Computational Methods in Astrophysics

Dr Rob Thacker (AT319E)

thacker@ap.smu.ca

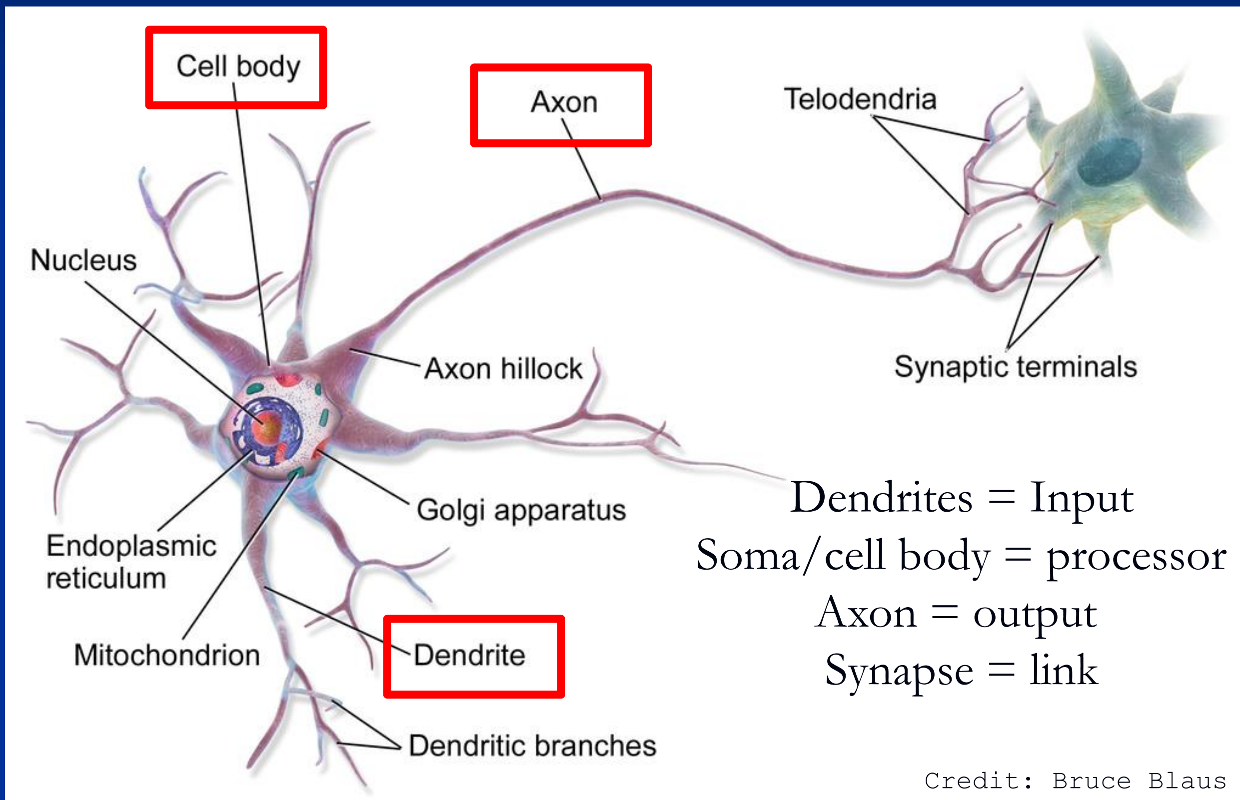
ANNs: Motivating factors

- Many problems have no obvious algorithmic solutions
  - e.g. face recognition, handwriting
  - Even if there were an algorithm it could still be exceptionally difficult to code
- The solutions derived in neural networks are often vastly different to hand-written serial approaches
  - Especially so for highly complicated nets
  - Indeed we frequently simply don't know what the algorithm is
- We anticipate neural networks will be able to become good at processes brains are good at
  - But that also means they will be poor at direct computations like arithmetic, but that doesn't matter…

History: Artificial Neural Networks

- Artificial models of of biological neurons conceived in 1940s – McCulloch & Pitts propose their model
- 1949 Hebb's proposes theory of learning in terms of synapse plasticity
- 1957 Perceptron algorithm is developed by Rosenblatt at Cornell
- 70s – becomes clear that there are fundamental limitations to "single layer" perceptrons
- 80s – New "back propagation" learning methods for multi-layer perceptrons discovered, field takes off again
- 2000s – SVM etc reduce popularity of ANN, but advent of deep learning again renews interest

