COSC343: Artificial Intelligence

Lecture 9 : Perceptrons

Lech Szymanski

Dept. of Computer Science, University of Otago

In today's lecture

- A bit of background about biological neurons
- Perceptron – a computation model of a neuron
- Perceptron learning

Bottom-up and top-down approaches to AI

A top-down approach:

- What are the tasks humans can perform?
- What is a good way of getting a computer to perform the same tasks?

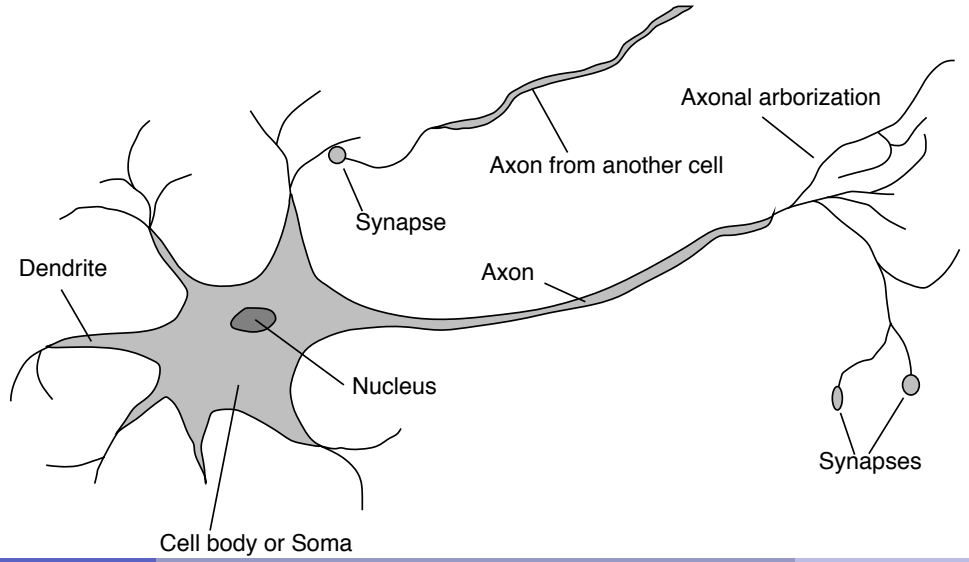
A bottom-up approach:

- What hardware do humans use for solving tasks?
- If we recreated this hardware, what sorts of tasks can we solve?

The brain's hardware

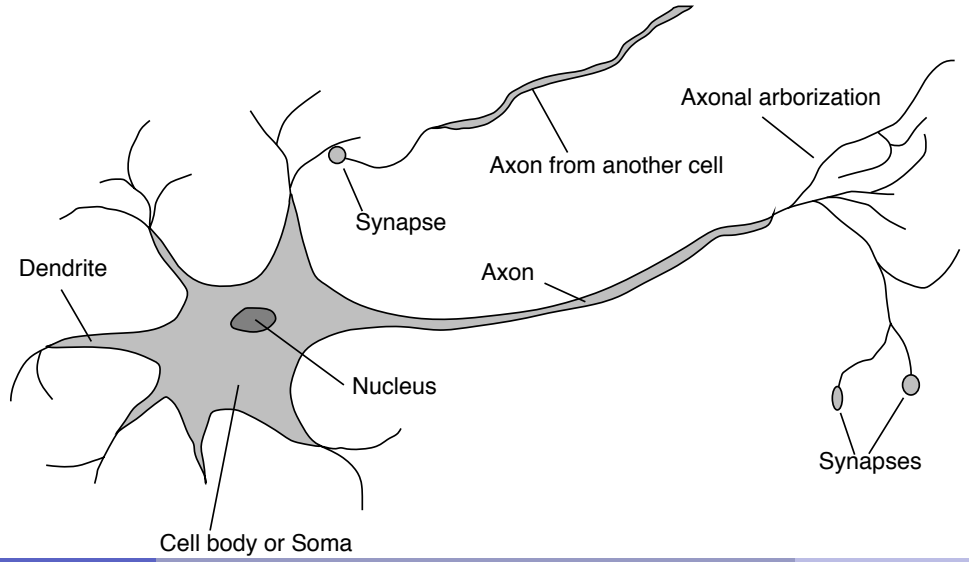
Computation in a human being is implemented in the brain.

- The brain is a network of cells called **neurons**.
- Neurons are connected to one another via **synapses**.
- There are around 10^{11} neurons, of many different types
- There are around 10^{14} synapses



Neural signals

A neuron has a number of **dendrites**, which gather input signals from other neurons. It has a single **axon**, which sends output signals to other neurons.

