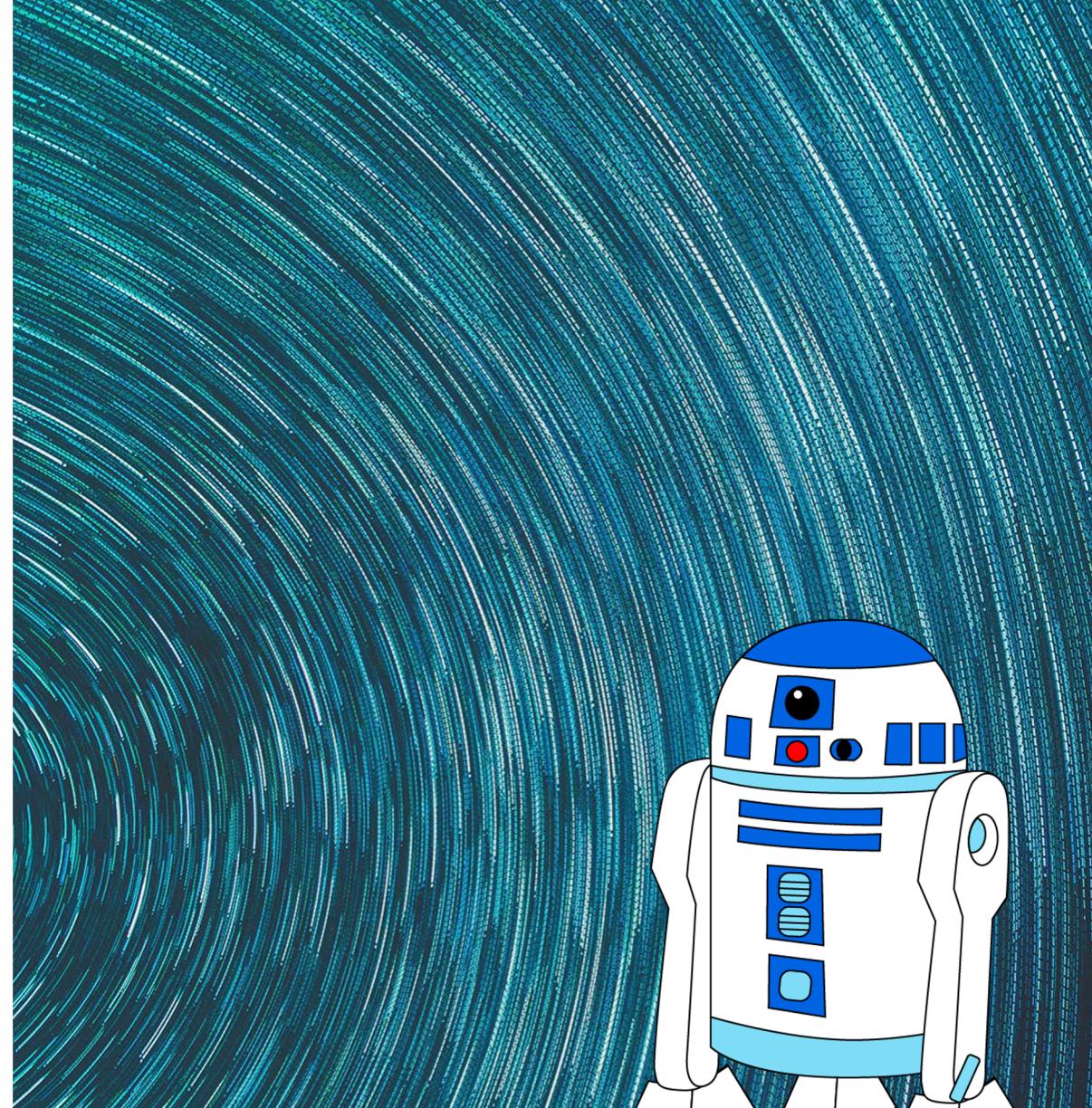


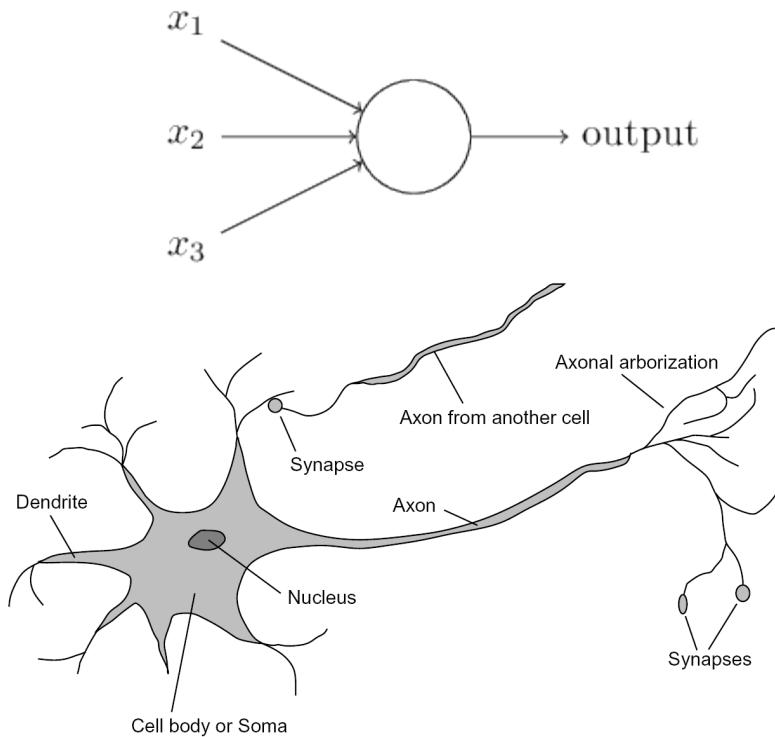
CIS 421/521:
ARTIFICIAL INTELLIGENCE

Perceptrons



Perceptrons

Perceptrons were developed in the 1950s and 1960s loosely inspired by the neuron.



Electronic 'Brain' Teaches Itself

The Navy last week demonstrated the embryo of an electronic computer named the Perceptron which, when completed in about a year, is expected to be the first non-living mechanism able to "perceive, recognize and identify its surroundings without human training or control." Navy officers demonstrating a preliminary form of the device in Washington said they hesitated to call it a machine because it is so much like a "human being without life."

Dr. Frank Rosenblatt, research psychologist at the Cornell Aeronautical Laboratory, Inc., Buffalo, N. Y., designer of the Perceptron, conducted the demonstration. The machine, he said, would be the first electronic device to think as the human brain. Like humans, Perceptron will make mistakes at first, "but it will grow wiser as it gains experience," he said.

The first Perceptron, to cost about \$100,000, will have about 1,000 electronic "association cells" receiving electrical impulses from an eyelike scanning device with 400 photocells. The human brain has ten billion responsive cells, including 100,000,000 connections with the eye.

Difference Recognized

recognize the difference between right and left, almost the way a child learns.

When fully developed, the Perceptron will be designed to remember images and information it has perceived itself, whereas ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons, Dr. Rosenblatt said, will be able to recognize people and call out their names. Printed pages, longhand letters and even speech commands are within its reach. Only one more step of development, a difficult step, he said, is needed for the device to hear speech in one language and instantly translate it to speech or writing in another language.

