

COMP9414: Artificial Intelligence

Lecture 9a: Neural Networks

Wayne Wobcke

e-mail:w.wobcke@unsw.edu.au

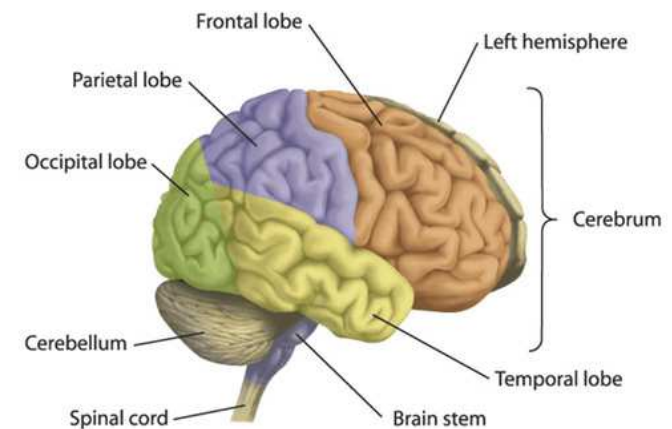
This Lecture

- Neurons – Biological and Artificial
- Perceptron Learning
- Linear Separability
- Multi-Layer Networks
- Backpropagation
- Application - ALVINN

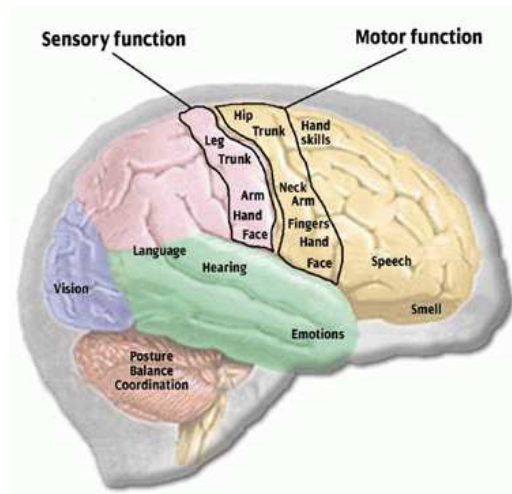
Sub-Symbolic Processing



Brain Regions



Brain Functions



Biological Neurons

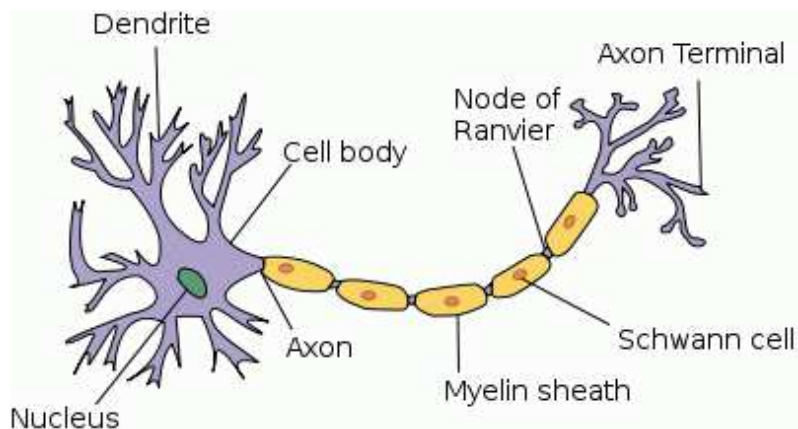
The brain is made up of **neurons** (nerve cells) which have

- A cell body (soma)
- **Dendrites** (inputs)
- An **axon** (outputs)
- **Synapses** (connections between cells)

