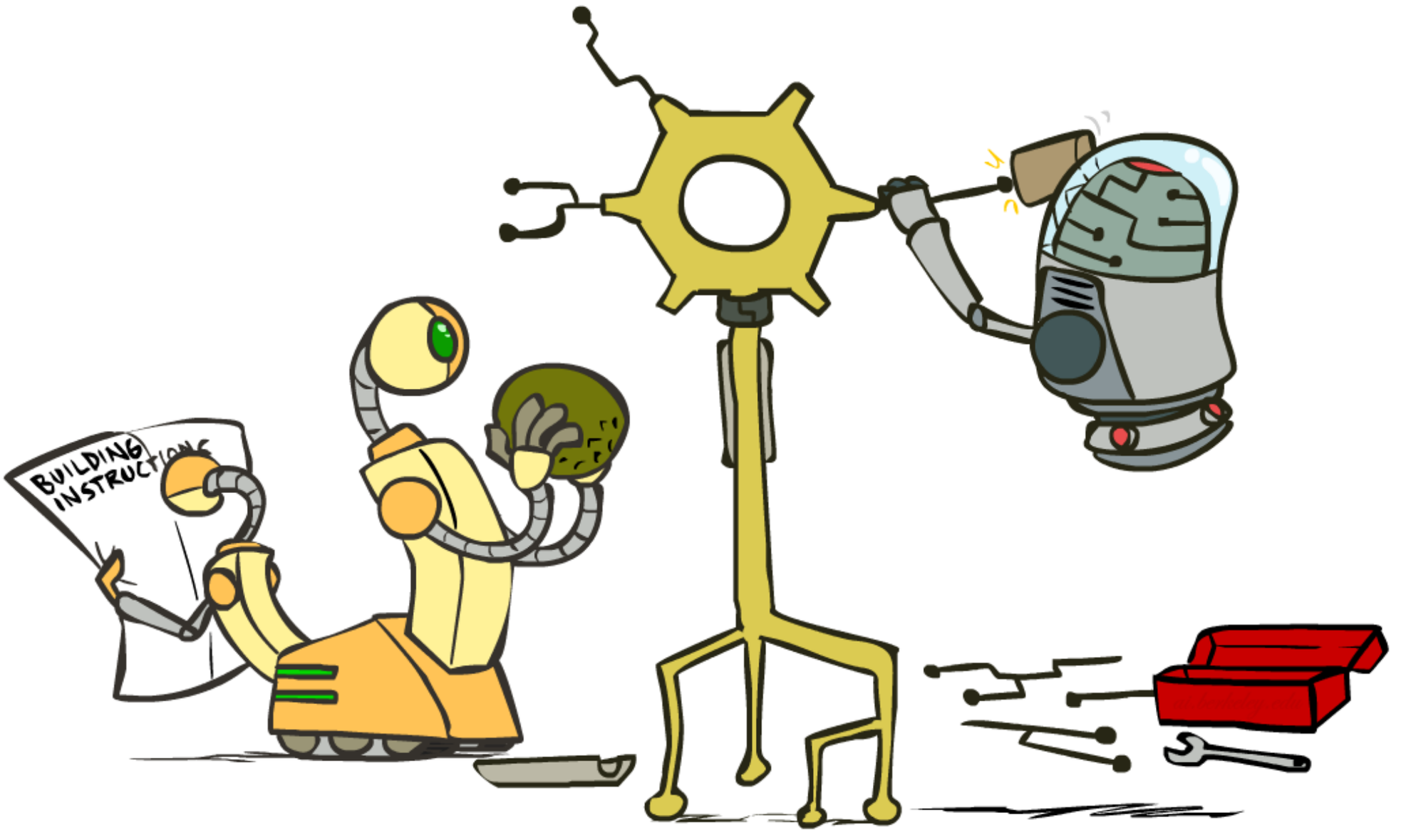
- Separability: true if some parameters get the training set perfectly correct



- Convergence: if the training is separable, perceptron will eventually converge (binary case)