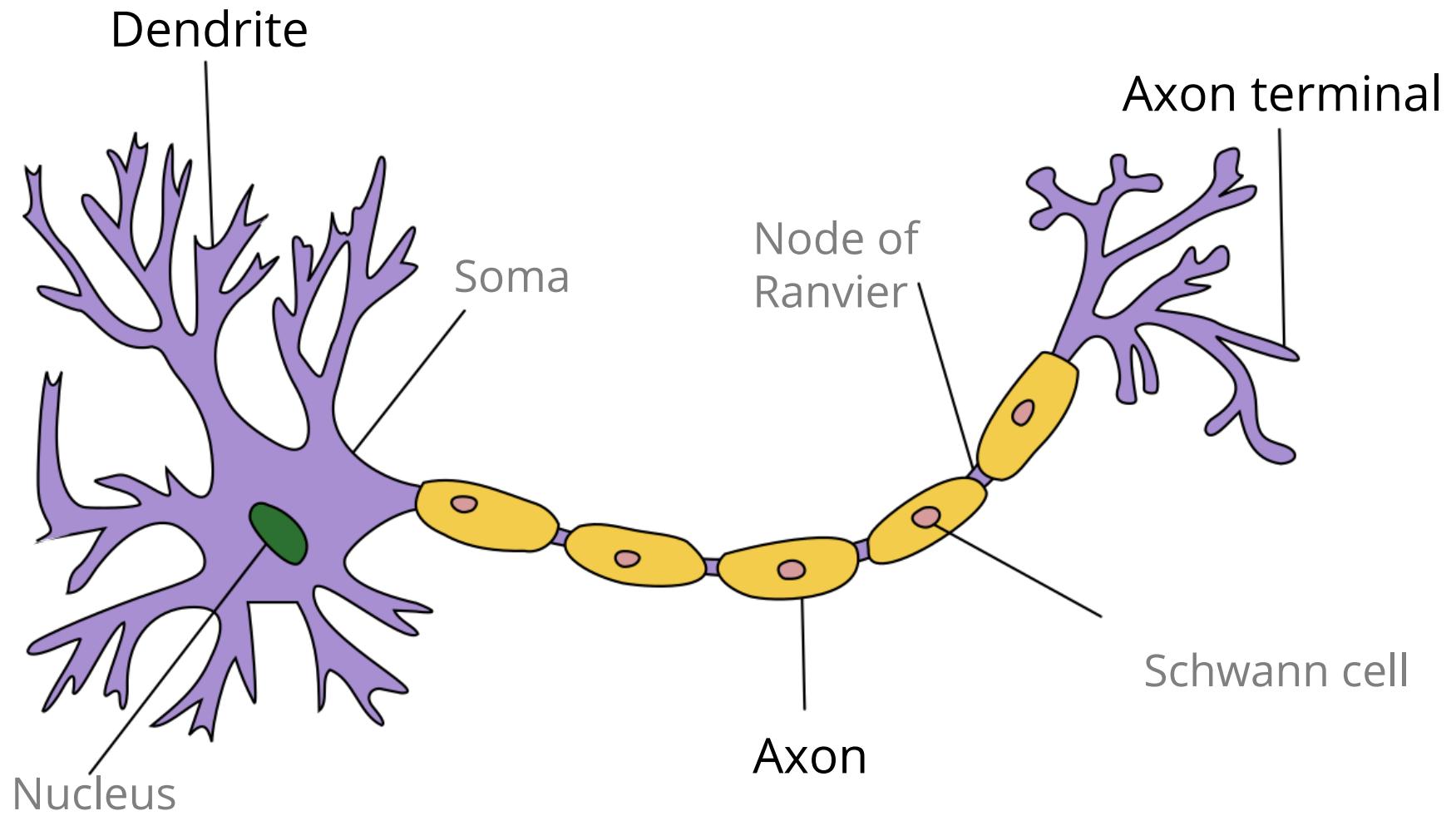
Self-Reproduction

In principle, Dr. Rosenblatt said, it would be possible to build Perceptrons that could reproduce themselves on an assembly line and which would be "conscious" of their existence.

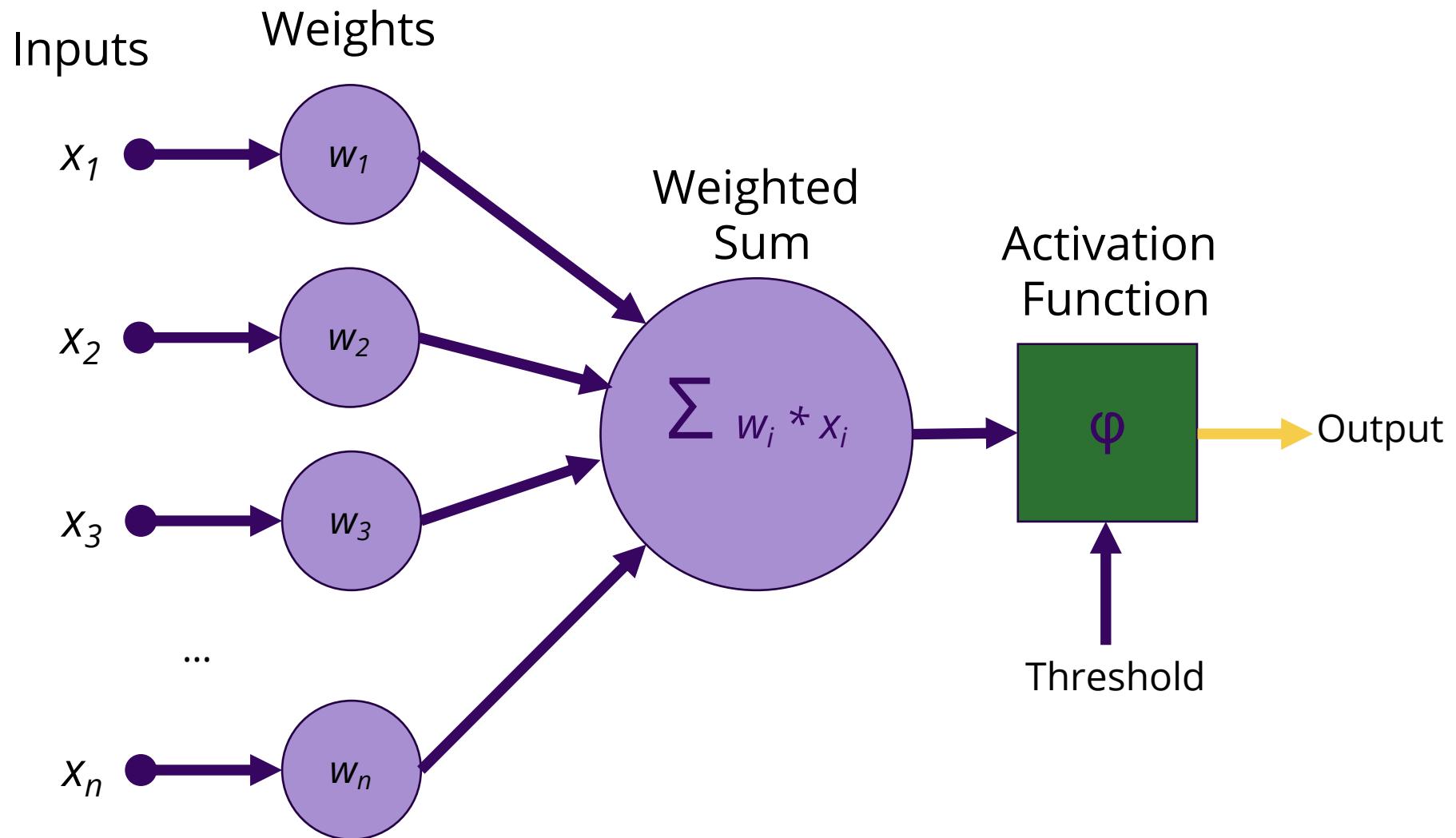
Perceptron, it was pointed out, needs no "priming." It is not necessary to introduce it to surroundings and circumstances, record the data involved and then store them for future comparison as is the case