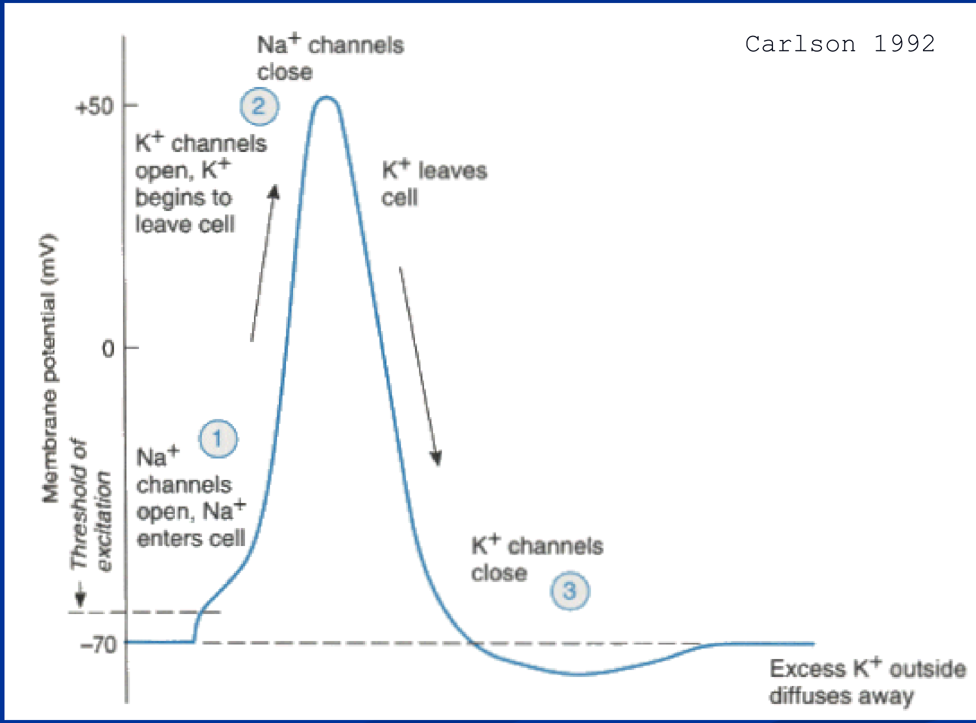
Biological neurons



- 3 key parts – body, dendrites and axon
- Dendrites branch and thin, axon usually remains thick
- Dendrites or cell body receive signals (via synapses)
- Axon transmits to other neurons (via synapses)

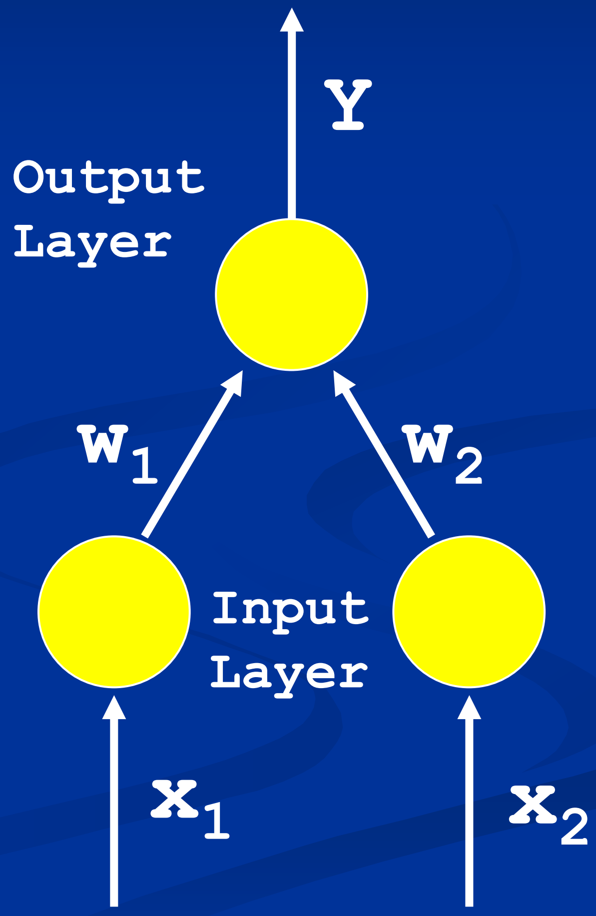
Biological neurons & brains

- Human brains have 80-90 billion neurons
  - 19-23 billion in cerebral cortex
- Each neuron may be connected to up to 10,000 others
- Typical frequency of firing is around 10 Hz, can go as high as 100 Hz
- Neurons maintain a –ve bias voltage (-70 mV) in their resting state giving them a membrane potential
  - Depolarization (to +ve bias) occurs via influx of Na+ ions
  - K+ ions then flow out to repolarize
    - This change in potential is propagated down the axon via the same depolarization/repolarization
    - Takes 0.002 seconds…



What is an ANN?

- Definition due to Hecht-Nielsen:
  - …a computing system made up of a number of simple, highly connected processing elements, which process information by their dynamic state response to external inputs…
- Needs to be input layer and output layer
  - Can also be hidden layers
- Weights can be trained by minimizing error
- Can be hardware, or algorithmic
- Largest simulation yet performed 160 billion parameters



Why are ANNs so popular? Drawbacks?