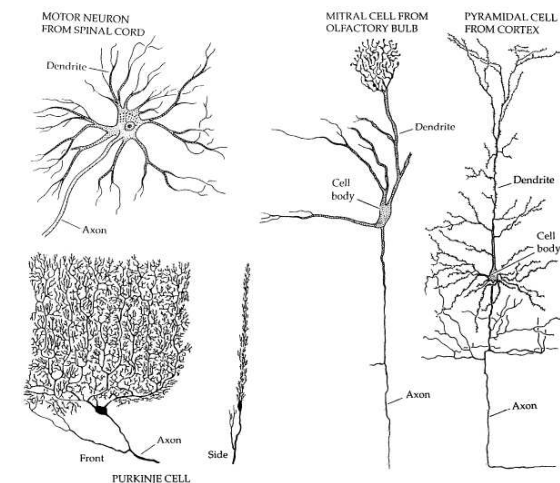
Synapses can be **excitatory** or **inhibitory** and may change over time.

When the inputs reach some threshold, an **action potential** (electrical pulse) is sent along the axon to the outputs

Structure of a Typical Neuron



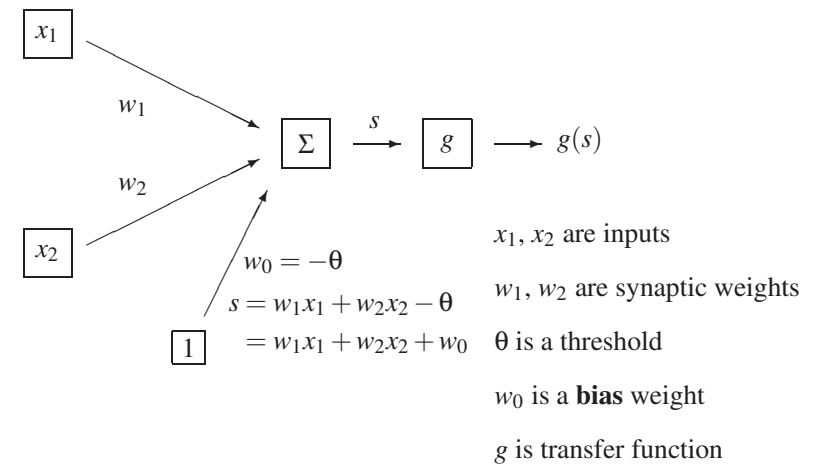
Variety of Neuron Types



The Big Picture

- Human brain has 100 billion neurons with an average of 10,000 synapses each
- Latency is about 3-6 milliseconds
- Therefore, at most a few hundred “steps” in any mental computation, but **massively parallel**

McCulloch & Pitts Model of a Single Neuron



Artificial Neural Networks

(Artificial) Neural Networks are made up of nodes which have

- Input edges, each with some **weight**
- Output edges (with **weights**)
- An **activation level** (a function of the inputs)

Weights can be positive or negative and may change over time (learning)

The **input function** is the weighted sum of the activation levels of inputs

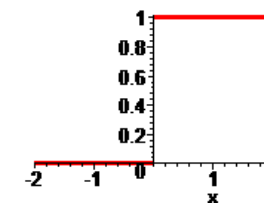
The activation level is a non-linear **transfer function** g of this input

$$\text{activation}_i = g(s_i) = g\left(\sum_j w_{ij}x_j\right)$$

Some nodes are inputs (sensing), some are outputs (action)

Transfer Function

Originally, a (discontinuous) step function



$$g(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{if } s < 0 \end{cases}$$

Later, other transfer functions which are continuous and smooth

Linear Separability

Question: What kind of functions can a perceptron compute?