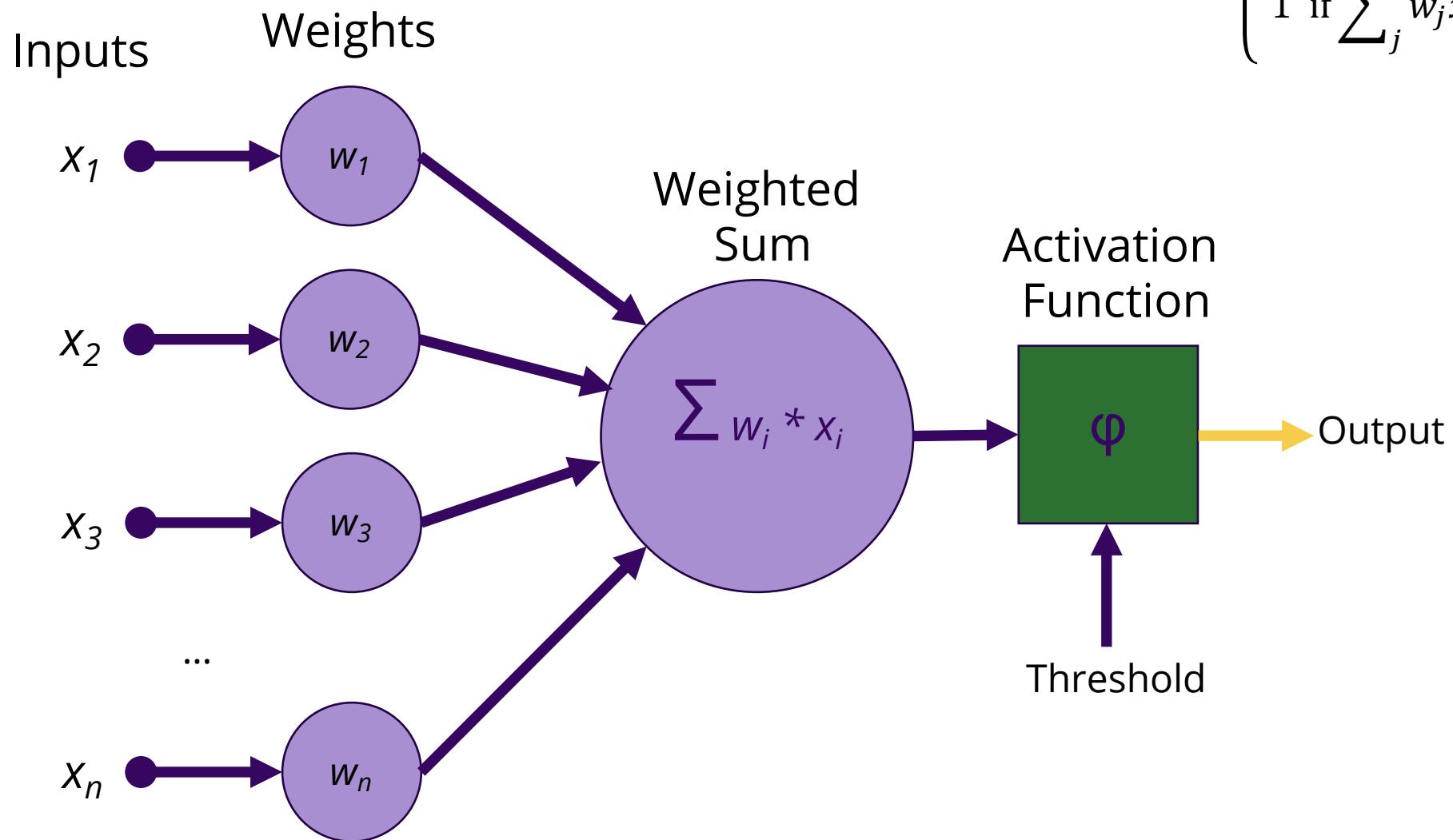
Neuron



Perceptron



Perceptron

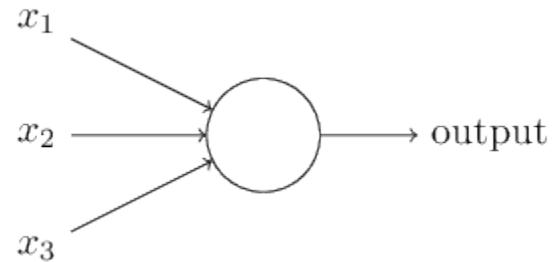


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Perceptrons for decision making

We can think about the perceptron or the sigmoid neuron as a device that makes decisions by weighing up evidence.

Example: Suppose there's a cheese festival in your town. You like cheese.



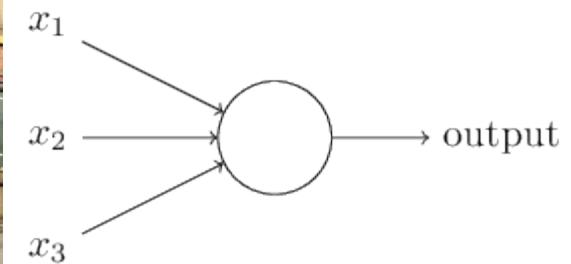
Example from Michael Nielsen's book [Neural Networks and Deep Learning](#)

Perceptrons for decision making

You might use 3 factors to decide whether to go.

1. Is the weather good?
2. Can your loyal companion come with you?
3. Is the festival near public transit?

These can be the binary input values to a perceptron

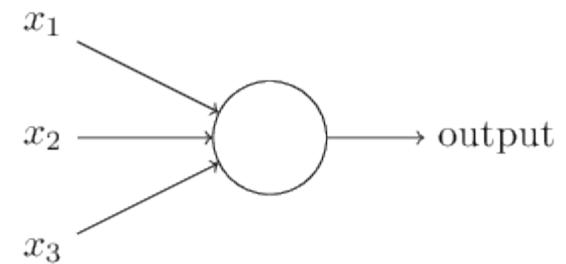
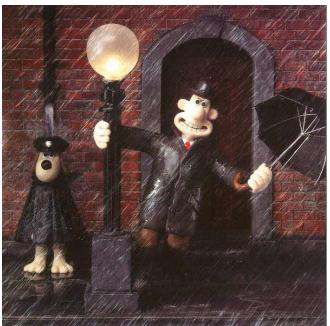


Perceptrons for decision making

By varying weights and the threshold we get different models of decision making

Example 1: $w_1 = 6 \quad w_2 = 2 \quad w_3 = 2$, threshold = 5

Example 2: $w_1 = 6 \quad w_2 = 2 \quad w_3 = 2$, threshold = 3



Notational changes

Change 1: We can write $\sum_j w_j x_j$ as a dot product of the input vector and the weight vector:

$$\sum_j w_j x_j \equiv \mathbf{w} \cdot \mathbf{x}$$

Change 2: We can move the threshold to other other side of the inequality. We define a perceptron's "bias" as the -1 * its threshold:

$$b \equiv -\text{threshold}$$

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

rewrites to

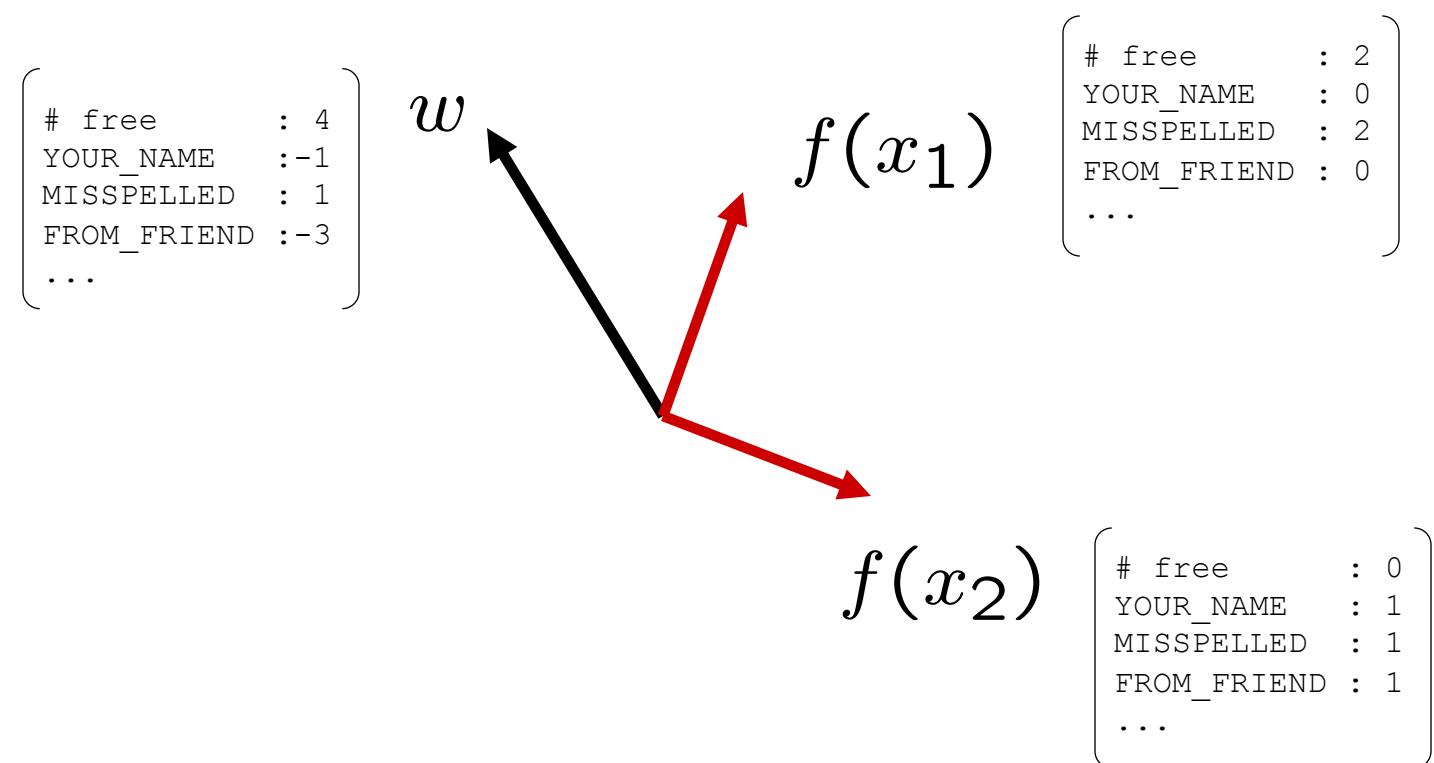
$$\text{output} = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

Learning weights from examples

Perceptions can be used for all kinds of classification problems.

Think of the inputs as *features* representing something we want to classify.

The feature values for inputs are fixed, but we can choose different weight vectors. Depending on the weight vector that we pick, we will get a different classifier.