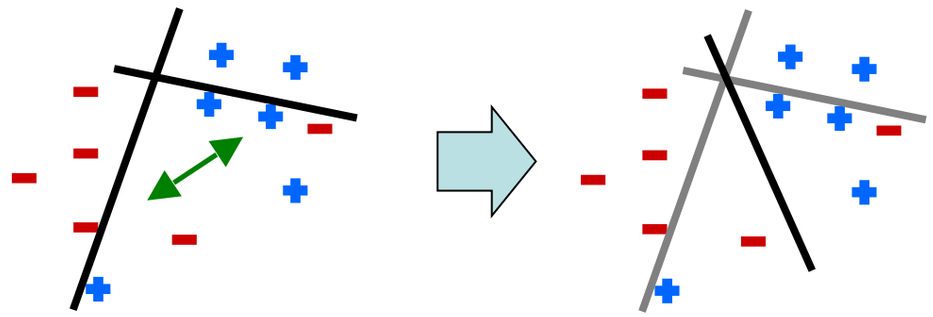


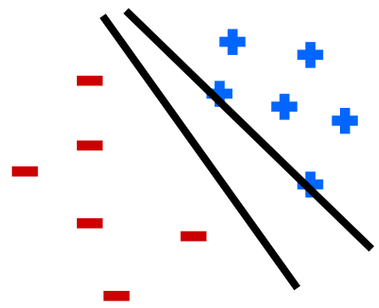
Improving the perceptron

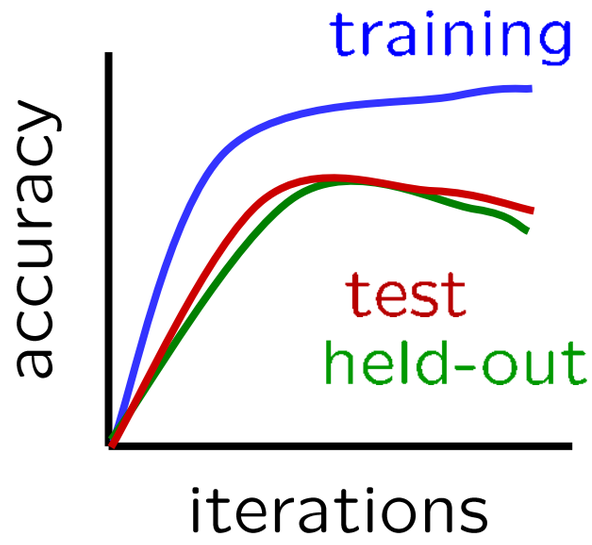


Perceptron problems

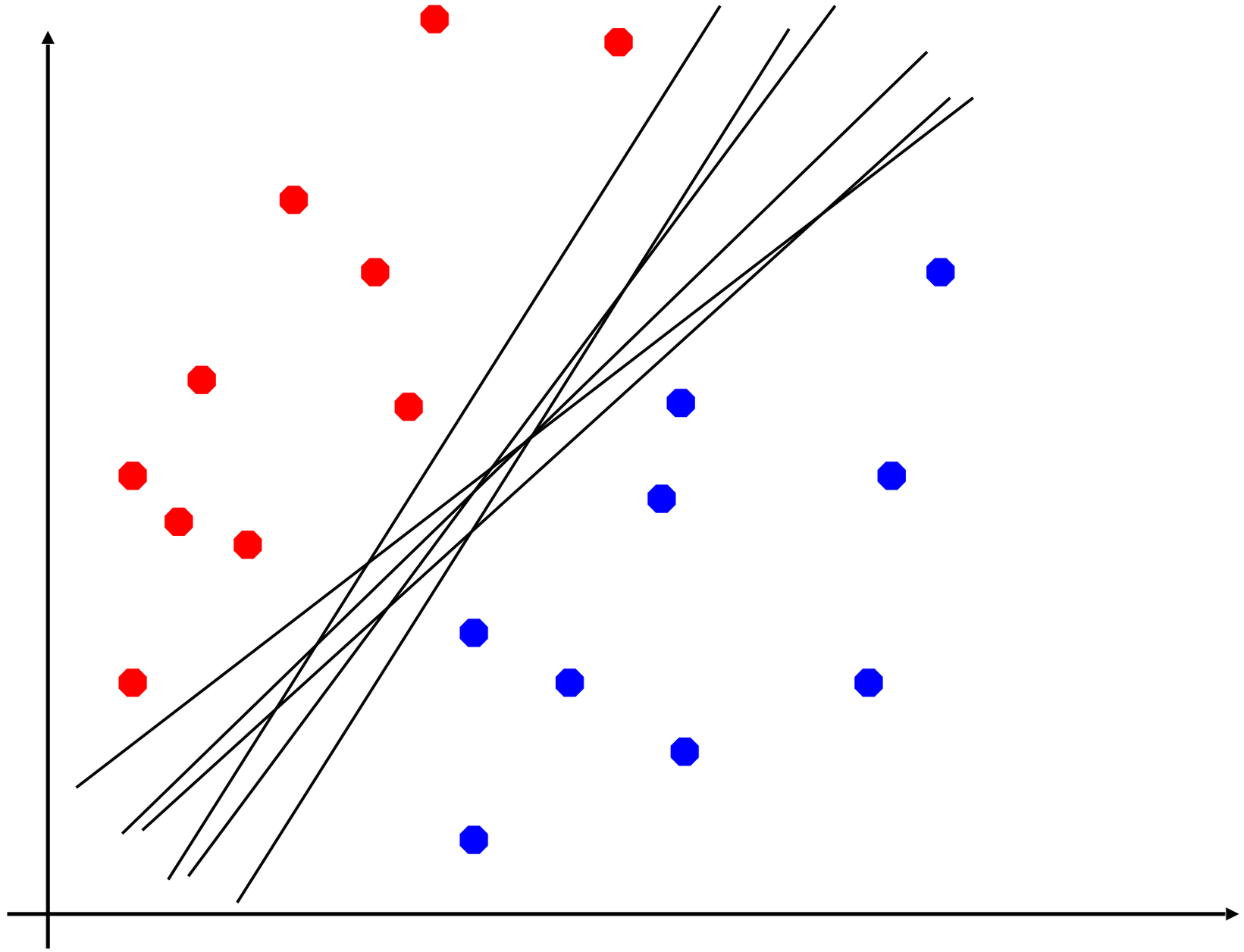
- Noise: if the data isn't separable, weights might thrash
  - Averaging weight vectors over time can help (averaged perceptron)
- Mediocre generalization: finds a "barely" separating solution
- Overtraining: test / held-out accuracy usually rises, then falls
  - Overtraining is a kind of overfitting





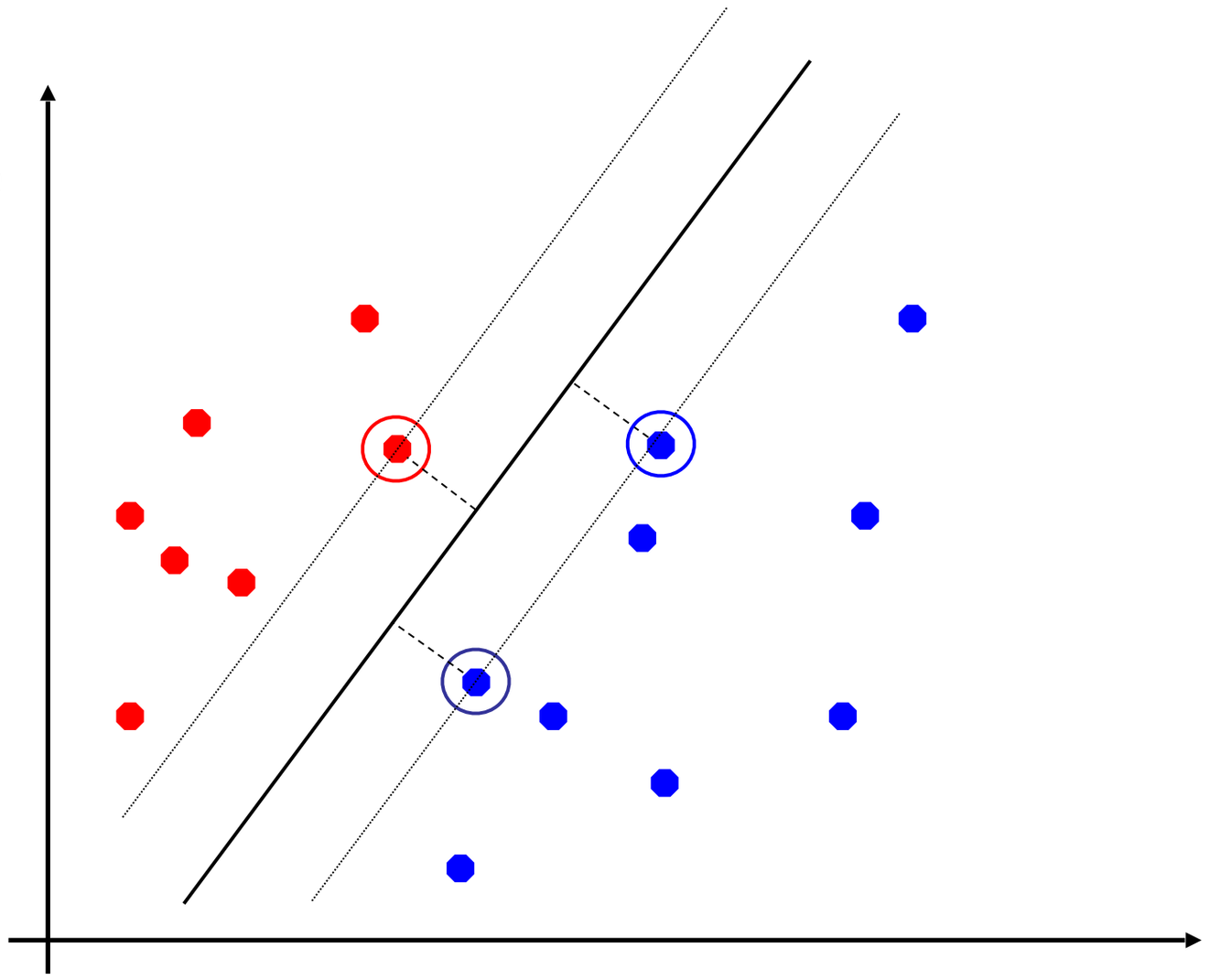


Which of these separators is optimal?