Answer: Linearly separable functions

Examples

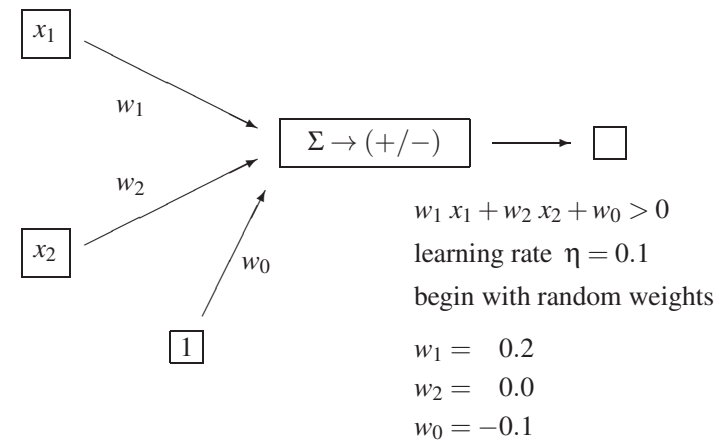
AND $w_1 = w_2 = 1.0, w_0 = -1.5$

OR $w_1 = w_2 = 1.0, w_0 = -0.5$

NOR $w_1 = w_2 = -1.0, w_0 = 0.5$

Question: How can we train a perceptron net to **learn** a new function?

Perceptron Learning Example



Perceptron Learning Rule

Adjust the weights as each input is presented

Recall $s = w_1 x_1 + w_2 x_2 + w_0$

if $g(s) = 0$ but should be 1, if $g(s) = 1$ but should be 0,

$$w_k \leftarrow w_k + \eta x_k$$

$$w_k \leftarrow w_k - \eta x_k$$

$$w_0 \leftarrow w_0 + \eta$$

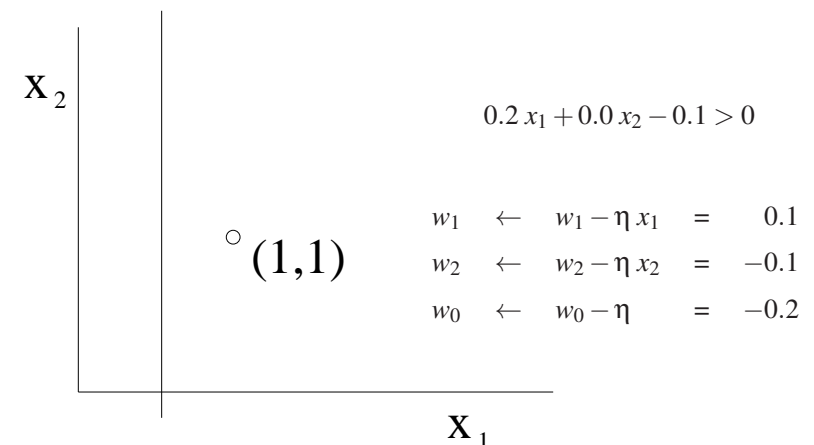
$$w_0 \leftarrow w_0 - \eta$$

$$\text{so } s \leftarrow s + \eta (1 + \sum_k x_k^2) \quad \text{so } s \leftarrow s - \eta (1 + \sum_k x_k^2)$$

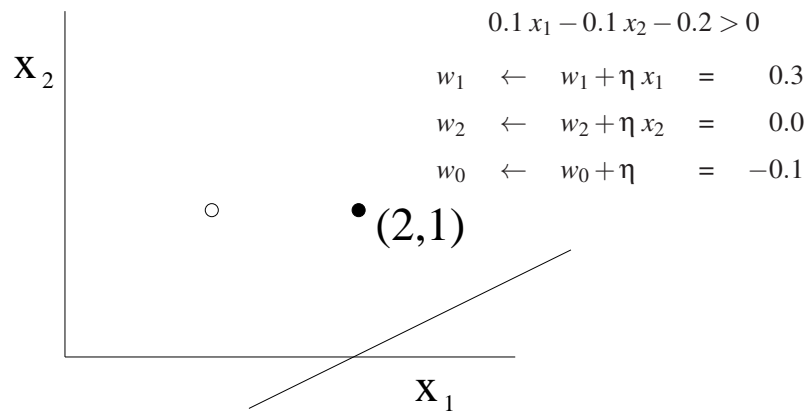
otherwise weights are unchanged ($\eta > 0$ is called the **learning rate**)

Theorem: This will eventually learn to classify the data correctly, as long as they are **linearly separable**.