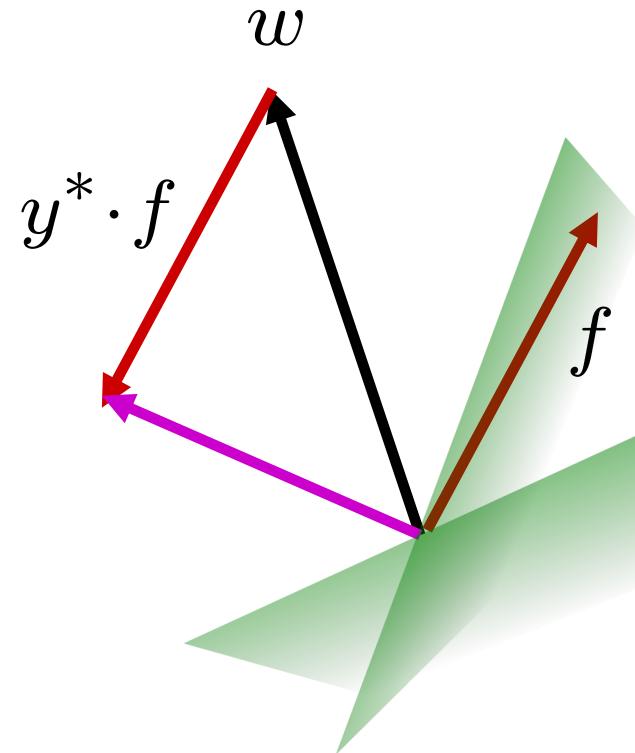


Learning a Binary Perceptron

Start with weights = 0

For each training instance:

- Classify with current weights
- $y = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{f}(x) \leq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{f}(x) > 0 \end{cases}$
- If correct (i.e., $y=y^*$), no change!
- If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y^* is -1.
- $\mathbf{w} = \mathbf{w} + y^* \cdot \mathbf{f}$



Multiclass Decision Rule

If we have multiple classes:

- A weight vector for each class:

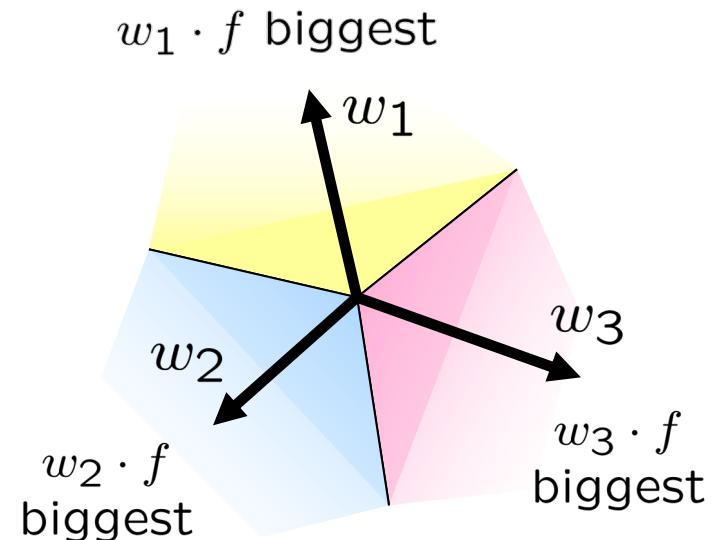
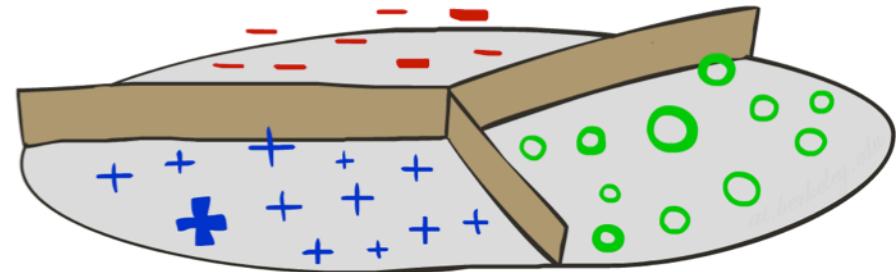
$$w_y$$

- Score (activation) of a class y :

$$w_y \cdot f(x)$$

- Prediction highest score wins

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



Binary = multiclass where the negative class has weight zero

Learning: Multiclass Perceptron

Start with all weights = 0

Pick up training examples one by one

Predict with current weights

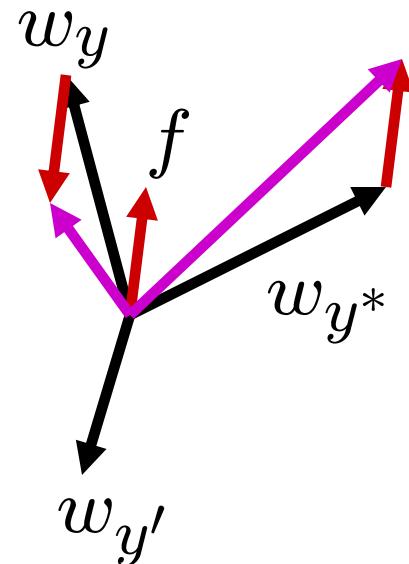
$$y = \arg \max_y w_y \cdot f(x)$$

If correct, no change!

If wrong: lower score of wrong answer. raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$



Example: Multiclass Perceptron

“win the vote”

“win the election”

“win the game”

w_{SPORTS}

BIAS	:	1
win	:	0
game	:	0
vote	:	0
the	:	0
...		

$w_{POLITICS}$

BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

w_{TECH}

BIAS	:	0
win	:	0
game	:	0
vote	:	0
the	:	0
...		

Properties of Perceptrons

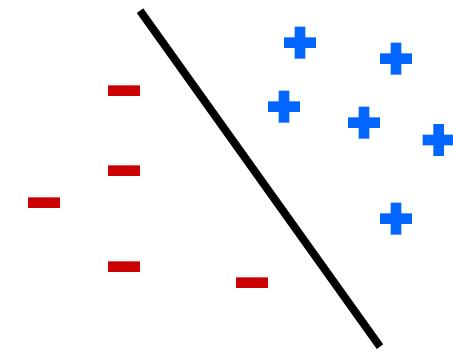
Separability: true if some parameters get the training set perfectly correct

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

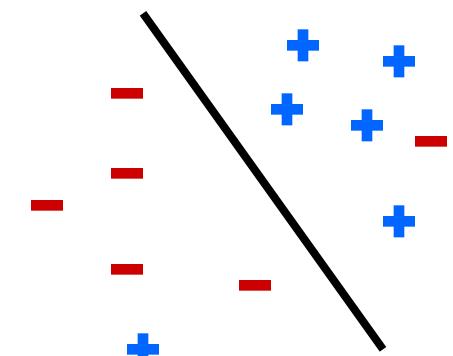
Mistake Bound: the maximum number of mistakes (binary case) related to the *margin* or degree of separability