- As long as a problem can be broken down into an appropriate set of numerical inputs and outputs it should be possible to do some kind of training
  - But to do this optimally requires carefully processing the data set first (avoiding redundancies, statistical outliers etc)
- ANN are good for problems with a large number of degrees of freedom
  - Images, speech etc.
- Drawback: if sufficiently complex, they are a black box
  - Getting a logical understanding of the classification method is frequently very difficult although "rule extraction" algorithms do exist
- Also difficult to "fine tune"

Brains vs computers

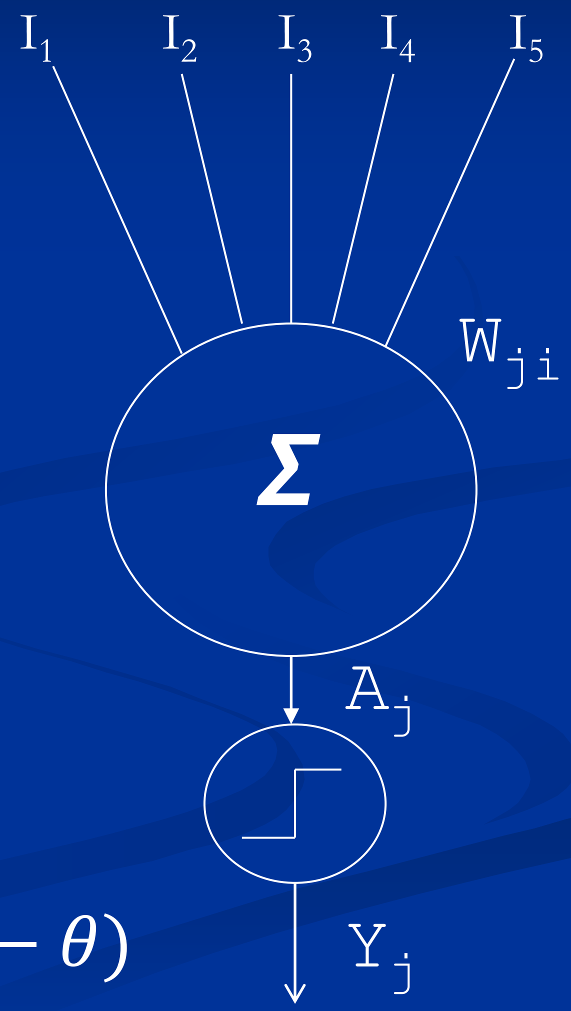
- Brains and computers process in remarkably different ways
- 10^9 transistors in single CPU – but 10^{14} in biggest supercomputers
- Compare that to 10^{14}-10^{15} synapses, 10^{11} neurons in brain
  - Neurons are more directly connected (1 to 10,000) than CPUs but this distinction is moot IMHO
- Brain is remarkably fault tolerant – computers fail quite easily (try spilling coffee!)
- Brains process in a strongly parallel way – computers tend to be serial (with exceptions for parallel algorithms)
- Adaptivity: Brains can learn – not even clear what it means for a serial computer to do that
- Evolution: Brains took millions of years to reach this point of evolution, computers a few decades

Deep Learning

- ANNs have consistently increased in complexity
- Deep learning is a logical extension of this trend, and corresponds to networks with five or more layers
  - Layers are trained in response to one another
- 2012 paper by Hinton et al (U. Toronto CS) gave breakthrough performance on a number of data sets
- The key promise of the method is in unsupervised learning
  - The notable ability of these system is to uncover difficult to detect structure and commonality
  - Humans learn in a similar way, although we do need some elements of supervised learning
- Notice that it is quite distinct from SVMs which try to factor out complexity with a single transformation