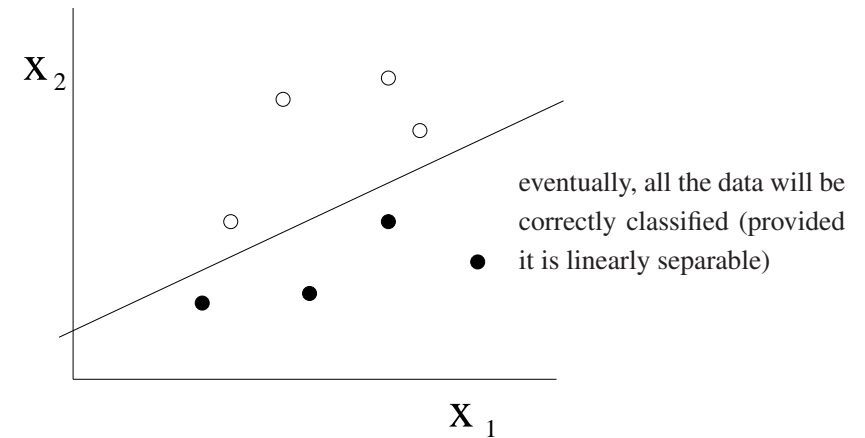
Training Step 1



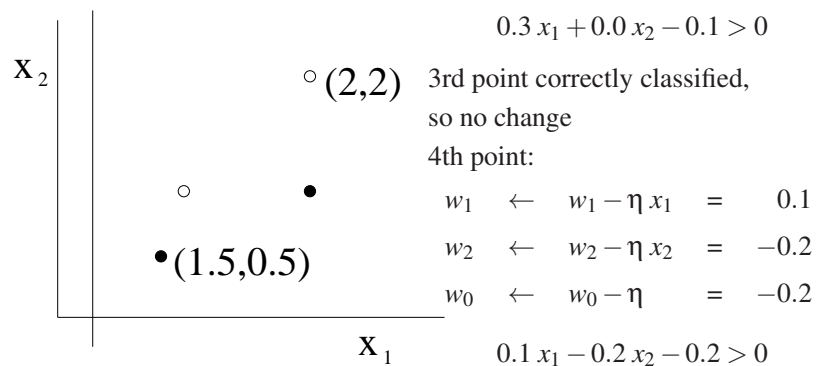
Training Step 2



Final Outcome

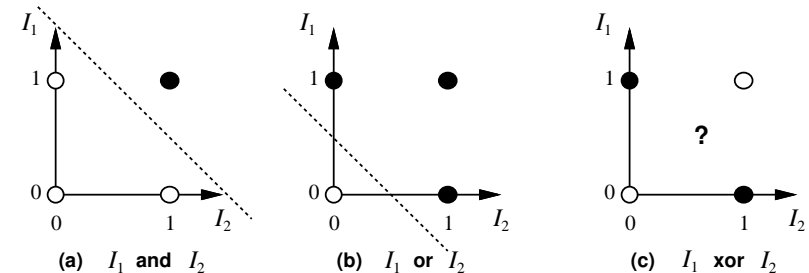


Training Step 3



Limitations

Problem: Many useful functions are not linearly separable (e.g. XOR)

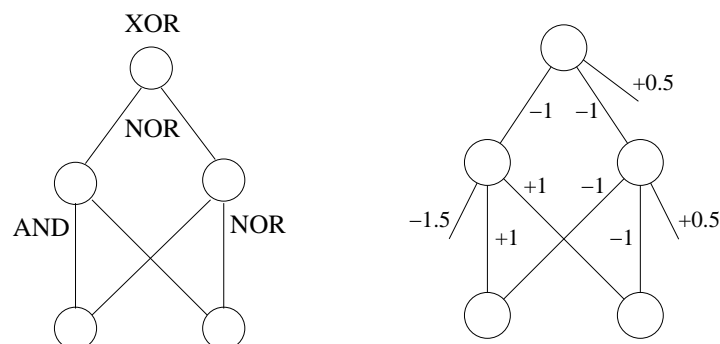


Possible solution

$x_1 \text{ XOR } x_2$ can be written as $(x_1 \text{ AND } x_2) \text{ NOR } (x_1 \text{ NOR } x_2)$

Recall that AND, OR and NOR can be implemented by perceptrons

Multi-Layer Neural Networks



Question: Given an explicit logical function, we can design a multi-layer neural network by hand to compute that function – but if we are just given a set of training data, can we train a multi-layer network to fit this data?