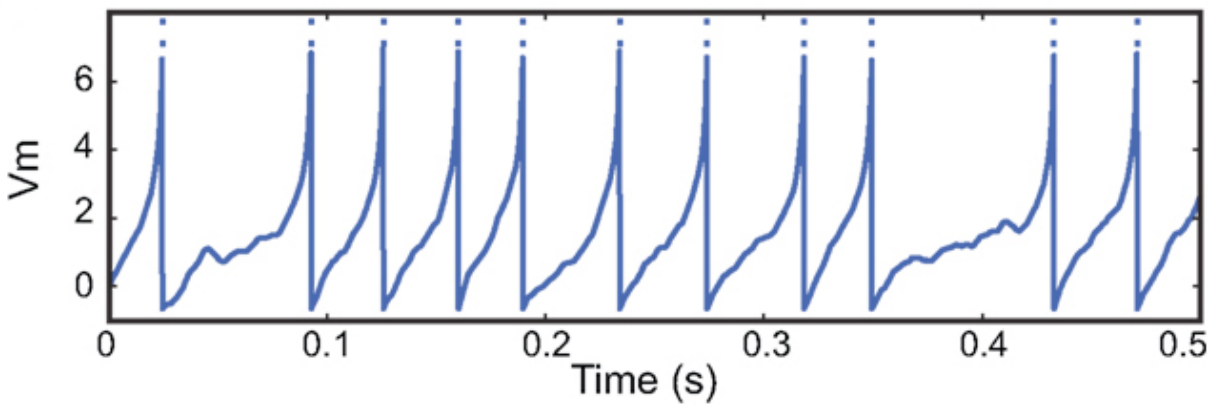


Neural signals

A neuron has a number of **dendrites**, which gather input signals from other neurons. It has a single **axon**, which sends output signals to other neurons.

The output of each neuron is a **spike train** of discrete pulses.

- Each pulse has the same amplitude; what varies is their frequency.



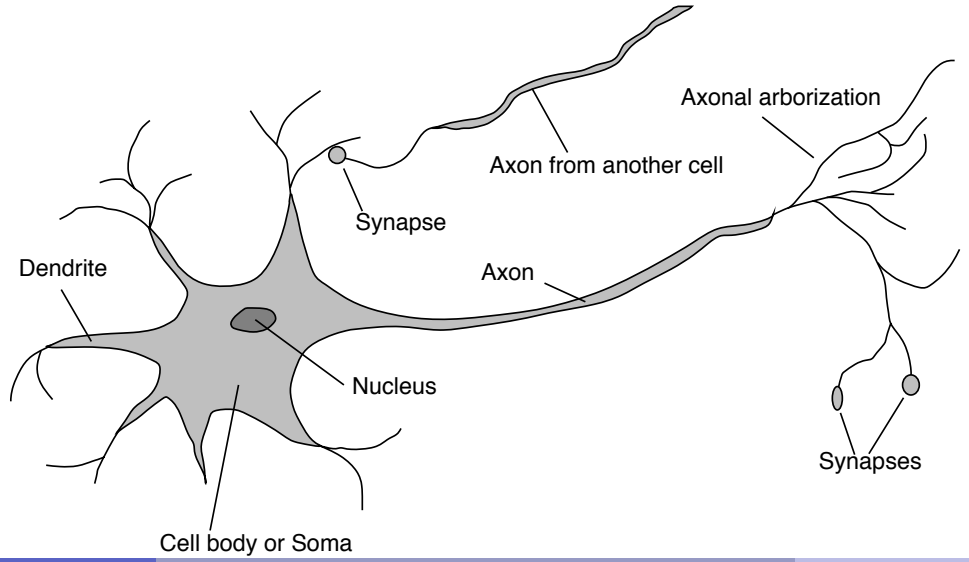
The computation performed by a neuron

In computational models, we can use real numbers to model spike rates.

- There's a *maximum firing* rate for neurons, so we typically 'squash' the number between 0 and 1.

A neuron 'accumulates' the spikes it receives: when the total exceeds a certain threshold, it *generates* a spike.

- Modelled as real numbers: the rate of output spikes ($0 \leq S \leq 1$) is a function of the sum of the rates of the input spikes



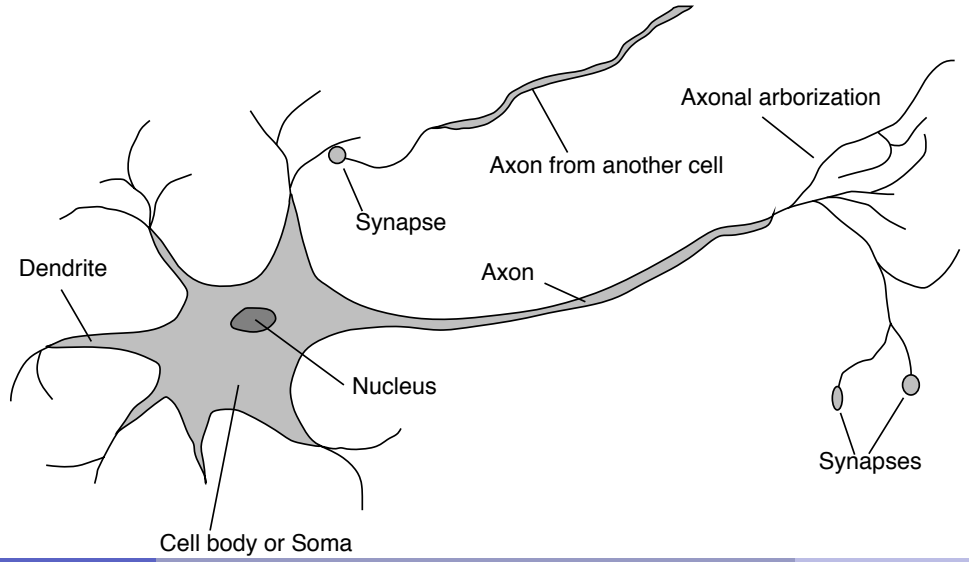
The computation performed by a neuron

A neuron's input synapses can have different **degrees of efficiency**.