McCulloch-Pitts

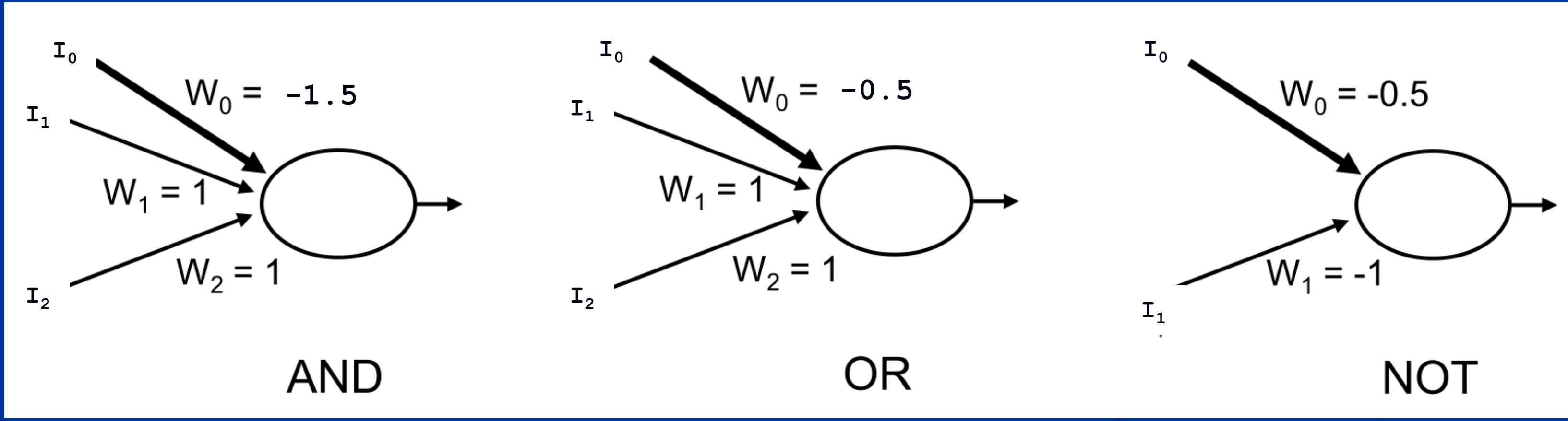
- First mathematical model of neuron
- Output Y_j is fire (1) or not fire (0)
- Inputs can be weighted excitatory (>0) in inhibitory (<0)
  - Traditionally exciting weights are labelled a, inhibitory weights labelled b (but watch for different notations)
  - Neuron fires if sum of weights*inputs exceeds threshold $\boldsymbol{\theta}$ - the activation threshold
- Output Y_j goes to another system



$$Y_j = H\left(\sum_{i=1}^{n} W_{ji} I_i - \theta\right)$$

McCulloch-Pitts – logic models

- The MP model can implement AND, OR and NOT gates

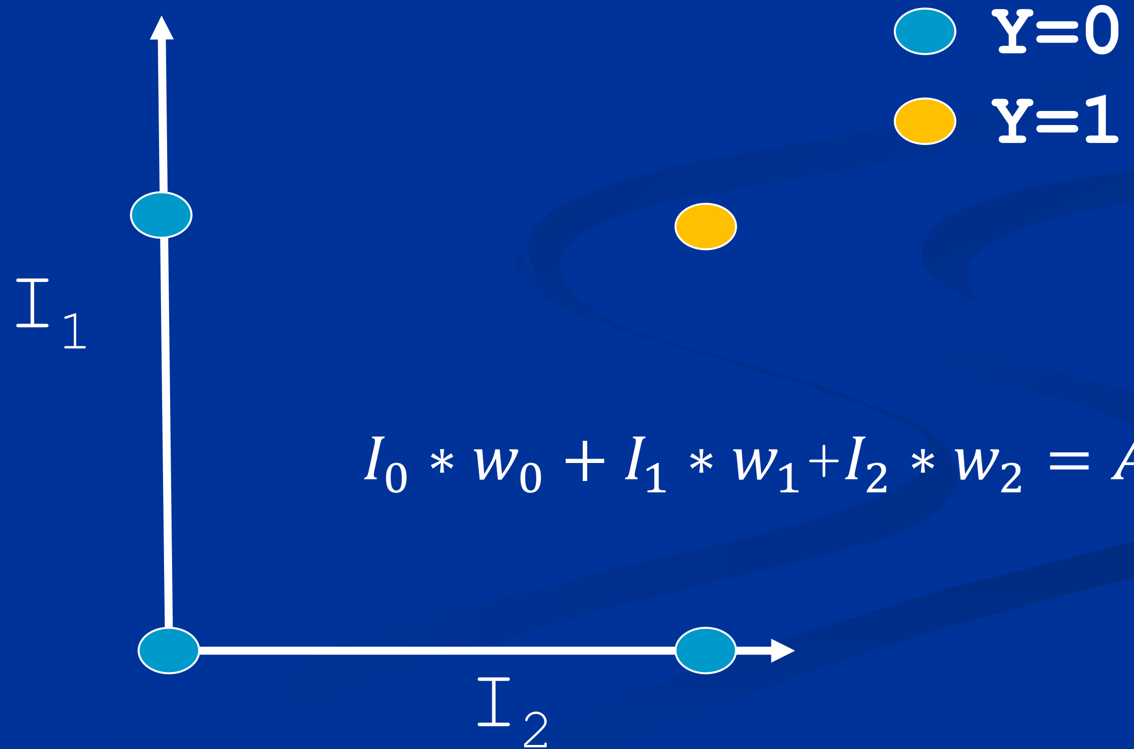


- Traditional to add extra input at +1 with specific weight
- Let's look at this in a little more detail