Training as Cost Minimization

Define an **error function** E to be (half) the sum over all input patterns of the square of the difference between actual output and desired output

$$E = \frac{1}{2} \sum (z - t)^2$$

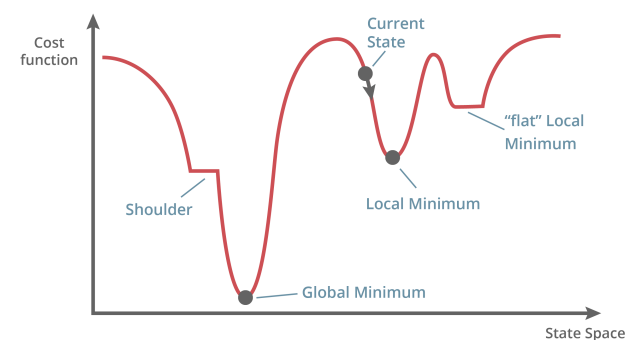
If we think of E as “height”, this gives an error “landscape” on the weight space. The aim is to find a set of weights for which E is very low.

Historical Context

In 1969, Minsky and Papert published a book highlighting the limitations of Perceptrons, and lobbied various funding agencies to redirect funding away from neural network research, preferring instead logic-based methods such as expert systems.

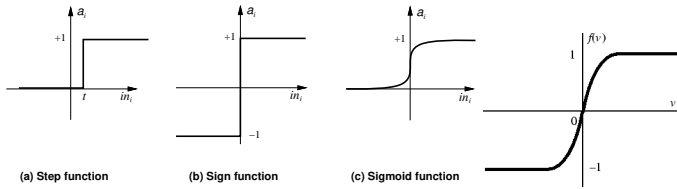
It was known as far back as the 1960s that any given logical function could be implemented in a 2-layer neural network with step function activations. But, the the question of how to learn the weights of a multi-layer neural network based on training examples remained an open problem. The solution, which we describe in the next section, was found in 1976 by Paul Werbos, but did not become widely known until it was rediscovered in 1986 by Rumelhart, Hinton and Williams.

Local Search in Weight Space



Problem: Because of the step function, the landscape will not be smooth but will instead consist almost entirely of flat local regions and “shoulders”, with occasional discontinuous jumps

Key Idea



Replace the (discontinuous) step function with a differentiable function, such as the sigmoid

$$g(s) = \frac{1}{1 + e^{-s}}$$

or hyperbolic tangent

$$g(s) = \tanh(s) = \frac{e^s - e^{-s}}{e^s + e^{-s}} = 2\left(\frac{1}{1 + e^{-2s}}\right) - 1$$

Chain Rule

If

$$y = y(u)$$

$$u = u(x)$$

Then

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

This principle can be used to compute the partial derivatives in an efficient and localized manner. Note that the transfer function must be differentiable (usually sigmoid, or tanh).

$$\begin{aligned} \text{Note: if } z(s) &= \frac{1}{1 + e^{-s}} & z'(s) &= z(1 - z) \\ \text{if } z(s) &= \tanh(s) & z'(s) &= 1 - z^2 \end{aligned}$$

Gradient Descent

Recall that the **error function** E is (half) the sum over all input patterns of the square of the difference between actual output and desired output

$$E = \frac{1}{2} \sum (z - t)^2$$

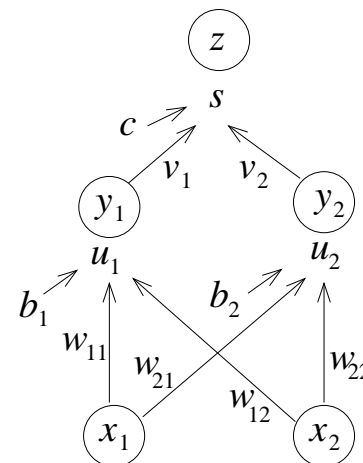
The aim is to find a set of weights for which E is very low.