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

Separable

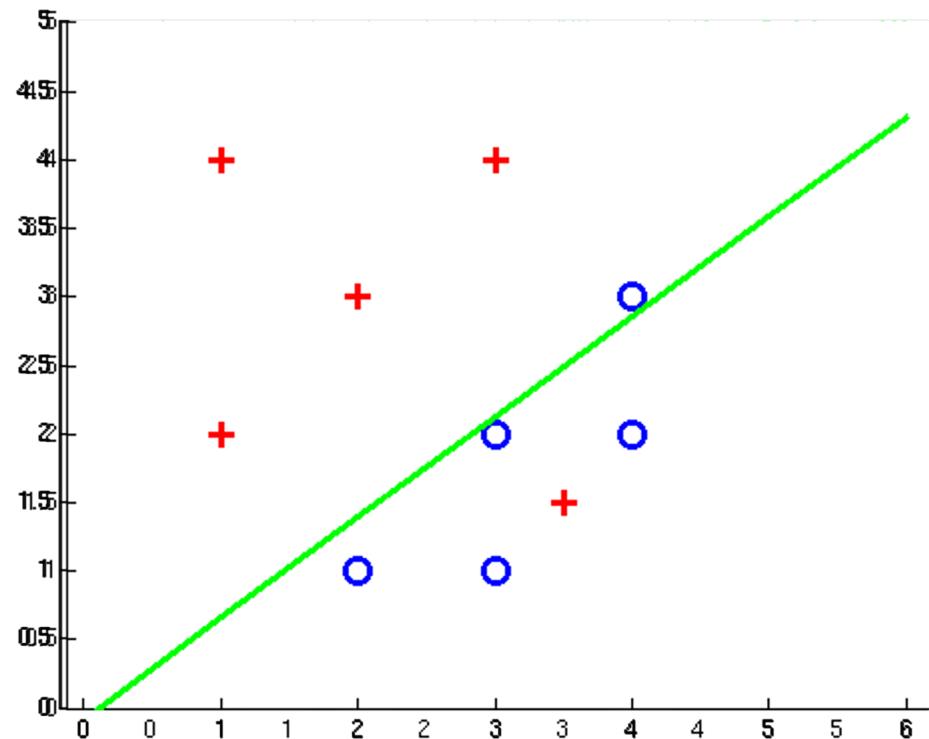


Non-Separable

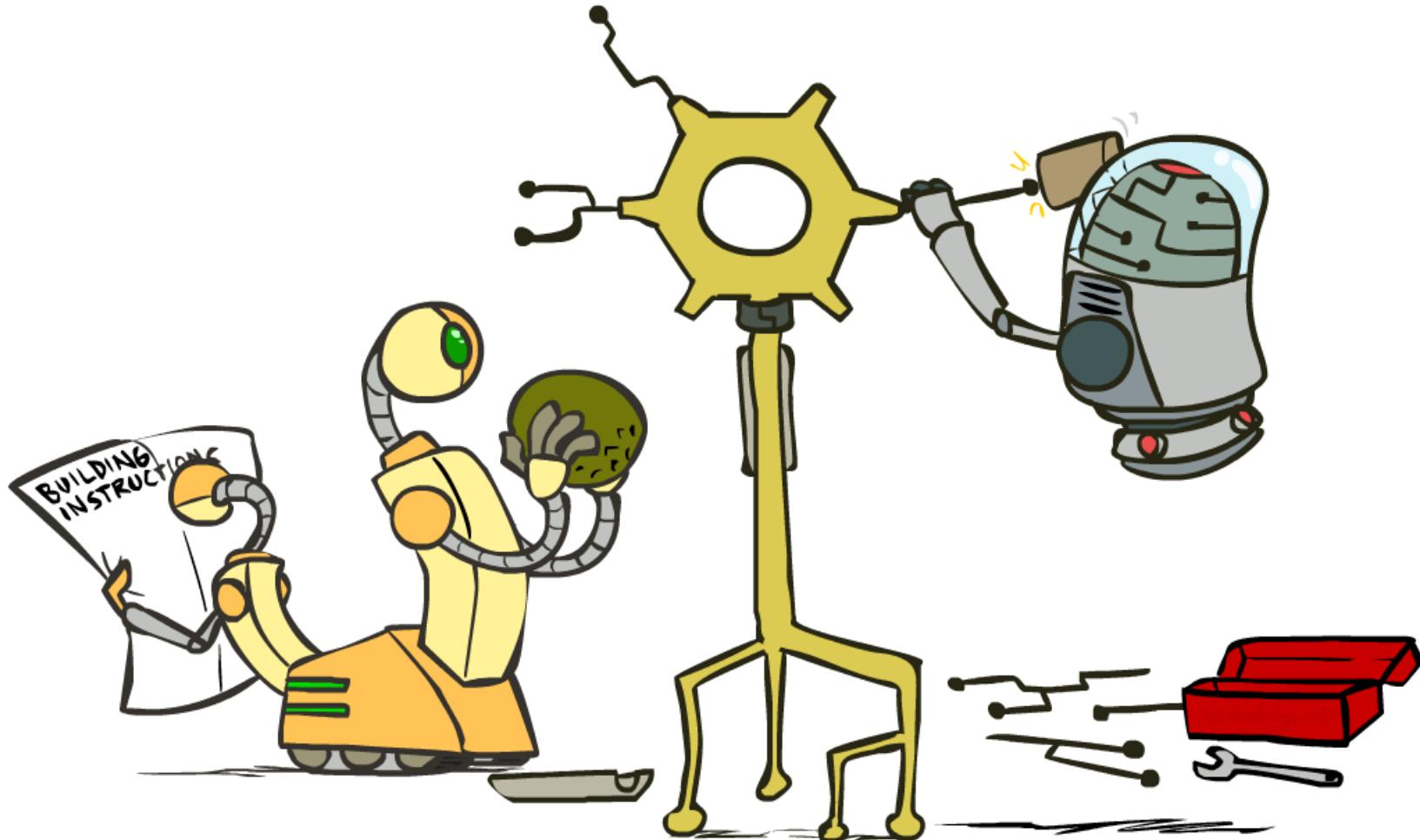


Examples: Perceptron

Non-Separable Case



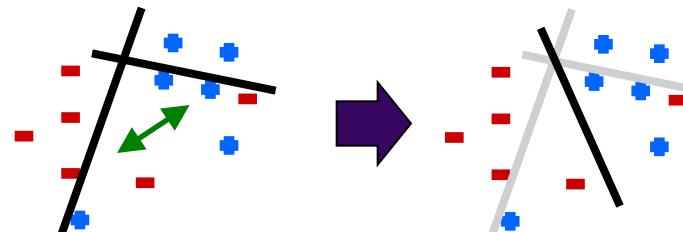
Improving the Perceptron



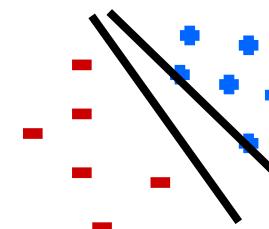
Problems with the Perceptron

Noise: if the data isn't separable, weights might thrash

- Averaging weight vectors over time can help (averaged perceptron)

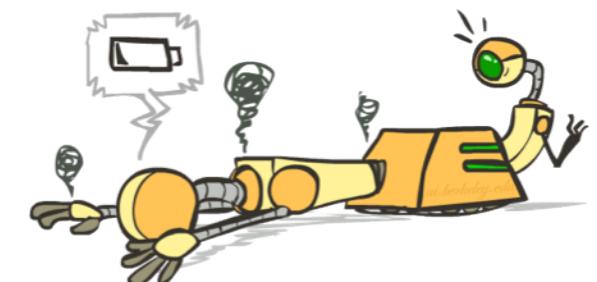
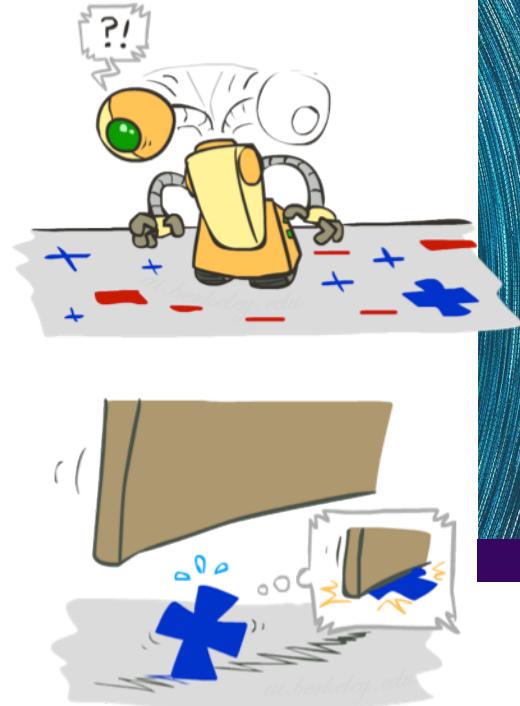
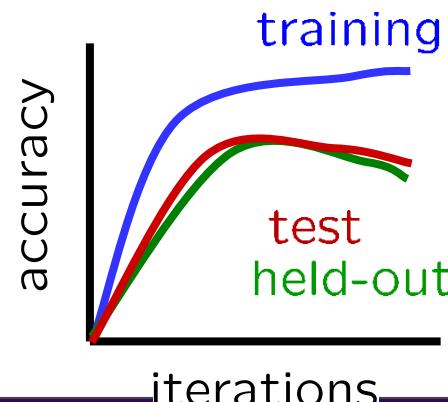


Mediocre generalization: finds a "barely" separating solution



Overtraining: test / held-out accuracy usually rises, then falls

- Overtraining is a kind of overfitting



Fixing the Perceptron

Idea: adjust the weight update to mitigate these effects

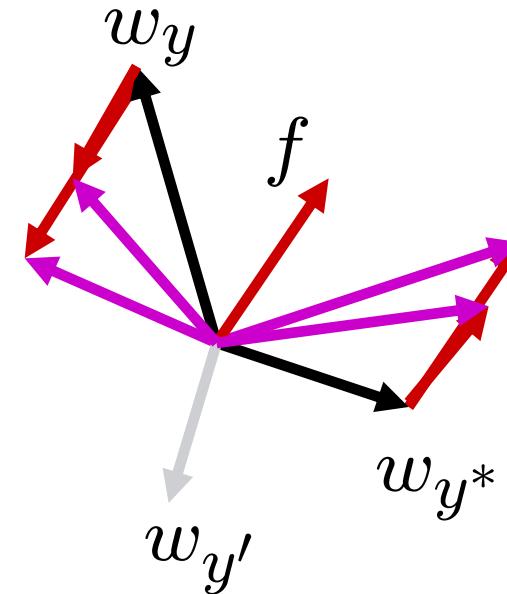
MIRA = Margin Infused Relaxed Algorithm

Choose an update size that fixes the current mistake...