MP models to the Perceptron – things to consider

- What kind of data can they discriminate?
- Consider the pattern space of inputs for an AND gate

I_1	I_2	Y
0	0	0
0	1	0
1	0	0
1	1	1



$$I_0 * w_0 + I_1 * w_1 + I_2 * w_2 = A_j$$

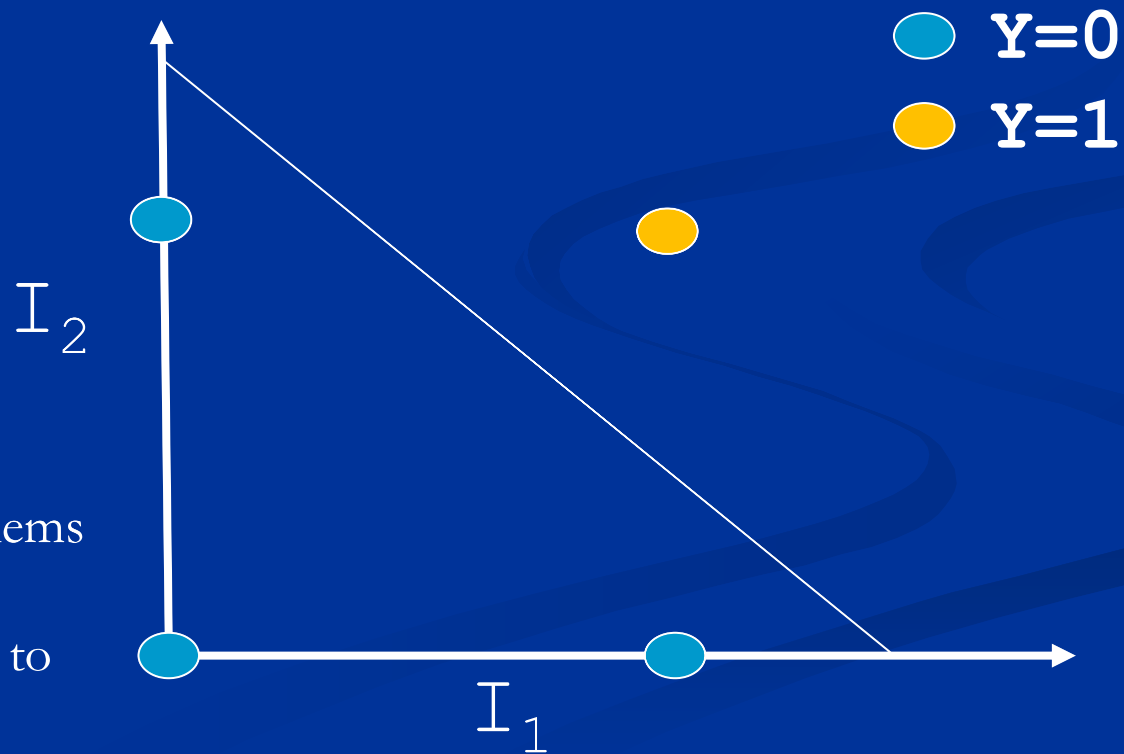
MP models to the Perceptron – things to consider

- Transition point at $I_1 + I_2 = 1.5 \Rightarrow I_2 = 1.5 - I_1$
- Decision boundary -1 slope, intercept 1.5

I_1	I_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Linearly separable problems will work

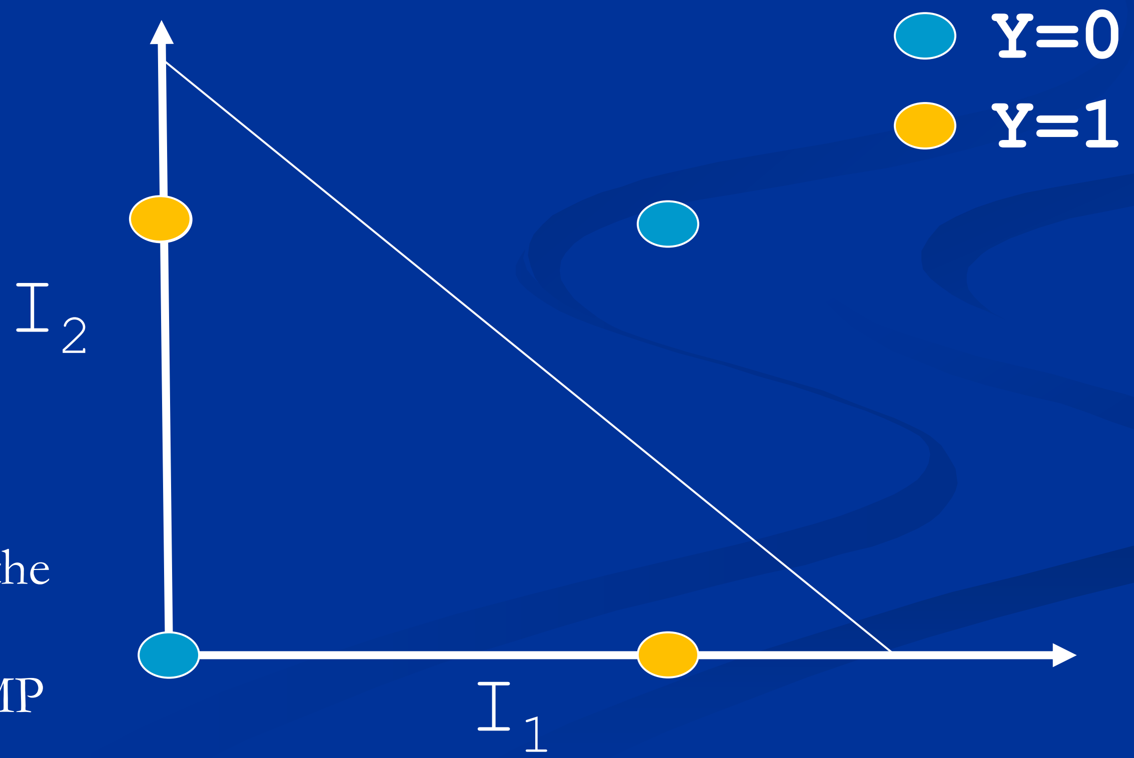
More inputs generalizes to higher dimensions