- So the neuron computes a *weighted* sum of its inputs.

Also: some synapses are **inhibitory**.

- So some weights in the weighted sum can be *negative.*

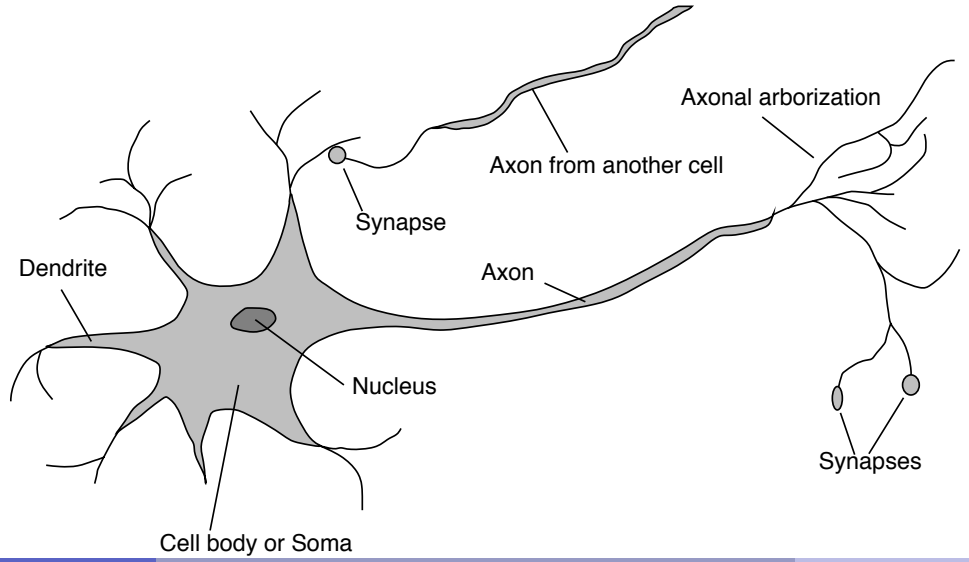


Neural vs. digital computation

The time taken for a single neuron to compute a response from its inputs is around 1-10ms.

- I.e. it's *slow*!

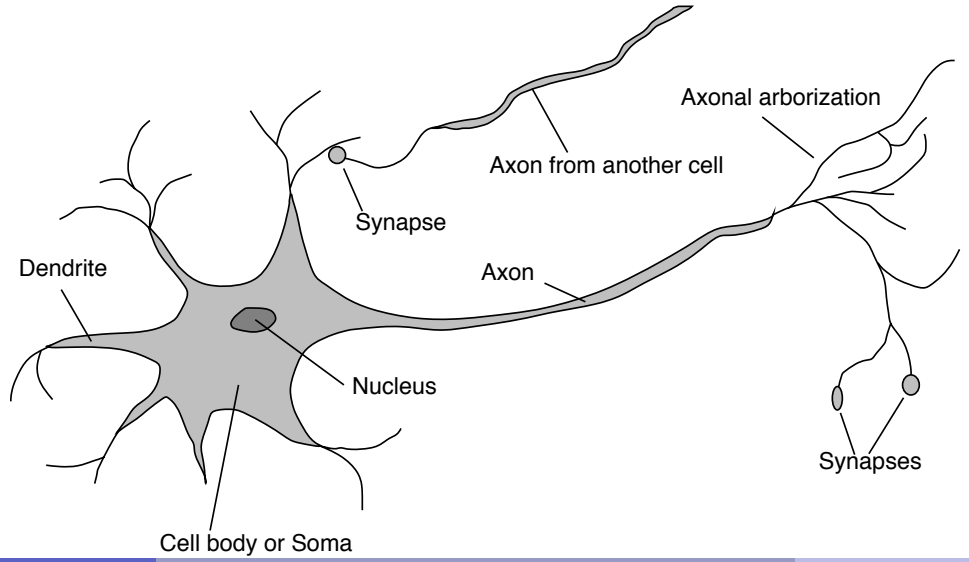
The power of a neural network is due to **parallelism**: it can perform many simple computations *simultaneously.*



Plasticity

The weights of many synapses in the brain are adjustable, and change as a function of the input signals they receive.

So networks can generate *responses* to inputs, but also *adapt* themselves to these inputs.