... but, minimizes the change to w

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}$$

The $+1$ helps to generalize



Guessed y instead of y^* on example x with features $f(x)$

$$w_y = w'_y - \tau f(x)$$

$$w_{y^*} = w'_{y^*} + \tau f(x)$$

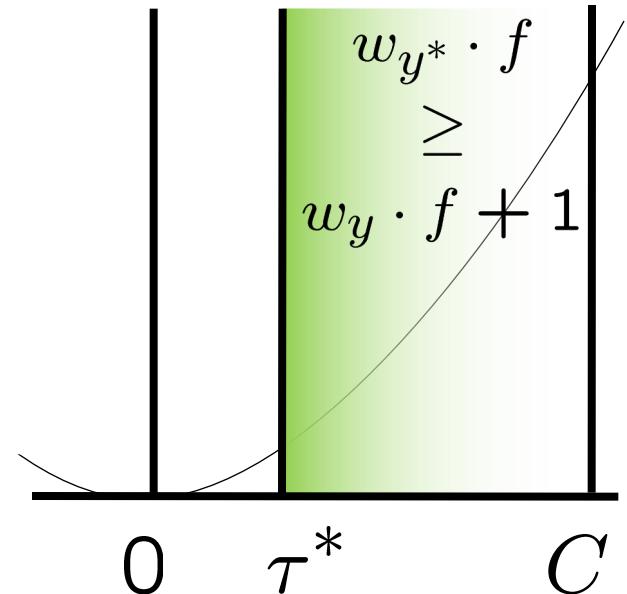
Maximum Step Size

In practice, it's also bad to make updates that are too large

- Example may be labeled incorrectly
- You may not have enough features
- Solution: cap the maximum possible value of τ with some constant C

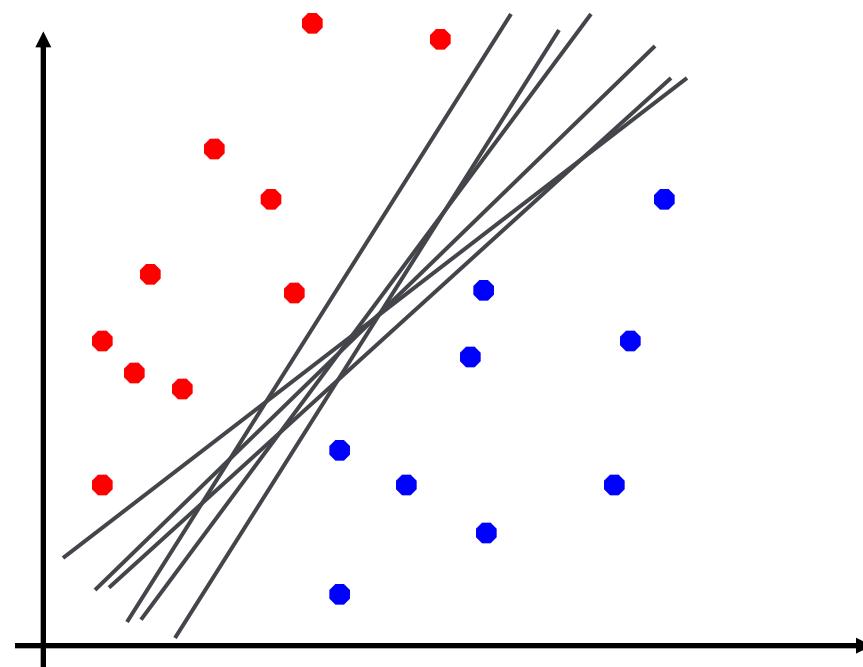
$$\tau^* = \min \left(\frac{(w'_y - w'_{y^*}) \cdot f + 1}{2f \cdot f}, C \right)$$

- Corresponds to an optimization that assumes non-separable data
- Usually converges faster than perceptron
- Usually better, especially on noisy data



Linear Separators

Which of these linear separators is optimal?



Support Vector Machines

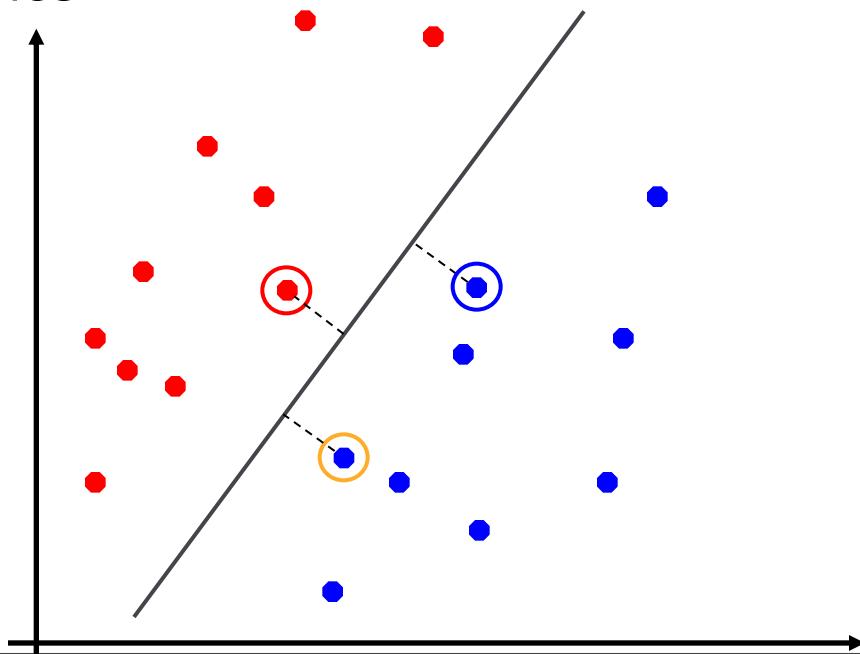
Maximizing the margin: good according to intuition, theory, practice

Only support vectors matter; other training examples are ignorable

Support vector machines (SVMs) find the separator with max margin

Basically, SVMs are MIRA where you optimize over all examples at once

$$\min_w \frac{1}{2} \|w - w'\|^2$$
$$w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

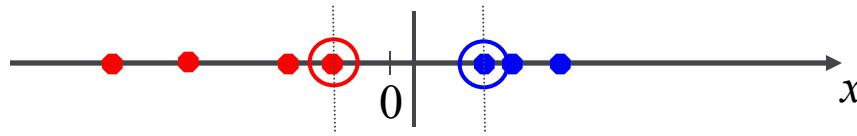


SVM

$$\min_w \frac{1}{2} \|w\|^2$$
$$\forall i, y \quad w_{y^*} \cdot f(x_i) \geq w_y \cdot f(x_i) + 1$$

Non-Linear Separators

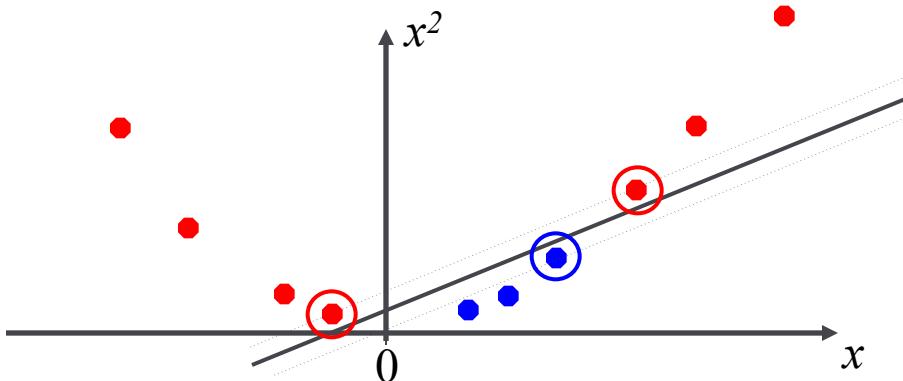
Data that is linearly separable works out great for linear decision rules:



But what are we going to do if the dataset is just too hard?