A simple non-linearly-separable data set: XOR

- Consider the exclusive OR

I_1	I_2	Y
0	0	0
0	1	1
1	0	1
1	1	0



No single line can split the data – XOR can't be formed by single layer MP neuron

Perceptrons

- Perceptrons (1958, Rosenblatt) generalized MP model
- Weights can have different values
- Generalized output to go from [-1,1] but this isn't a big difference
  - We will work with [-1,1] from now on
  - Note there are also different transfer functions that can be chosen – more later
- But the key difference was adding in the concept of learning via a learning rule

Perceptron – Learning algorithm

- This will be a supervised learning situation
- For the dth case we have a true value of t^d, while the system output o^d (we called it Y before)
  - Define error = $t^d - o^d$
- In response to an error what do we want?
  - Suppose $t^d=1$ $o^d=-1$, so error is +ve
  - The output is too low => increase sum to increase output
    - If input I_i is +ve increase W_{ji}
    - If input I_i is –ve decrease W_{ji}
  - Suppose $t^d=-1$ $o^d=1$, so error is –ve
  - The output is to high => decrease sum to decrease output
    - If input I_i is +ve decrease W_{ji}
    - If input I_i is –ve increase W_{ji}
- If there is no error, then no need to do anything!

$$A_j = \sum_{i=1}^{n} W_{ji} I_i$$

Perceptron – How much to adjust by?

- There is no obvious value to change the weights by
- But only makes sense to change non-zero inputs, so include I_i value in the adjustment formula
- Also the t^d-o^d determines direction of correction
- So then we need one last factor that determines the size of the correction – the so-called "learning rate" $\boldsymbol{\alpha}$

$$W_{ji} \leftarrow W_{ji} + \alpha I_i^d (t^d - o^d)$$

- Learning rate usually small, say 0.1
- But we have to do this *repeatedly* for different inputs until they classify correctly

Perceptron – How much to adjust by?

- Need to consider what initial W_{ji} values should be
  - No obvious answer, so it is best to put the system in a 'random' state by assigning random values to W_{ji} initially
- Repeat as previously suggested but consider d examples together

$$W_{ji} \leftarrow W_{ji} + \alpha \sum_{d} I_i^d (t^d - o^d)$$

- If training cases are linearly separable and the learning rate is sufficiently small, this will converge
  - It is a gradient descent along the error surface
- Multiple examples on web show this training in progress (e.g. to an AND gate)
- But what about non-linearly separable? Need a different approach

Gradient descent/Adaline/Delta Rule

- What about a "best fit" methodology?
  - No surprise that we can do that via a squares minimization approach

$$E_j \equiv \frac{1}{2}\sum_d (t^d - o^d)^2$$

$$E_j \equiv \frac{1}{2}\sum_d (t^d - f(\sum_{i=1}^{n} W_{ji} I_i^d))^2$$

  - Notice if f is not differentiable (step function!) so we can't differentiate this formula to find minimum (threshold dropped for ease of notation)
- Can think about changing f, or substituting A_j rather than output value
- ADALINE=ADAptive LINear Element (Widrow & Hoff 1960)