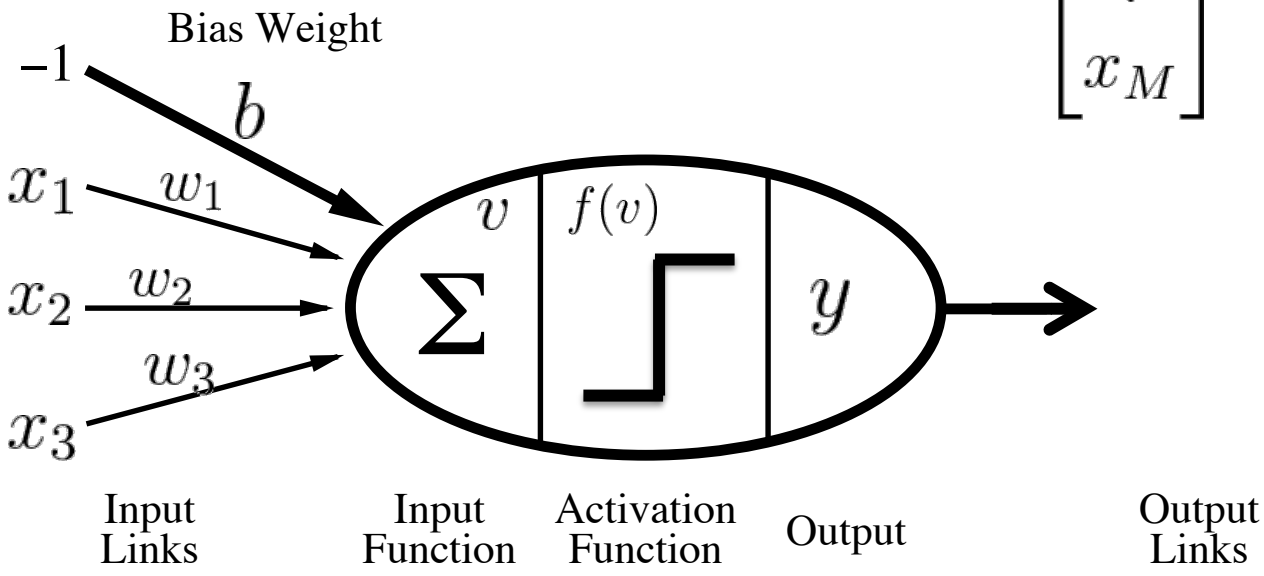
A simple model of a neuron

A simple model of a neuron is a **McCulloch-Pitts unit**.

$$v = \mathbf{w}^T\mathbf{x} - b$$

$$\mathbf{w}^T = \begin{bmatrix} w_1 & \dots & w_M \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}$$



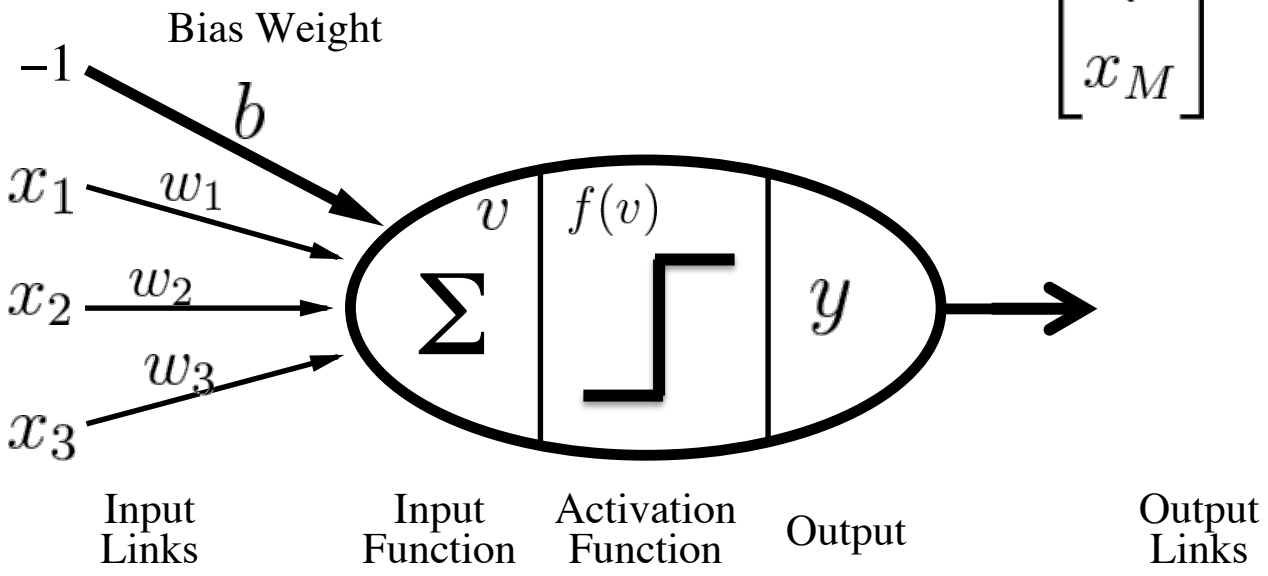
A simple model of a neuron

A simple model of a neuron is a **McCulloch-Pitts unit**.