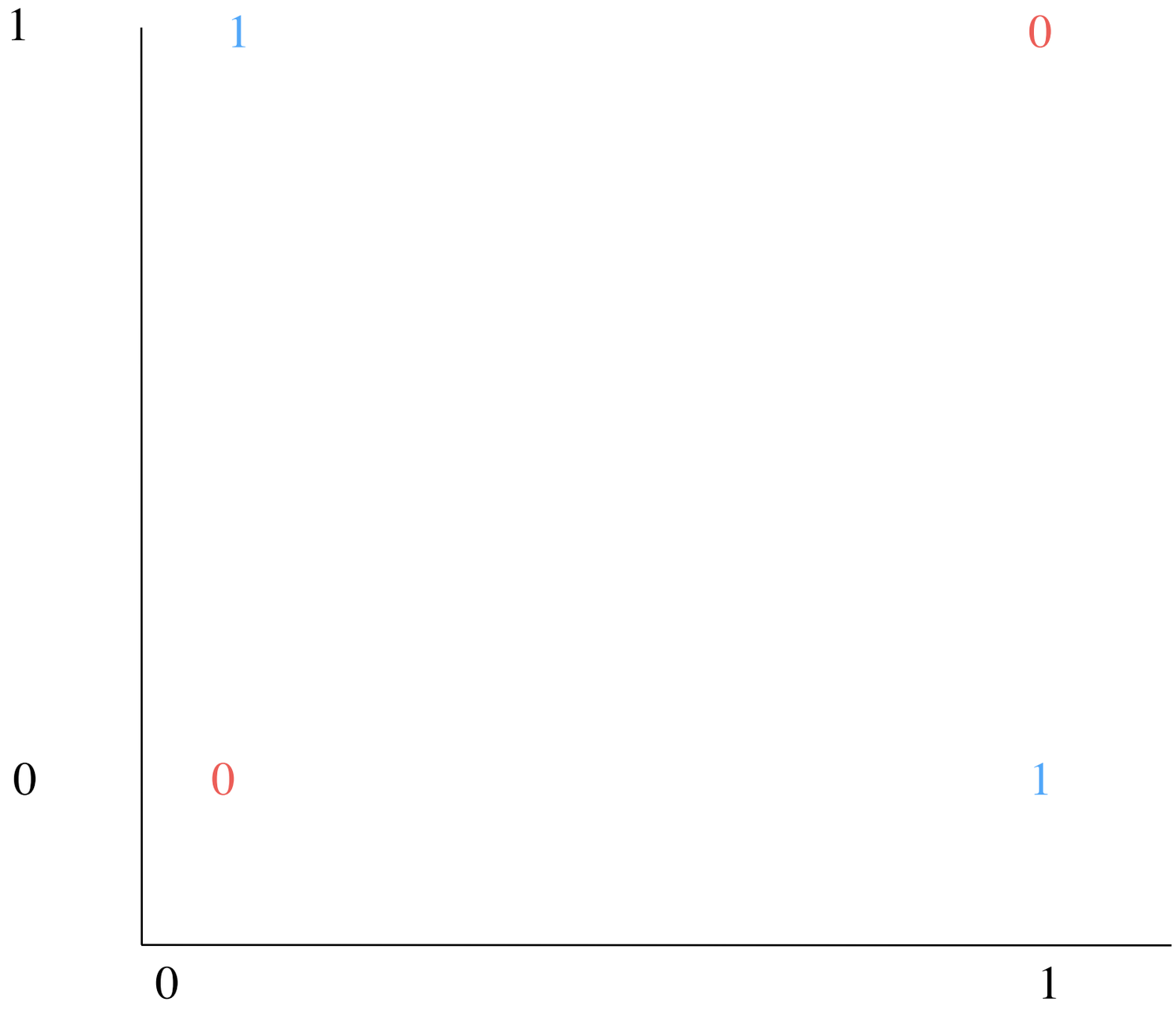


Support vector machines

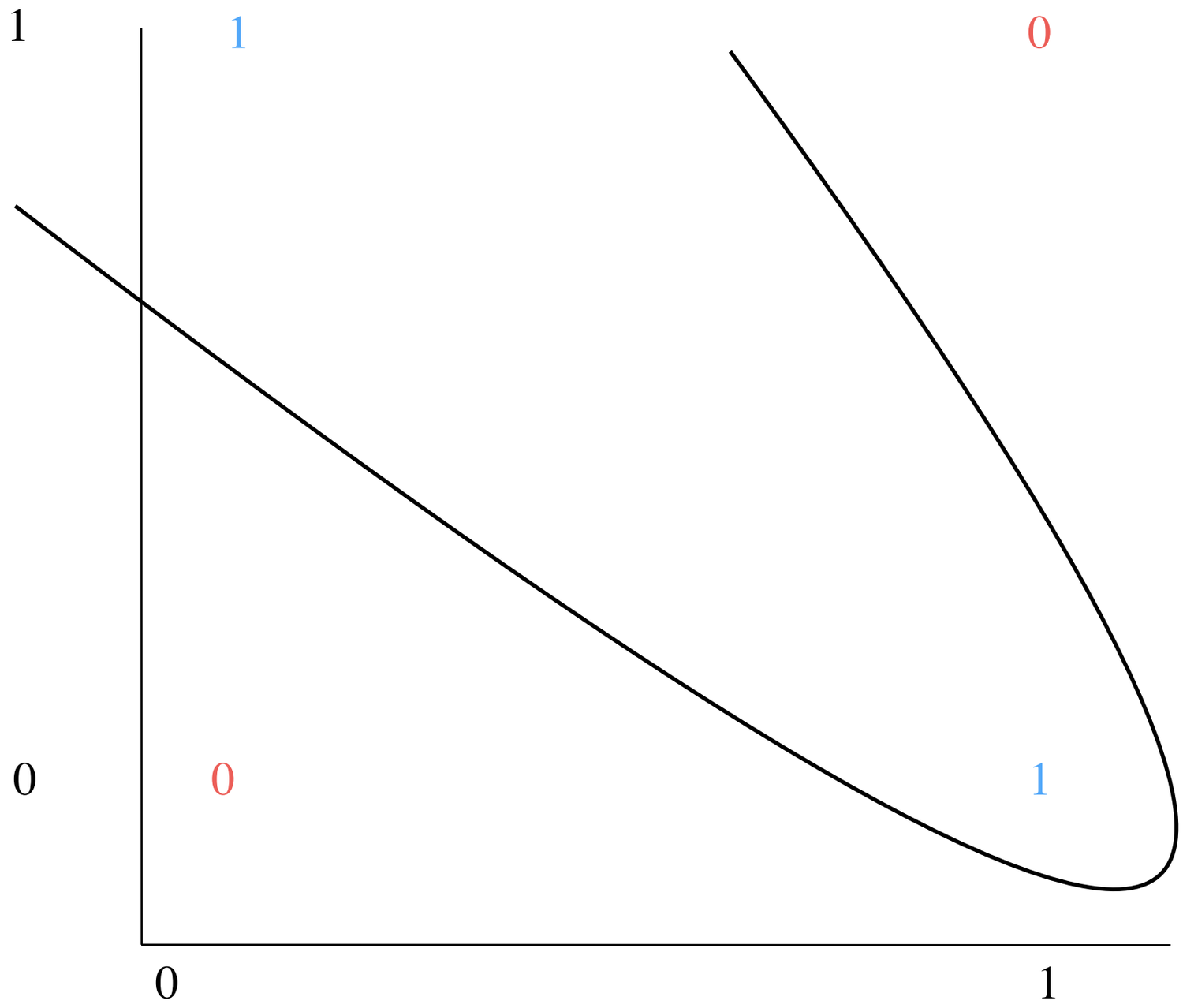
- Maximizing the margin: good according to intuition, theory, practice
- Only support vectors matter; other training examples are ignorable
- Support vector machines (SVMs) find the separator with max margin