Differentiate

- Using A_j rather than the full output then gives

$$E_j \equiv \frac{1}{2}\sum_d (t^d - \sum_{i=1}^{n} W_{ji} I_i^d)^2$$

- We want to differentiate w.r.t. to the weights so for ease of notation we'll drop the j from now on

$$\frac{\partial E}{\partial W_k} = \frac{1}{2}\sum_d 2(t^d - \sum_{i=1}^{n} W_i I_i^d)\frac{\partial}{\partial W_k}(t^d - \sum_{i=1}^{n} W_i I_i^d)$$

$$\frac{\partial E}{\partial W_k} = \sum_d (t^d - \sum_{i=1}^{n} W_i I_i^d)\,(-I_k^d)$$

Steepest descent

- Gradient defines direction of most rapid change in a field
- Take negative of that value to get descent toward minimum & multiply by scaling factor ***α***

$$\Delta W_k = \alpha \sum_d I_k^d (t^d - \sum_{i=1}^{n} W_i I_i^d)$$

- Thus update as:

$$W_k \leftarrow W_k + \alpha \sum_d I_k^d (t^d - \sum_{i=1}^{n} W_i I_i^d)$$

Convergence properties compared

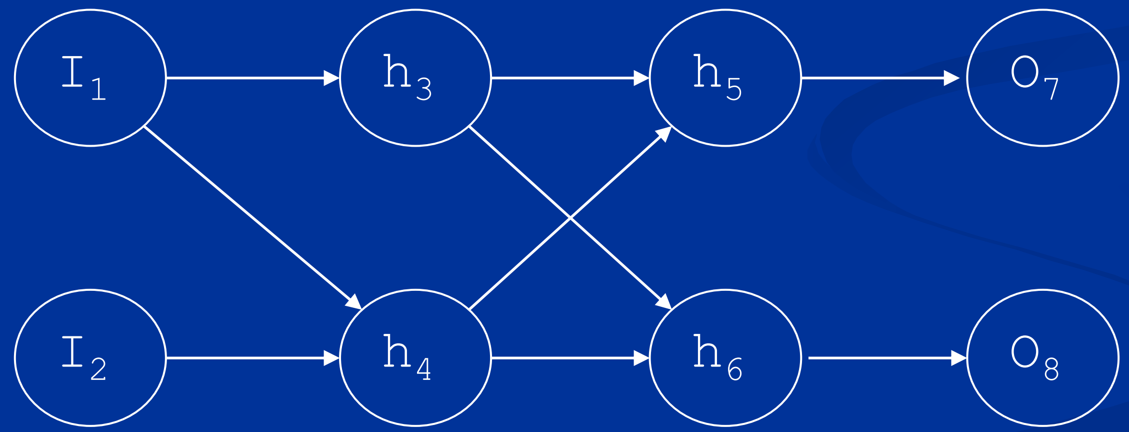
- Suppose problem is linearly separable
  - Perceptron learning will converge to unique linear separator
  - Steepest descent is not guaranteed to do so
- Suppose system is not linearly separable, will the minimum be found?
  - Perceptron learning cannot guarantee the minimum
  - Steepest descent will converge to the global minimum
- How do outliers influence results?
  - Perceptron – same weight for all
  - Steepest descent – have progressively larger influence
- How do correctly classified points influence training?
  - Perceptron – no influence
  - Steepest descent – strongly positive values have even larger impact

Minski & Papert 1969

- Perceptrons became popular despite their obvious limitations in single layer implementations
- M&P concluded that perceptrons are insufficient to construct generally intelligent machines
  - In essence general intelligence relies upon non-linearly separable hypotheses
  - What features of an object are used? Locality? Or global properties?
    - e.g. connectedness cannot be captured by local features alone – need global features
- Global features corresponds to potentially huge space
- But at time not clear how to make high level features sufficiently adaptive
- Evolution of backpropagation methods helped to do this

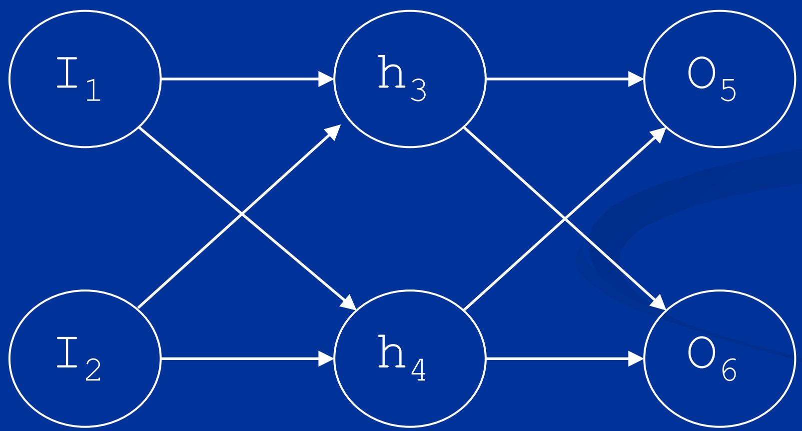
Multi-layer networks

- Organize into inputs, hidden units, and outputs
- Links (edges) go from one layer to the next