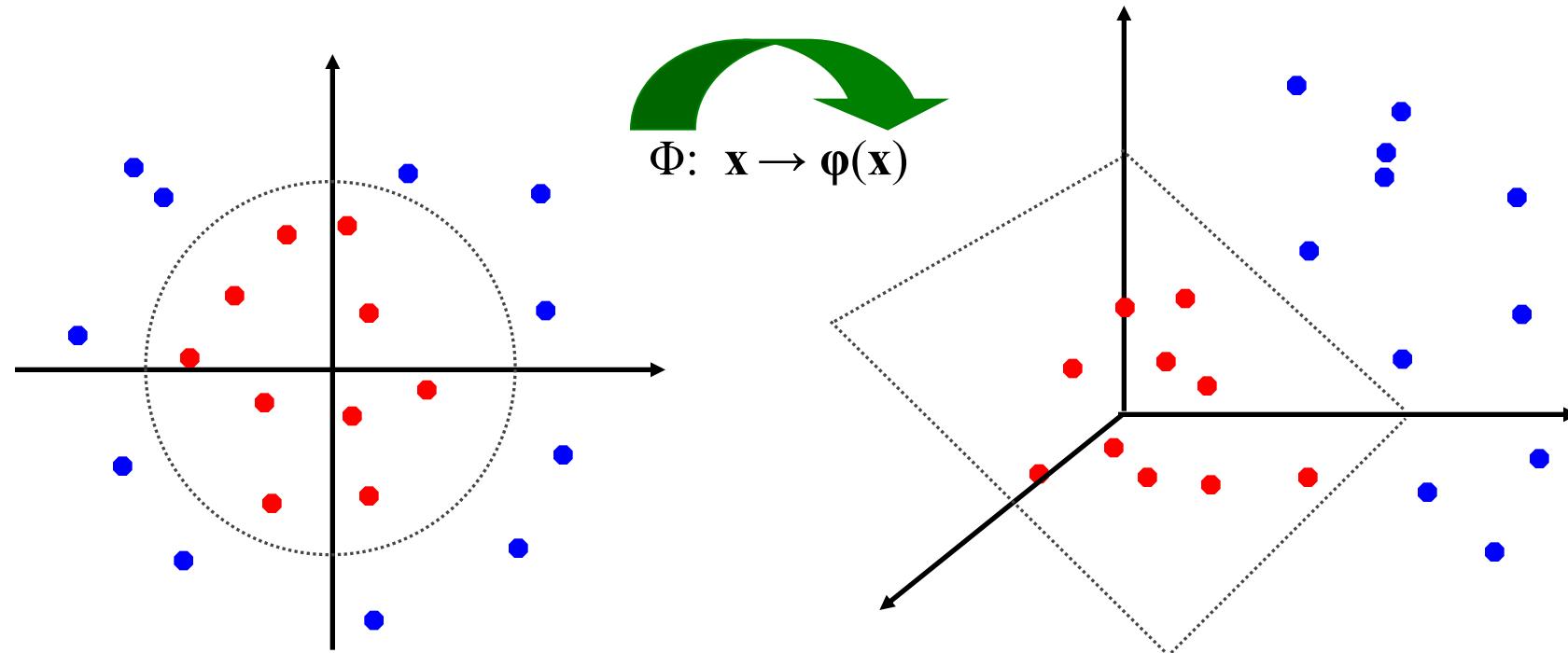


How about... mapping data to a higher-dimensional space:



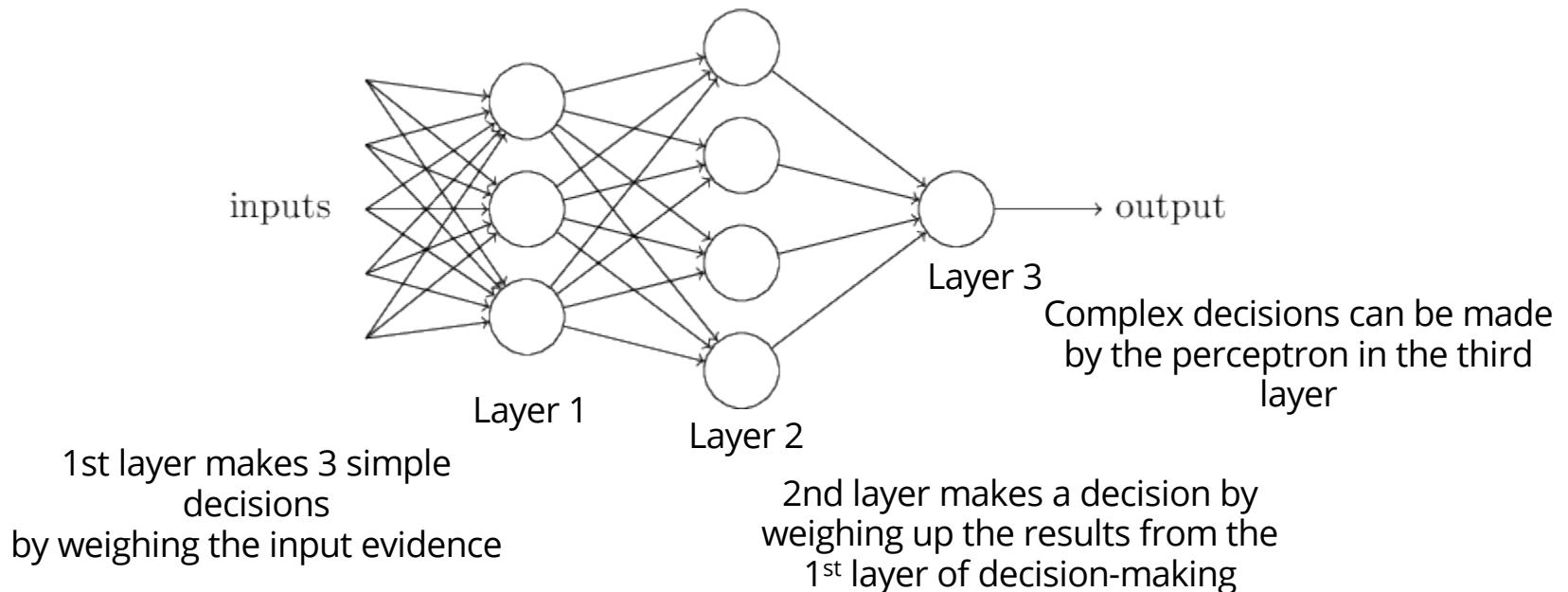
Non-Linear Separators

General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



Perceptrons for decision making

- A complex network of perceptrons could make quite subtle decisions:



Weights, bias and dot products

- Two notational changes simplify the way that perceptrons are described.
- The first change is to replace the weighted sum as a dot product

$$w \cdot x \equiv \sum_j w_j x_j$$

- The second change is to move the threshold to the other side of the inequality, and to replace it by a *bias*, b -threshold

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Activation Functions

Instead of directly outputting $z = w \cdot x + b$, which is a linear function of x , neuron units apply a non-linear function f to z .

The output of this function is called the **activation value** for the unit, represented by the variable **a**. The output of a neural network is called **y**, so if the activation of a node is the final output of a network then

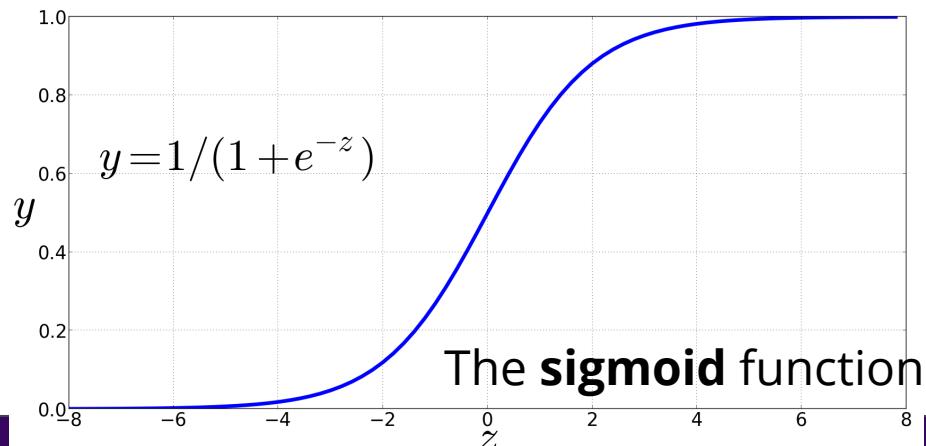
$$y=a=f(z)$$

There are 3 commonly used non-linear functions used for f :