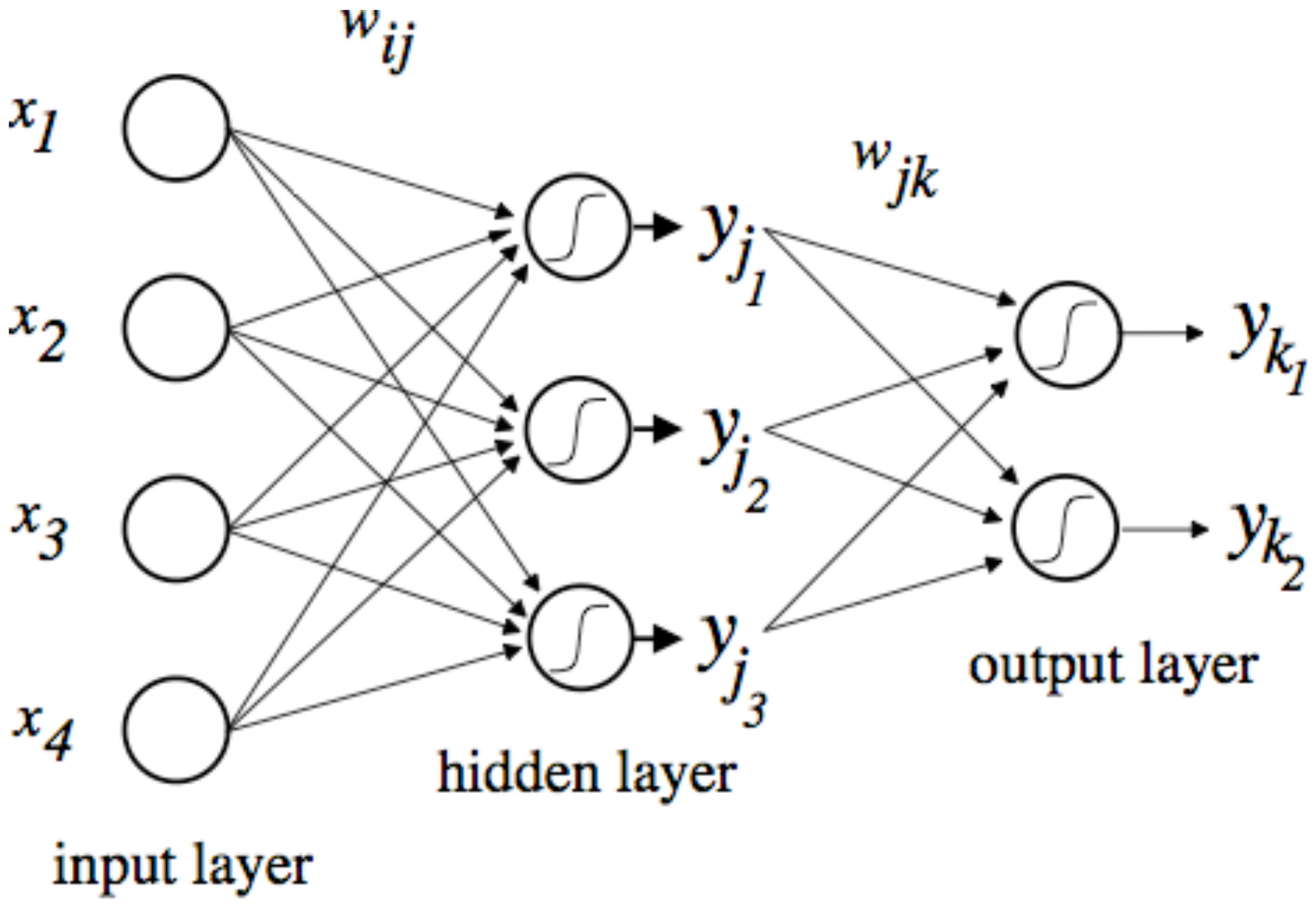
Problem: learn this!



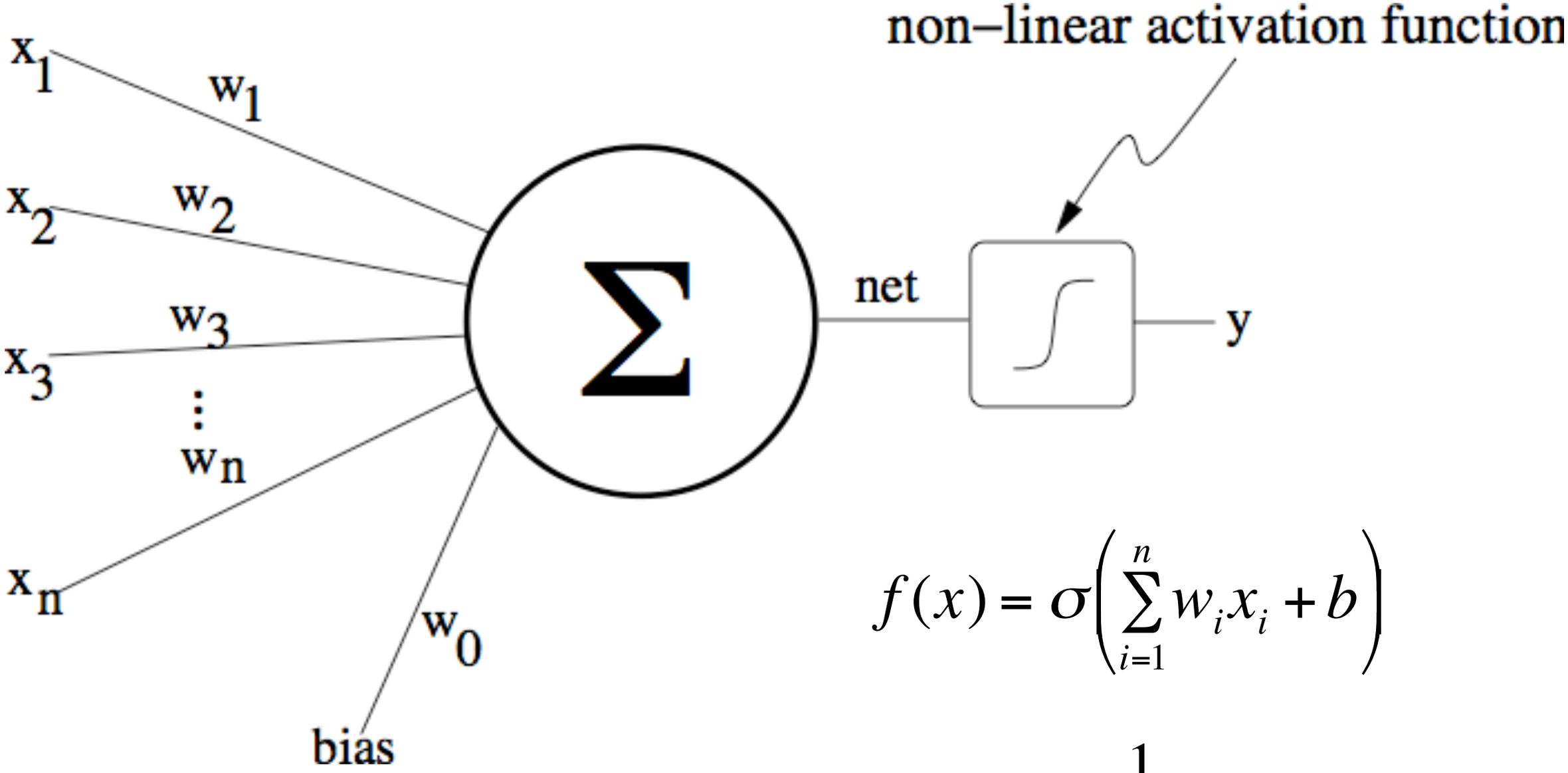
Problem: learn this!