Single hidden layer – two inputs, two outputs

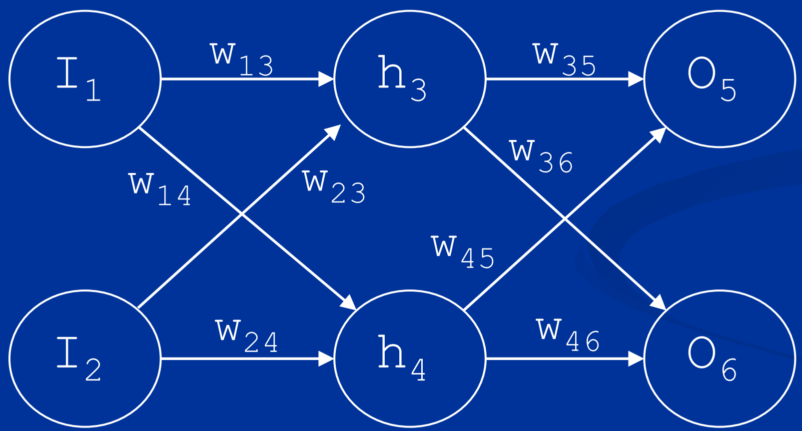
- Each input has an edge to the each hidden layer unit
- Each hidden layer unit has an edge to the output units



- Every edge has a weight, so need to label appropriately

Describing weights

- As in single layer, weights are parameters to be trained
- Use two index notation “from-to” for each w



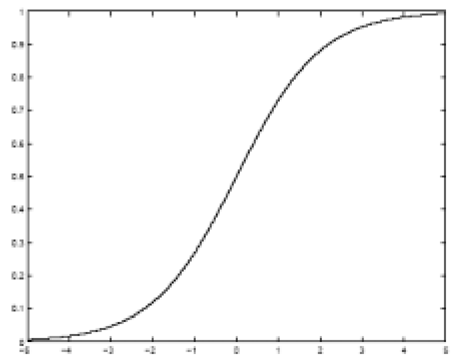
What to do about transfer functions?

- Each element in hidden and output layers has inputs and an associated transfer function f
- Assume output of each unit is $o_j=f(\mathbf{\Sigma} w_{ji} I_i)$
- We could again use the step-activated function, but non-differentiability is limiting
- Simple linear function is differentiable but doesn't have rapid transition properties
- Are there differentiable alternatives? Of course! ☺

Sigmoid

- Simple, continous differentiable function than transitions between 0 and 1 in well behaved way

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



$$\frac{dg(x)}{dx} = g(x)(1 - g(x))$$

Can include a bias too

Error & gradient descent

- For a two output model the error for input d will be the sum of the squares, $E^d=0.5*(t^d_5-o^d_5)^2+0.5*(t^d_6-o^d_6)^2$
- This can be differentiated and chain rule used twice

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

- Where

$$net_j = \sum_{k=1}^{n} w_{kj} o_k \quad so\ that \quad \frac{\partial net_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj} o_k\right) = o_i$$

More analysis

- Derivative of o_j wrt net_j is

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j} g(net_j) = g(net_j)(1 - g(net_j))$$

- That is why differentiability of transfer function is required
- For E differentiated wrt o_j – straightforward if o_j is a direct output

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

More analysis

- If o_j is not an output unit then need to consider E as a function of all units, L=u, v,…, w receiving input from j

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(net_u, net_{v,} \ldots, net_w)}{\partial o_j} = \sum_{l \epsilon L} \left(\frac{\partial E}{\partial net_l}\frac{\partial net_l}{\partial o_j}\right)$$

$$\frac{\partial E(o_j)}{\partial o_j} = \sum_{l \epsilon L} \left(\frac{\partial E}{\partial o_l}\frac{\partial o_l}{\partial net_l} w_{jl}\right)$$

- Can combine to get…

Back propagation

- Get

$$\frac{\partial E}{\partial w_{ij}} = \delta_j o_i$$

- Where

$$\delta_j = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} = \begin{cases} (o_j - t_j)o_j(1 - o_j) if\ j\ is\ output \\ (\sum_{l \epsilon L} \delta_l w_{jl})o_j(1 - o_j)\ \ if\ j\ is\ inner \end{cases}$$

- Updates to w_{ij} are given by

$$\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$$

- Where – sign takes in direction of minimum

See http://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
for a great step-by-step example on the network we've used

Summary

- Neural network theory begins from simple building blocks of the MP model
  - Generalize to perceptrons
  - Then onward to multi-layer systems
  - Train by minimizing an error function – no surprise
- Training of linear separable systems is optimized using perceptron, but if not linearly separable steepest descent is better
- For multi-layer networks steepest descent methods can be applied, but analysis becomes recursive over layers