$$y = f(\mathbf{w}^T\mathbf{x} - b)$$

$$\mathbf{w}^T = \begin{bmatrix} w_1 & \dots & w_M \end{bmatrix}$$

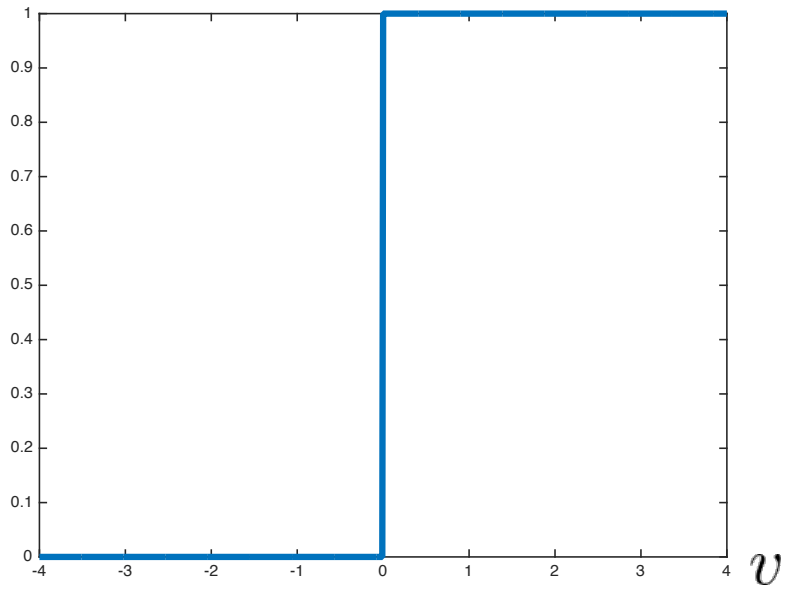
$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix}$$



Activation function

Hard limiting function

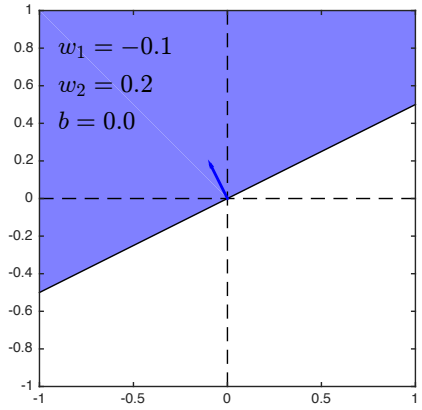
$$f_{\text{hardlim}}(v) = \begin{cases} 1 & v \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

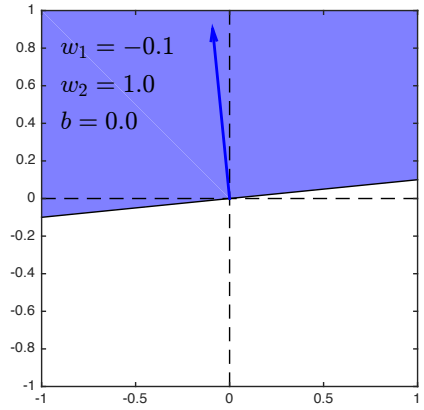


Geometric interpretation

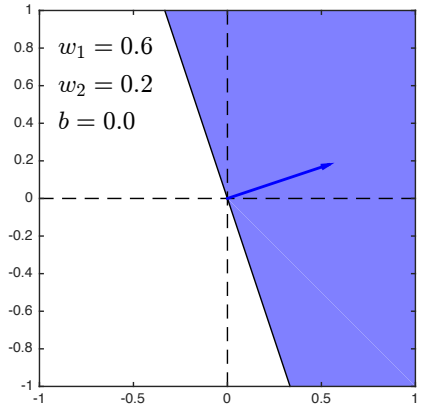
Let's

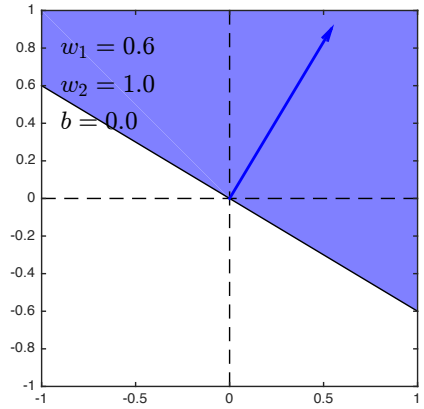
- Take a problem with a **2-D** input
- Set bias to zero
- **Change the weight vector values**





- Weight vector specifies the orientation of a separating **line**
  - Input from one side of the line will produce output value of 1
  - Input from the other side of the line will produce output value of 0