Multi-layer Perceptron (MLP)



Non-Linear Neuron



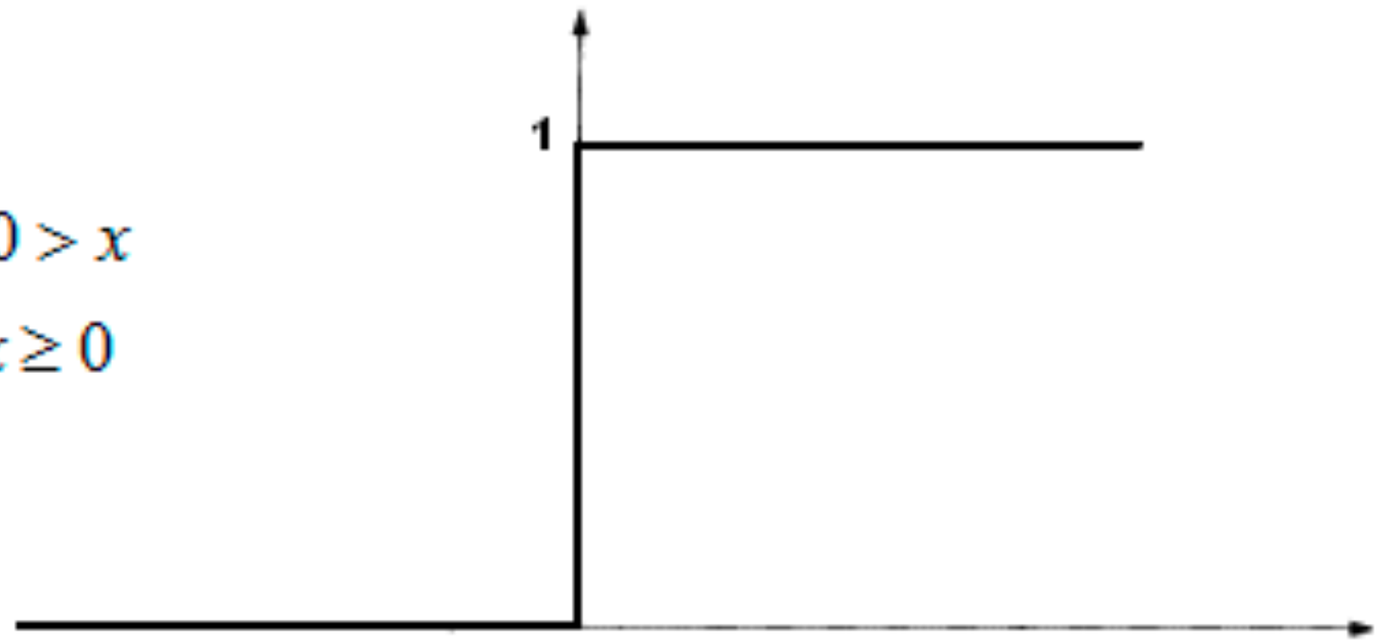
$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

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

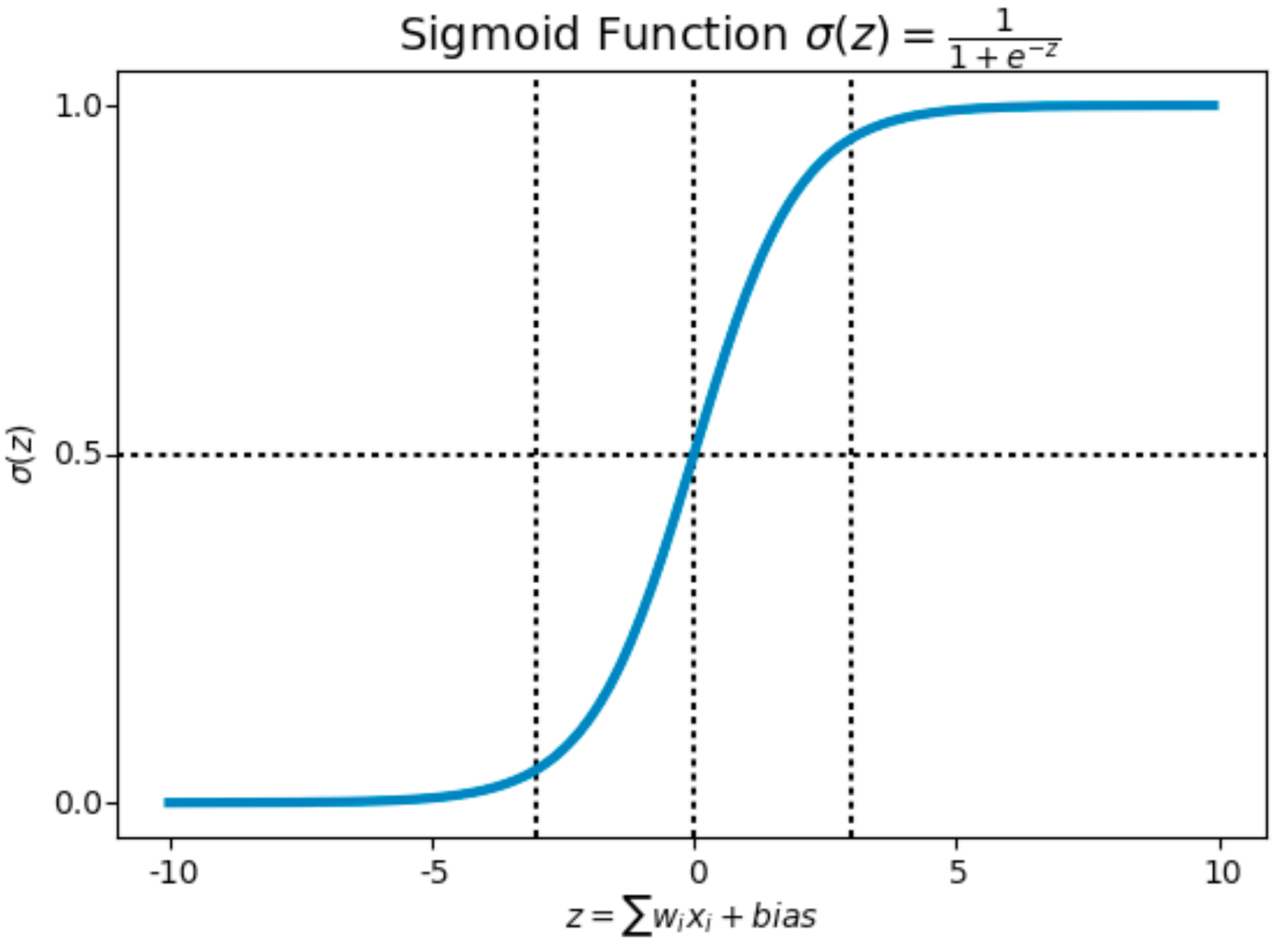
Activation Functions

Unit step (threshold)