If the functions are smooth, use multi-variable calculus to define how to adjust the weights so error moves in steepest downhill direction

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}$$

Parameter η is called the **learning rate**

Forward Pass



$$\begin{aligned} u_1 &= b_1 + w_{11}x_1 + w_{12}x_2 \\ y_1 &= g(u_1) \\ s &= c + v_1y_1 + v_2y_2 \\ z &= g(s) \\ E &= \frac{1}{2} \sum (z - t)^2 \end{aligned}$$

Backpropagation

Partial Derivatives

$$\frac{\partial E}{\partial z} = z - t$$

$$\frac{dz}{ds} = g'(s) = z(1 - z)$$

$$\frac{\partial s}{\partial y_1} = v_1$$

$$\frac{dy_1}{du_1} = y_1(1 - y_1)$$

Useful notation

$$\delta_{\text{out}} = \frac{\partial E}{\partial s} \quad \delta_1 = \frac{\partial E}{\partial u_1} \quad \delta_2 = \frac{\partial E}{\partial u_2}$$

Then

$$\delta_{\text{out}} = (z - t) z (1 - z)$$

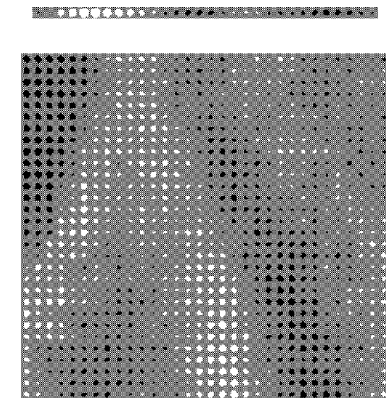
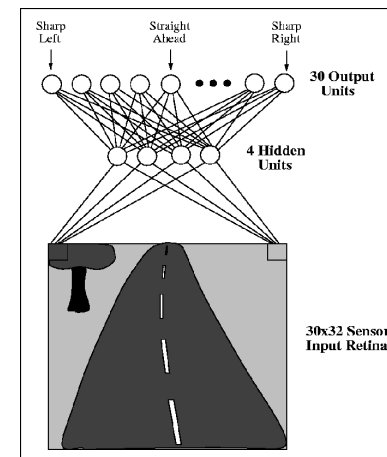
$$\frac{\partial E}{\partial v_1} = \delta_{\text{out}} y_1$$

$$\delta_1 = \delta_{\text{out}} v_1 y_1 (1 - y_1)$$

$$\frac{\partial E}{\partial w_{11}} = \delta_1 x_1$$

Partial derivatives can be calculated efficiently by backpropagating deltas through the network

ALVINN



Neural Network – Applications

- Autonomous Driving
- Game Playing
- Credit Card Fraud Detection
- Handwriting Recognition
- Financial Prediction

ALVINN

- Autonomous Land Vehicle In a Neural Network
- Later version included a sonar range finder
 - ▶ 8×32 range finder input retina
 - ▶ 29 hidden units
 - ▶ 45 output units
- Supervised Learning, from human actions (Behavioural Cloning)
 - ▶ Additional “transformed” training items to cover emergency situations
- Drove autonomously from coast to coast across US

Training Tips

- Re-scale inputs and outputs to be in the range 0 to 1 or -1 to 1
- Initialize weights to very small random values
- On-line or batch learning
- Three different ways to prevent overfitting:
 - ▶ Limit the number of hidden nodes or connections
 - ▶ Limit the training time, using a validation set
 - ▶ Weight decay
- Adjust learning rate (and momentum) to suit the particular task

Summary

- Neural networks are biologically inspired
- Multi-layer networks can learn non-linearly separable functions
- Backpropagation is effective and widely used