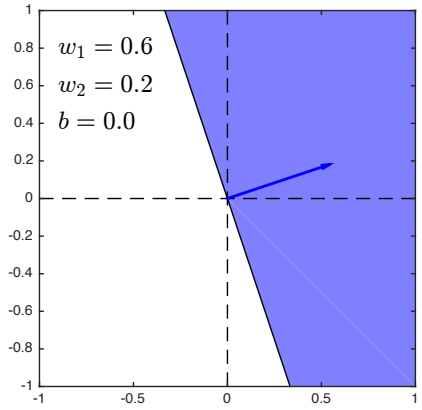


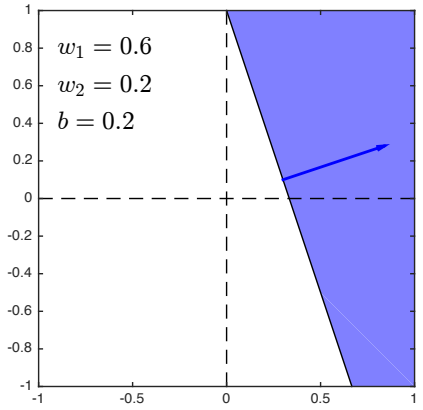


Geometric interpretation

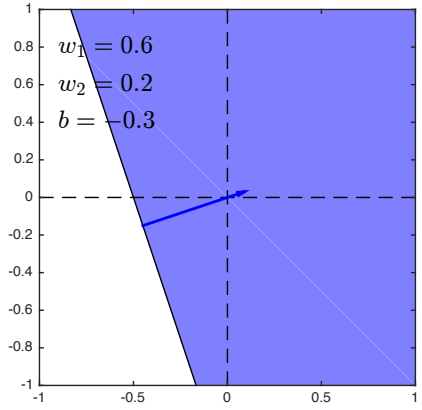
Let's

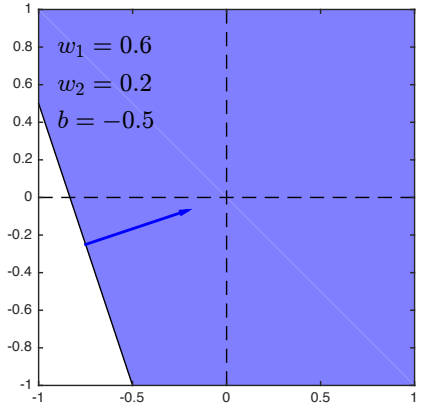
- Take a problem with a **2-D** input
- Fix the weight vector values
- **Change the bias value**





- Bias vector influences the distance of the separating **line** from the origin
  - Input from one side of the line will produce output value of 1
  - Input from the other side of the line will produce output value of 0



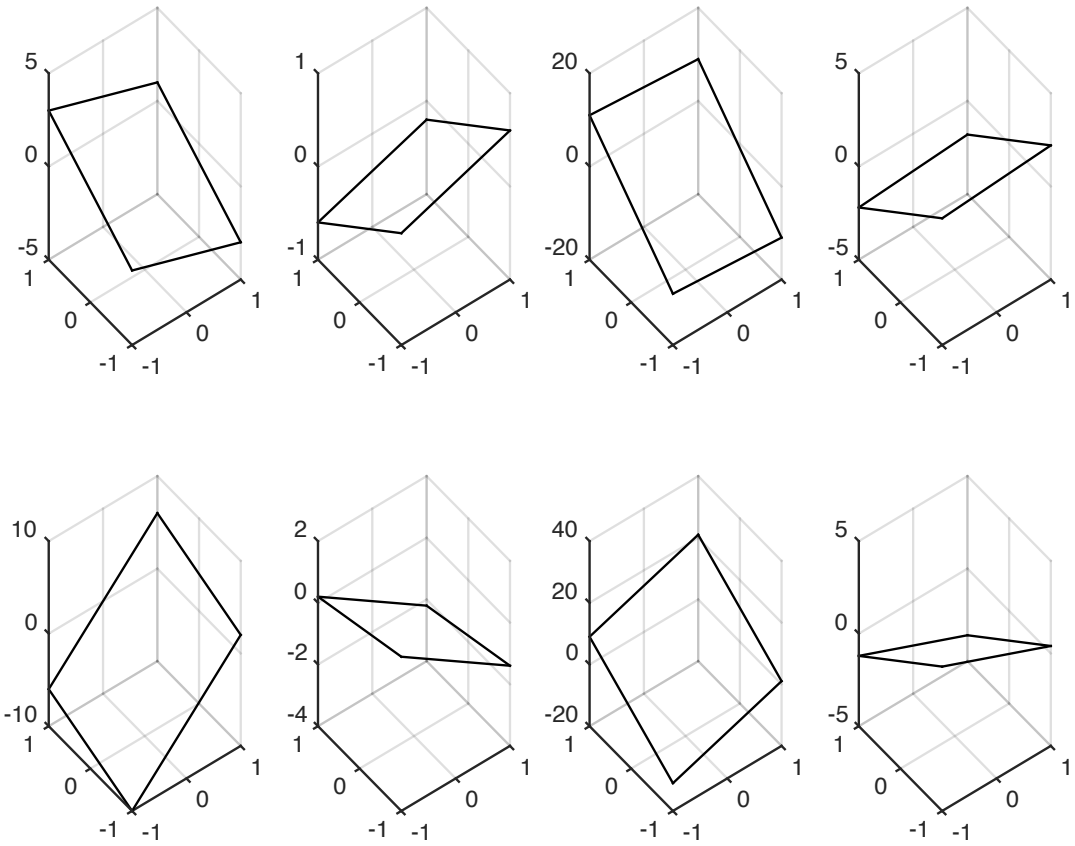


Geometric interpretation

Let's

- Take a problem with a **3-D** input

- The weight vector + bias define a **plane** that divdes space into two half-spaces
  - Input from one half-space produces output 1
  - Input from the other half space produces output 0

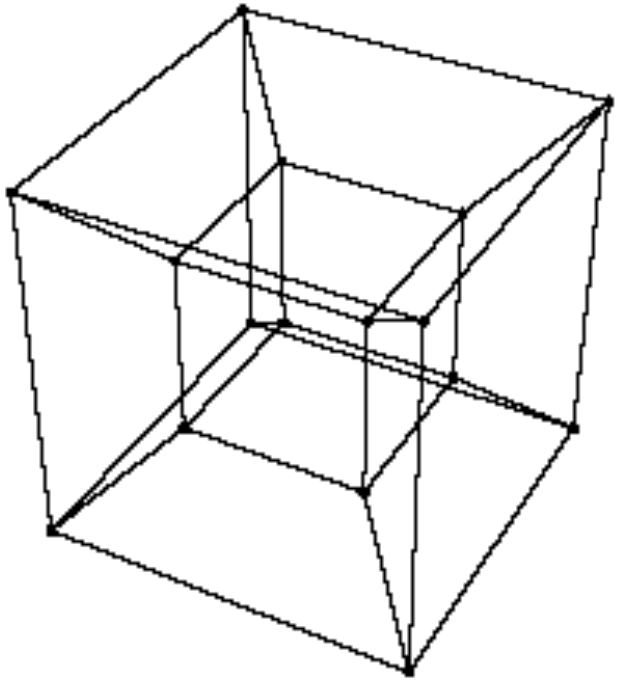


Geometric interpretation

Let's

- Take a problem with **M-D** input

- The weight vector + bias define a **hyperplane** that divades space into two half-spaces
  - Input from one half-space produces output 1
  - Input from the other half space produces output 0