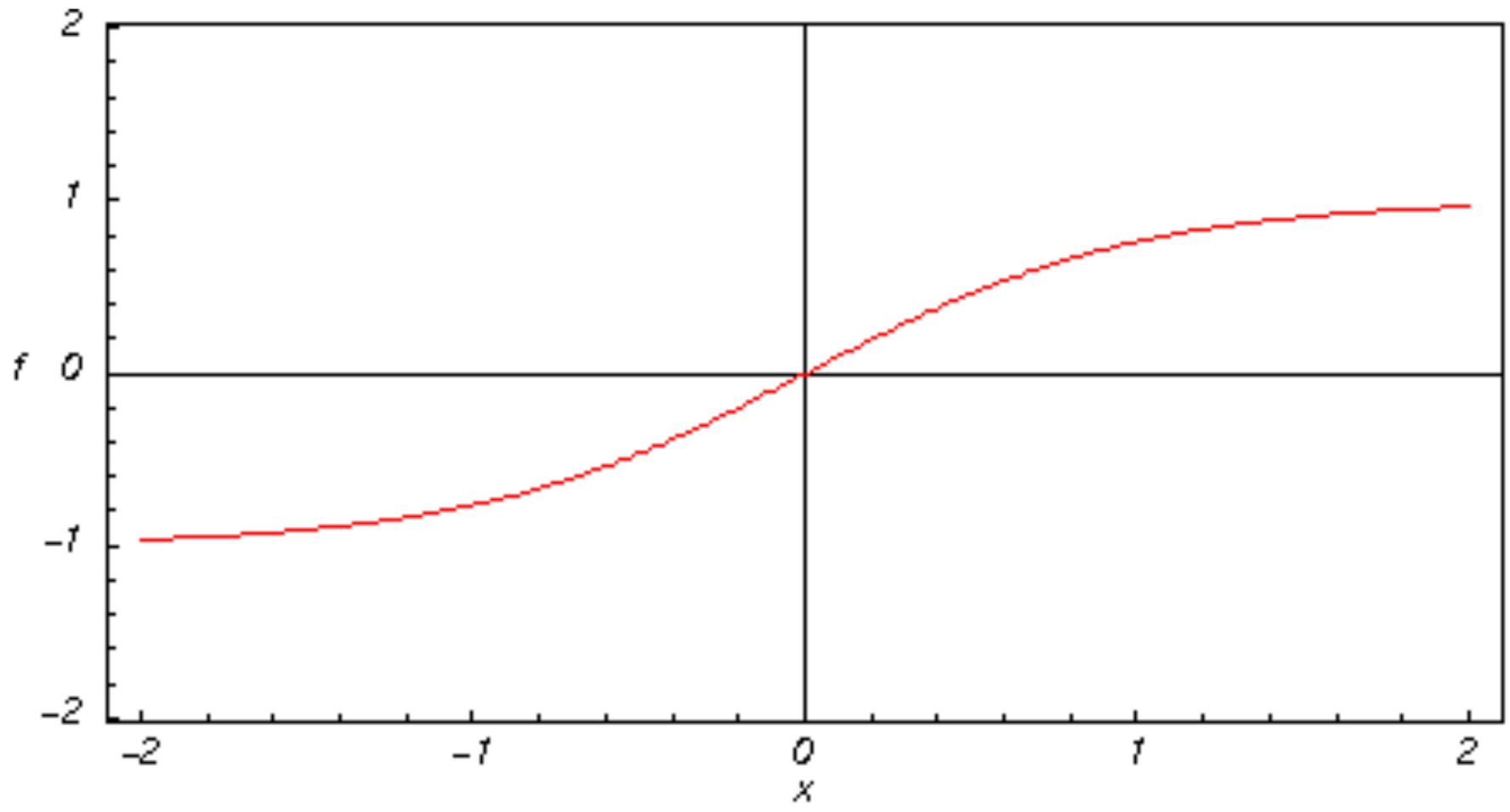
$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