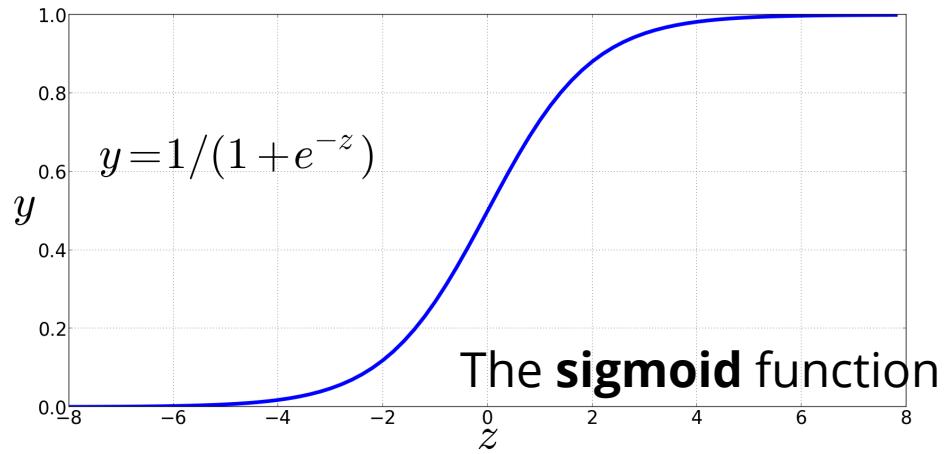
The **sigmoid** function

The **tanh** function

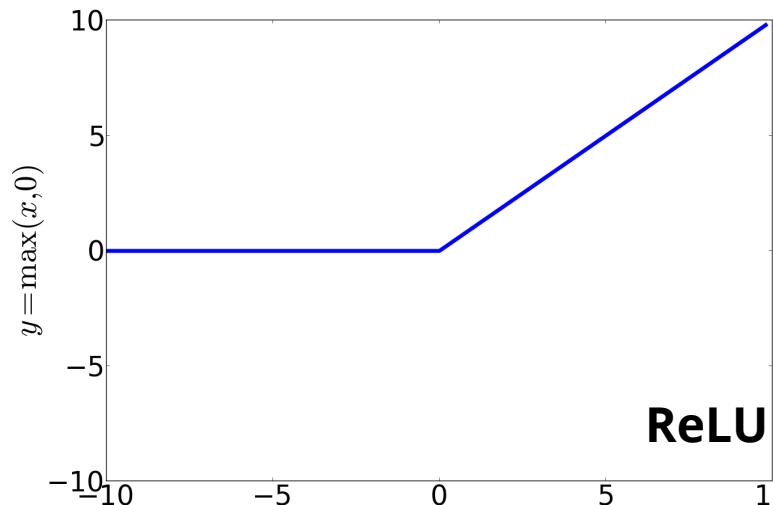
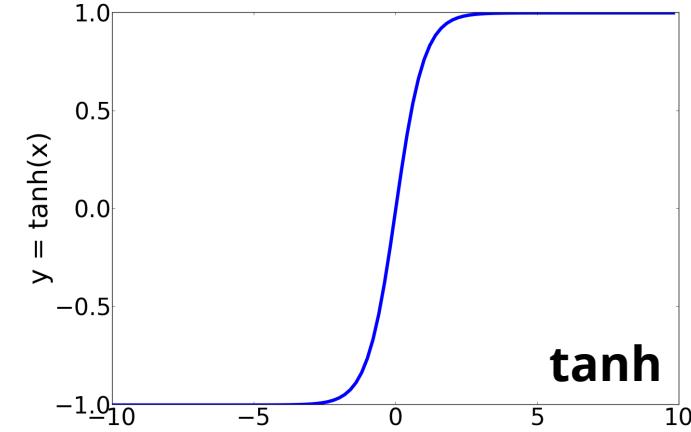
The **rectified linear unit ReLU**



Activation Functions



The sigmoid function



ReLU

The XOR problem

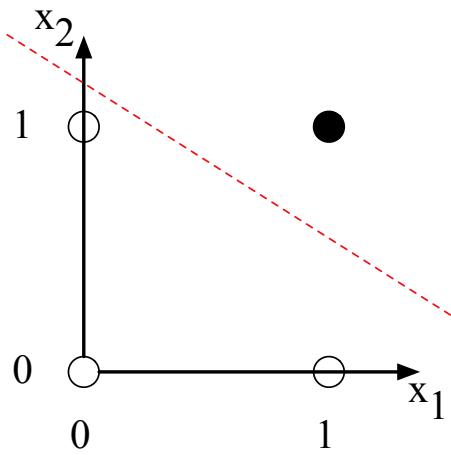
A single neural unit cannot be used to compute the XOR function

AND		
x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

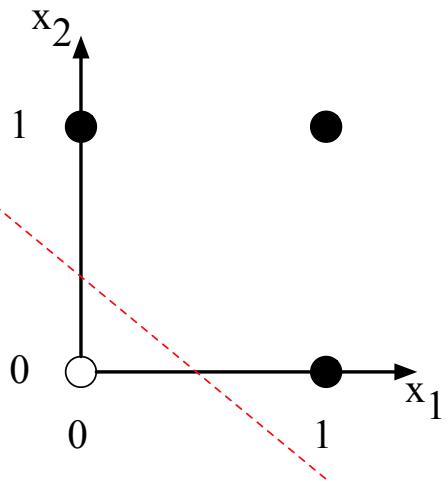
OR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

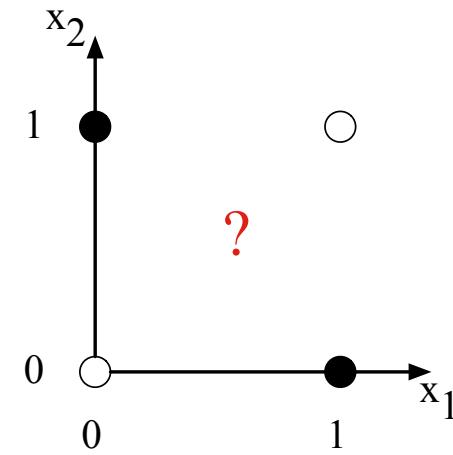
The XOR Problem



a) x_1 AND x_2

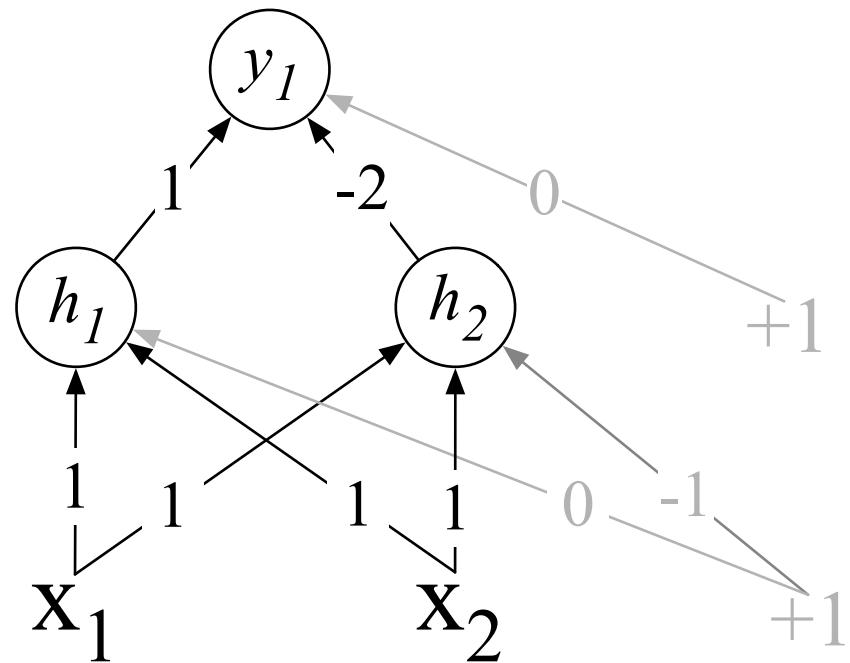


b) x_1 OR x_2

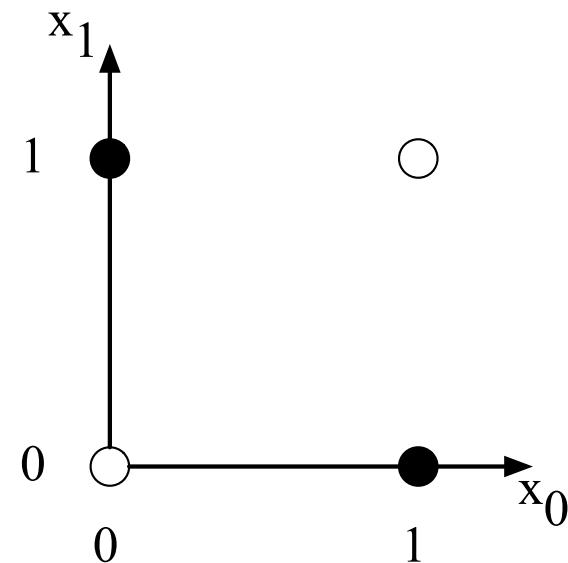


c) x_1 XOR x_2

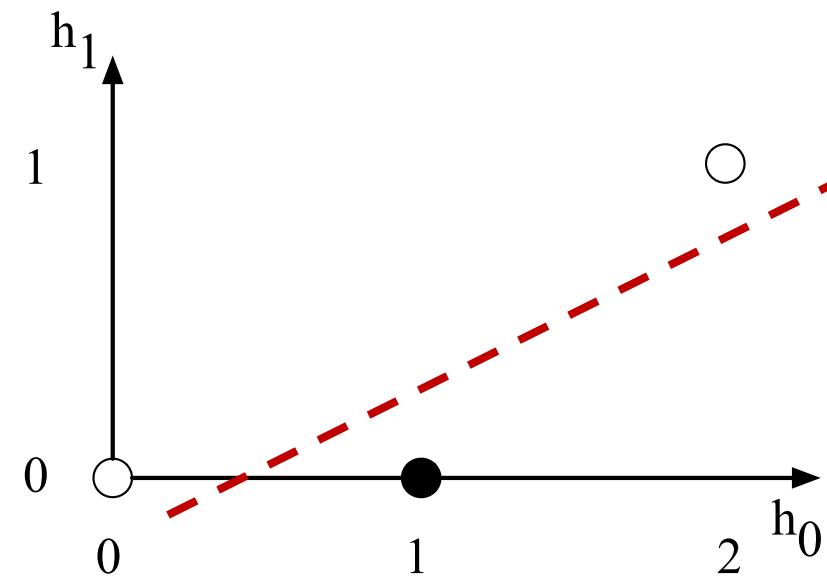
The XOR Solution



The XOR Solution



a) The original x space



b) The new h space