Perceptron modelling

Given a data sample $\{(\mathbf{x}_1, \tilde{y}_1), \ldots, (\mathbf{x}_N, \tilde{y}_N)\}$ where input $\mathbf{x}_i = \begin{bmatrix} x_{1i} & \ldots & x_{Mi}^T \end{bmatrix}$ is an M dimensional vector and the desired output takes a value of 0 or 1, $\tilde{y}_i \in \{0, 1\}$, the hypothesis function is:

$$y_i = f_{\text{hardlim}}(\mathbf{w}^T \mathbf{x}_i - b)$$, where

the weights are an M dimensional vector $\mathbf{w}^T = \begin{bmatrix} w_1 & \ldots & w_M \end{bmatrix}$ and b is the bias value.

Perceptron learning

For each training example from the training dataset $\{(\mathbf{x}_1, \tilde{y}_1), \ldots, (\mathbf{x}_N, \tilde{y}_N)\}$, compute the hypothesis error, which subtracts perceptron output from the desired output:

$$y_i = f_{\text{hardlim}}(\mathbf{w}^T \mathbf{x}_i - b)$$

$$e_i = \tilde{y}_i - y_i$$

Note the order of subtraction here versus the definition of error in the previous lectures.

Perceptron learning rule

$$y_i = f_{\text{hardlim}}(\mathbf{w}^T \mathbf{x}_i - b) \qquad e_i = \tilde{y}_i - y$$

- If error is positive, $e_i > 0$, perceptron output needs to be bigger:
  - The weights corresponding to positive inputs, $x_{ui} > 0$, should increase
  - The weights corresponding to negative inputs, $x_{ui} < 0$, should decrease
  - The bias should decrease

$$w_j = w_j + \alpha x_{ju} \qquad b_j = b_j - \alpha$$

- If error is positive, $e_i < 0$, perceptron output needs to be smaller:
  - The weights corresponding to positive inputs, $x_{ui} > 0$, should decrease
  - The weights corresponding to negative inputs, $x_{ui} < 0$, should increase
  - The bias should increase

$$w_j = w_j - \alpha x_{ju} \qquad b_j = b_j + \alpha$$

- If error is zero, $e_i = 0$, no changes should be made

Note that, since the desired output is always either 0 or 1, and the hardlimiting activity assures perceptron output is always either 0 or 1, then the error must be either -1, 0, or 1.

Perceptron learning rule

$$y_i = f_{\text{hardlim}}(\mathbf{w}^T \mathbf{x}_i - b) \qquad e_i = \tilde{y}_i - y$$

Guaranteed to converge given:

- The problem is linearly separable
- The choice of α is sufficiently small

$$w_j = w_j + \alpha e_i x_{ju}$$

$$b_j = b_j - \alpha e_i$$

Perceptron learning rule

$$y_i = f_{\text{hardlim}}(\mathbf{w}^T \mathbf{x}_i - b) \qquad e_i = \tilde{y}_i - y$$

Guaranteed to converge given:

- The problem is linearly separable
- The choice of α is sufficiently small

called often **delta rule**, because error was denoted symbol δ

$$w_j = w_j + \alpha e_i x_{ju}$$

$$b_j = b_j - \alpha e_i$$

Doesn't this update equation look familiar?

Training regimes

There are two ways of using the perceptron rule:

Online training

- We can make weight and bias changes after each training example

$$w_j = w_j + \alpha e_i x_{ju} \qquad b_j = b_j - \alpha e_i$$

Batch training

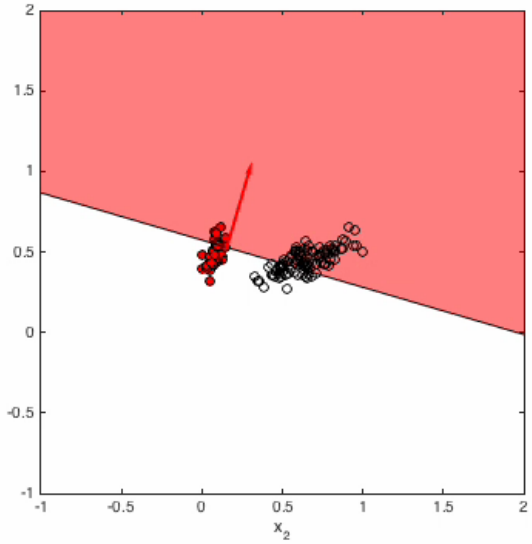
- We can remember the weight changes for each output for a full set of training examples, and then apply the average weight change