Activation Functions

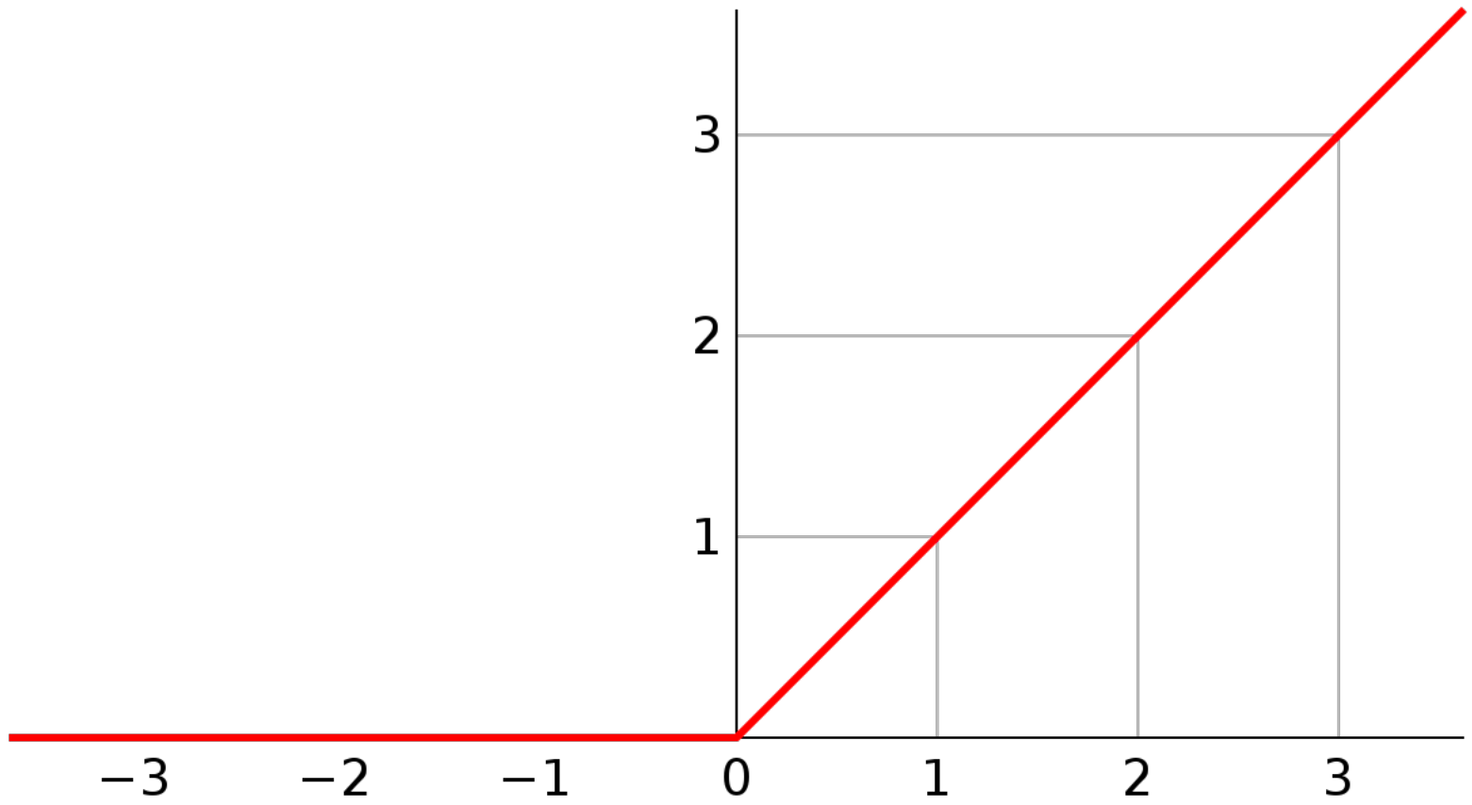


Activation Functions



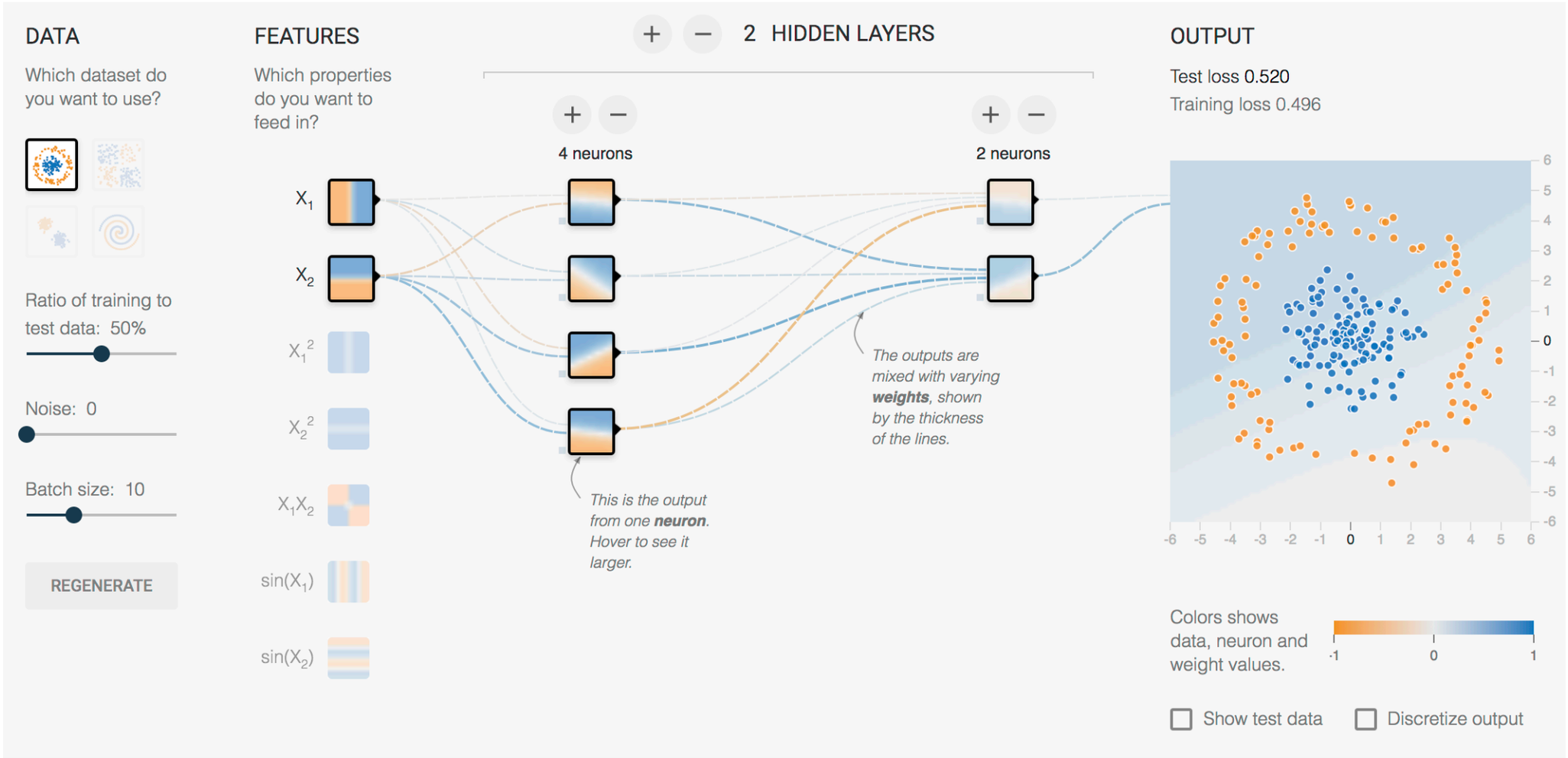
Hyperbolic Tangent

Activation Functions



Rectified Linear Units
(ReLu)

Elements of a Neural Network



http://playground.tensorflow.org/

Universal Approximator

- George Cybenko in 1989 provided one of the first proofs for the Universal Approximation Theorem for Neural Networks

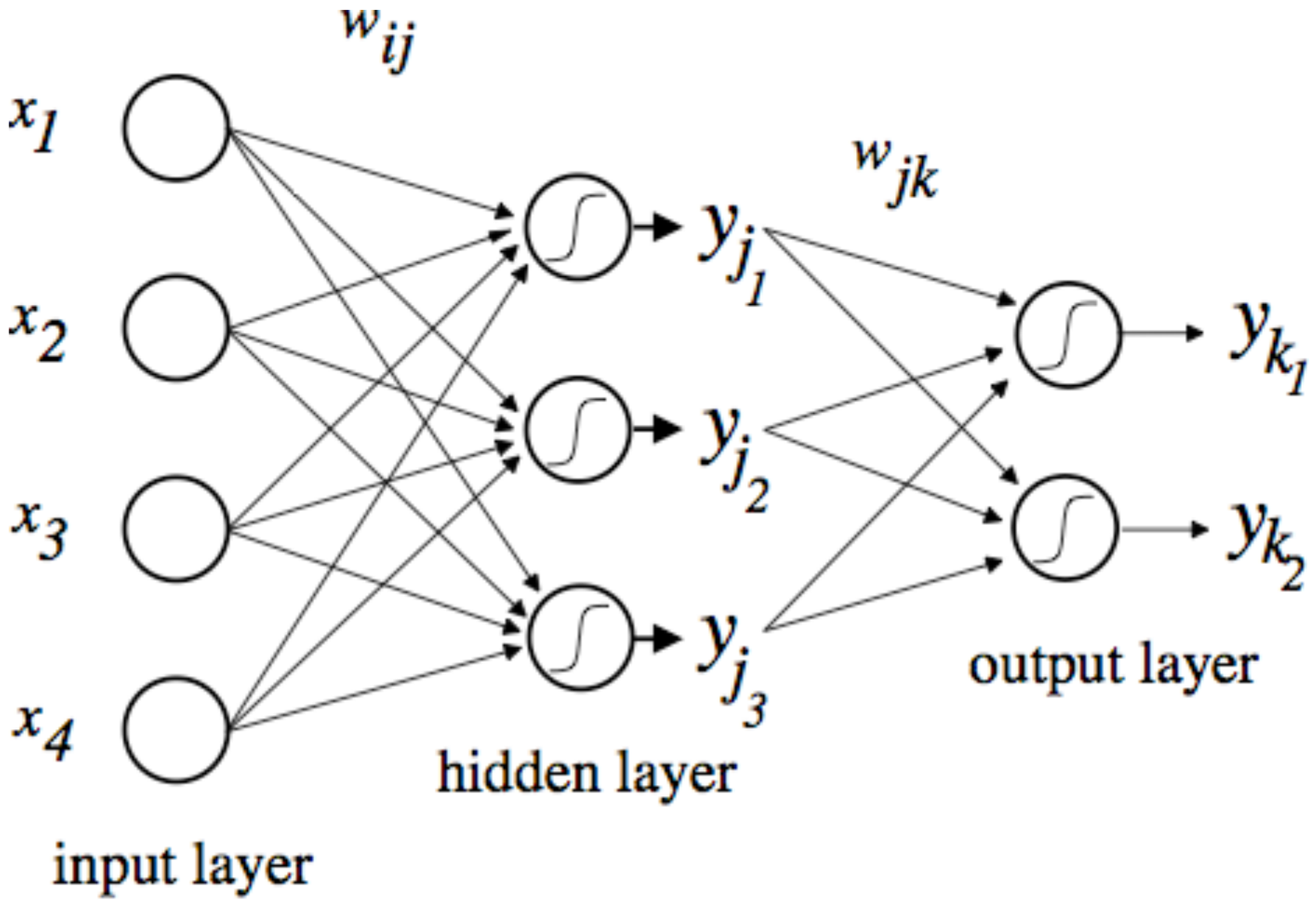
- States that a feedforward network with at least one hidden layer of a finite size can approximate any continuous function on compact subsets of R^n

- Says nothing about learnability

    - How do you train a hidden layer? What is its loss?