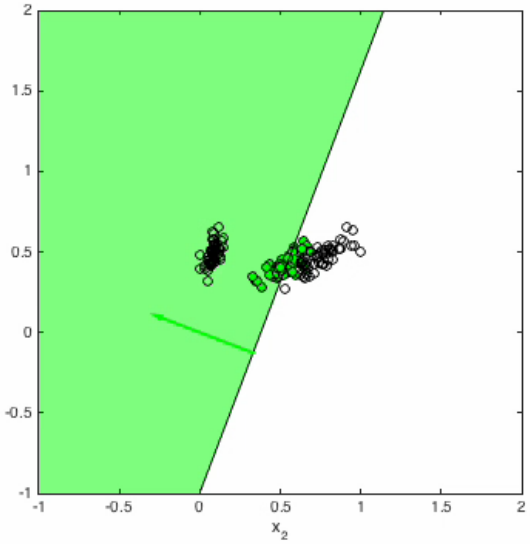
$$w_j = w_j + \frac{\alpha}{N} \sum_i e_i x_{ju} \qquad b_j = b_j - \frac{\alpha}{N} \sum_i e_i$$

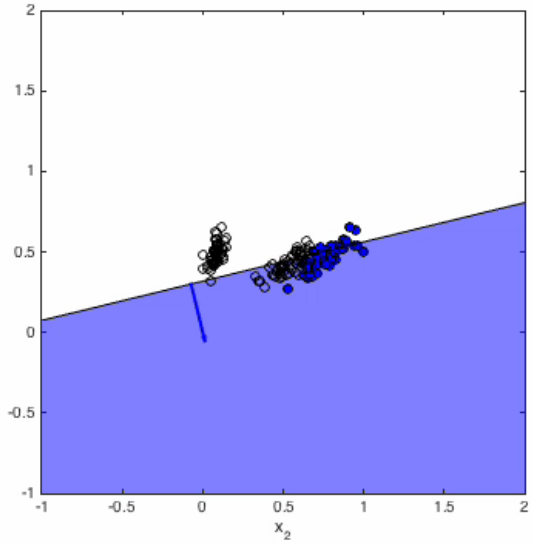
This is exactly like the steepest gradient descent update with a flipped sign (due to subtraction order in the definition of the error)

An example: iris dataset

- Classification task with three classes
- Three perceptrons, each tasked with distinguishing a given class from the others
  - $\tilde{y}_{ri} = 1$ when point is labelled red, zero otherwise
  - $\tilde{y}_{gi} = 1$ when point is labelled green, zero otherwise
  - $\tilde{y}_{bi} = 1$ when point is labelled blue, zero otherwise







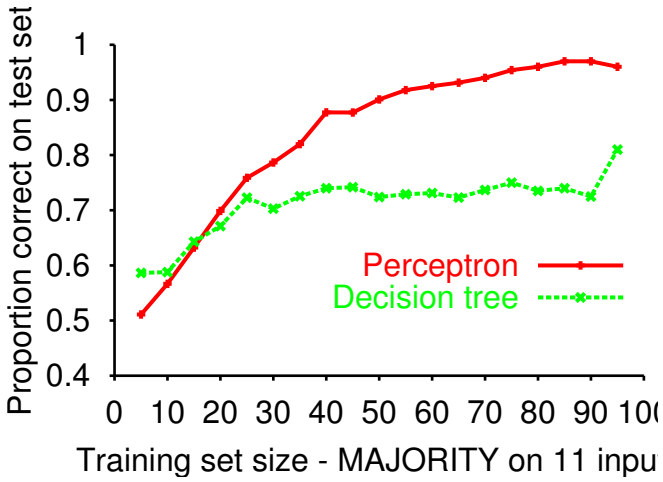
Perceptrons vs. decision trees

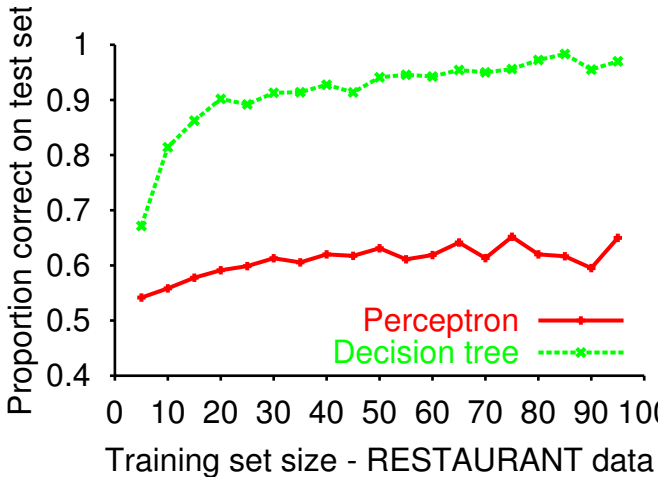
Some function are easy for perceptrons, and hard for decision trees.

- E.g. Majority (which takes M Boolean inputs, and returns T if more than half of these input are T)

Others are the other way around

- E.g. Russel's 'restaurant' function (which is not linearly separable





Summary and reading

- Perceptron – a linear model with a simple learning rule
- Though a revolutionary concept back in the 50's, actually not all that powerful, except...

...it leads to artificial neural networks!!!

Reading for the lecture: AIMA Chapter 18 Sections 7.1,7.2

Reading for next lecture: AIMA Chapter 18 Section 7.3,7.4