Training a Neural Network

Multi-layer Perceptron (MLP)



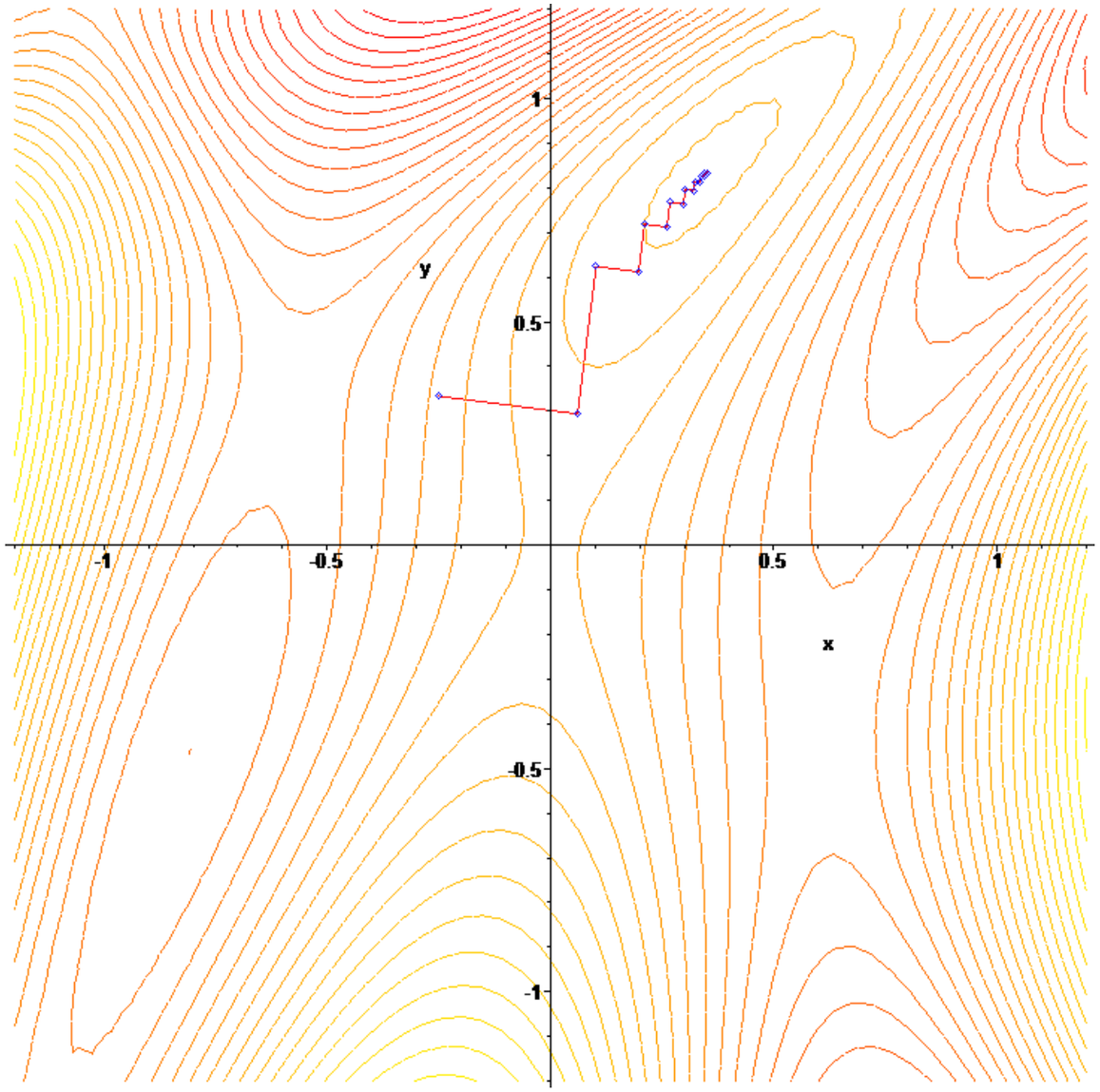
Backpropagation

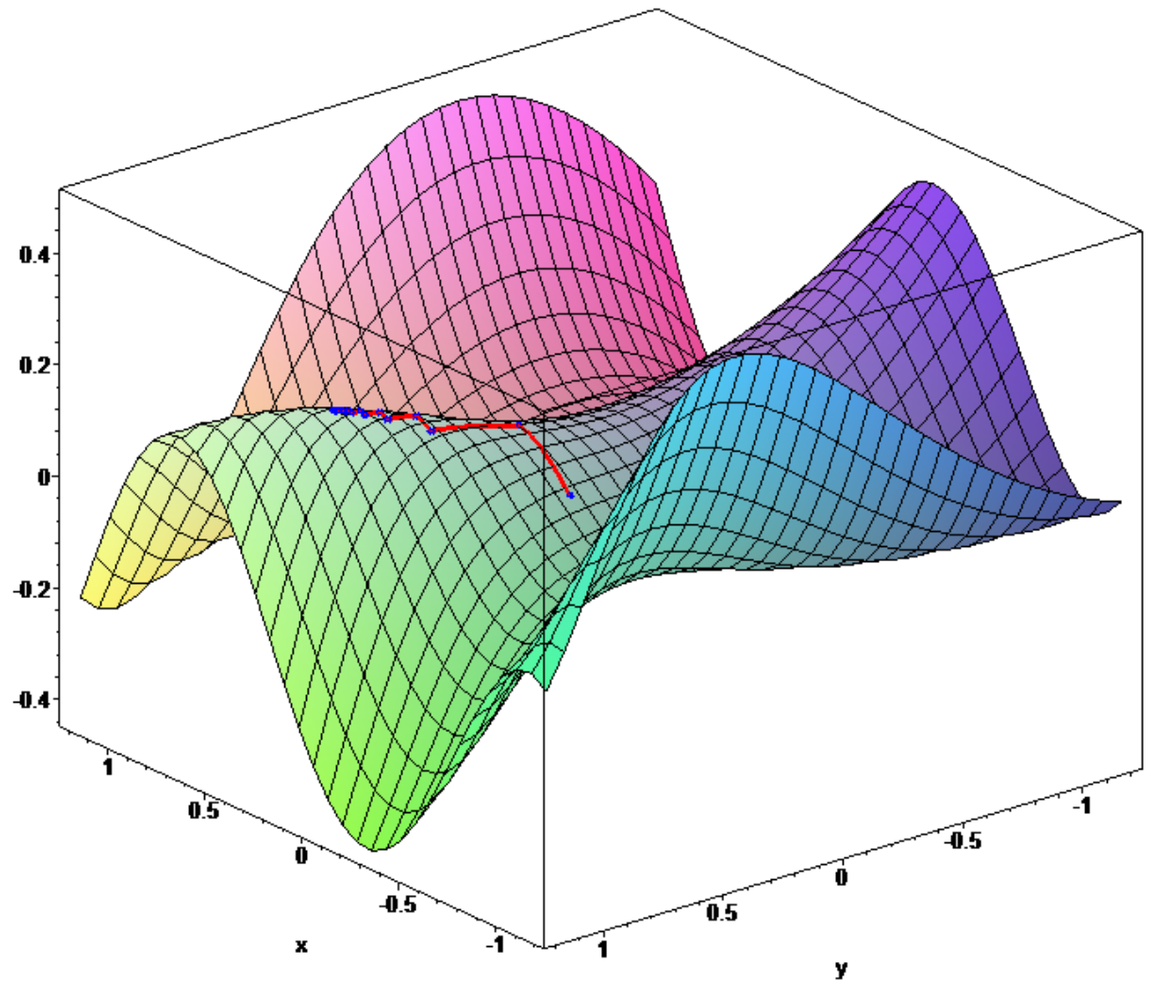
- Forward Pass: present training input pattern to network and activate network to produce output (can also do in batch: present all patterns in succession)

- Backward Pass: calculate error gradient and update weights starting at output layer and then going back

    - Modern libraries will compute the gradient for you!

Gradient Descent





Weight Update Equation

$$\theta_w' = \theta_w - \eta * \frac{\partial TC}{\partial \theta_w}$$

New weight

Learning Rate