Computational Psycholinguistics

Lecture 8: Introduction to Connectionist Models

Marshall R. Mayberry

Computerlinguistik
Universität des Saarlandes

Connectionist language learning: contents

- Connectionist Information Processing
  - Simple connectionist models and their properties: The perceptron
  - Multi-layer perceptrons: feed-forward networks and internal representations
  - The encoding problem: Localist and distributed representations
  - Generalisation and association

- Connectionist Models of Language
  - Modelling acquisition of the English Past-Tense and reading aloud
  - Processing sequences: Simple recurrent networks
  - Modelling acquisition of hierarchical syntactic knowledge

- Tutorials: `tLearn` neural network simulator (Win/Mac/Linux)
  - Introduction and Learning with `tLearn`
  - Autoassociation and cluster analysis
  - The English past-tense
  - SRNs

Theories of Language

- Theories of the human language faculty:
  - Knowledge: what is the nature of our knowledge of language?
    - Rules and Representations
    - Symbolic versus Distributed
    - Explicit versus Implicit
  - Acquisition: where does knowledge come from?
    - Is some knowledge innate?
    - What linguistic knowledge is learned, and how?
  - Usage: how do people use knowledge to process new input?
    - What mechanisms do people use in applying existing linguistic knowledge to the interpretation of novel input?
- Connectionist models of human language:
  - Addresses these issues simultaneously

Connectionist Information Processing

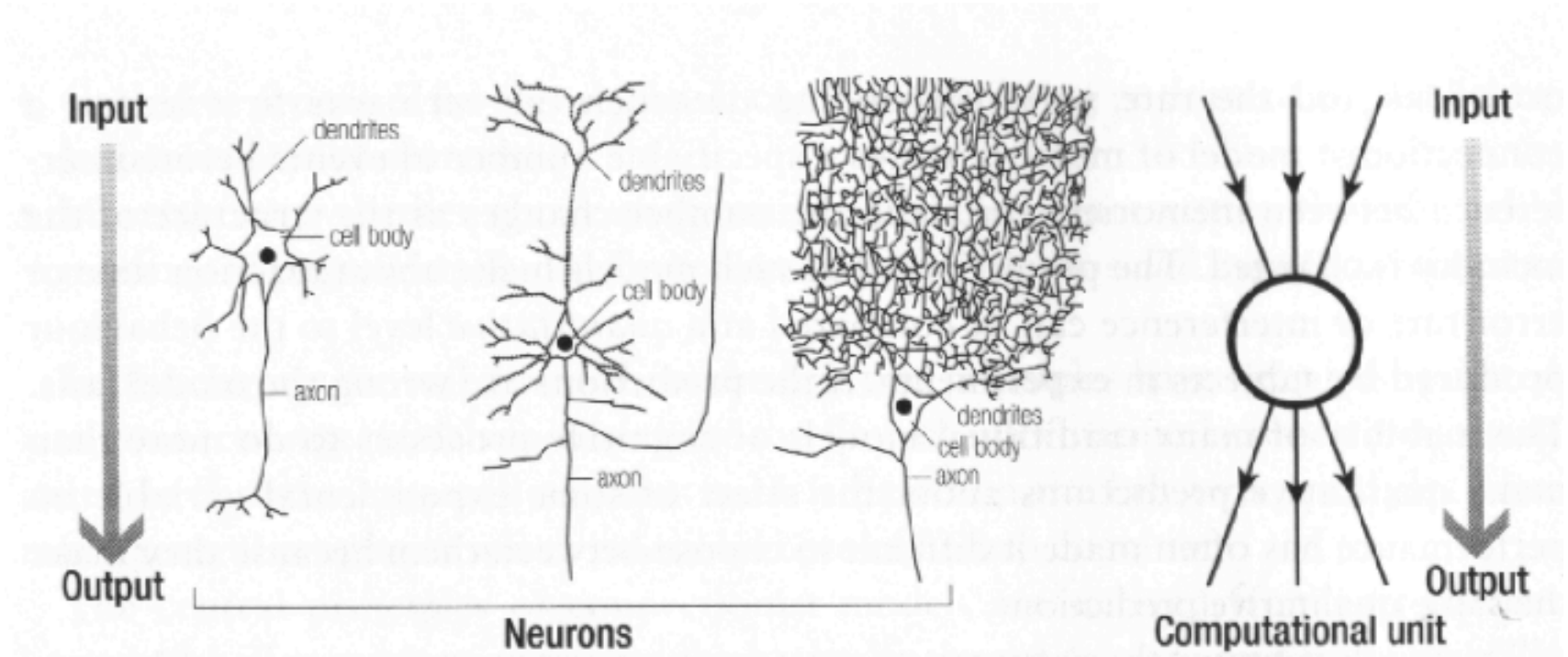
- Connectionist models of information processing can become complex, but the idea is based on simple neuronal processing in the brain:
  - Basic computational operation involves one neuron passing information related to the sum of the signals reaching it onto other neurons
  - Learning involves changing the strength of the connections between neurons, and thus the influence they have upon each other
  - Cognitive processes involve the use of large numbers of neurons to perform these basic computations in parallel
  - Information about an input signal or memory of past events is distributed across many neurons and connections
- Terms: connectionism, parallel distributed processing, neural networks, neurocomputing

Assumptions about the brain ...

- ... On which connectionist models are based.

- Neurons integrate information:
  - *All neuron types sum inputs and compute an output*
- Neurons pass information about the strength of their input:
  - *Output encodes information about the degree of input: firing rate*
- Brain structure is layered:
  - *Information passes through sequences of independent structures*
- Influence of one neuron upon another depends on connection strength:
  - *A given neuron is connected to thousands of other neurons, but its influence on a particular node is determined by synaptic strength*
- Learning is accomplished through changing connection strengths:
  - *There is evidence that this is so*
  - *BUT most connectionist learning rules are not biologically plausible.*

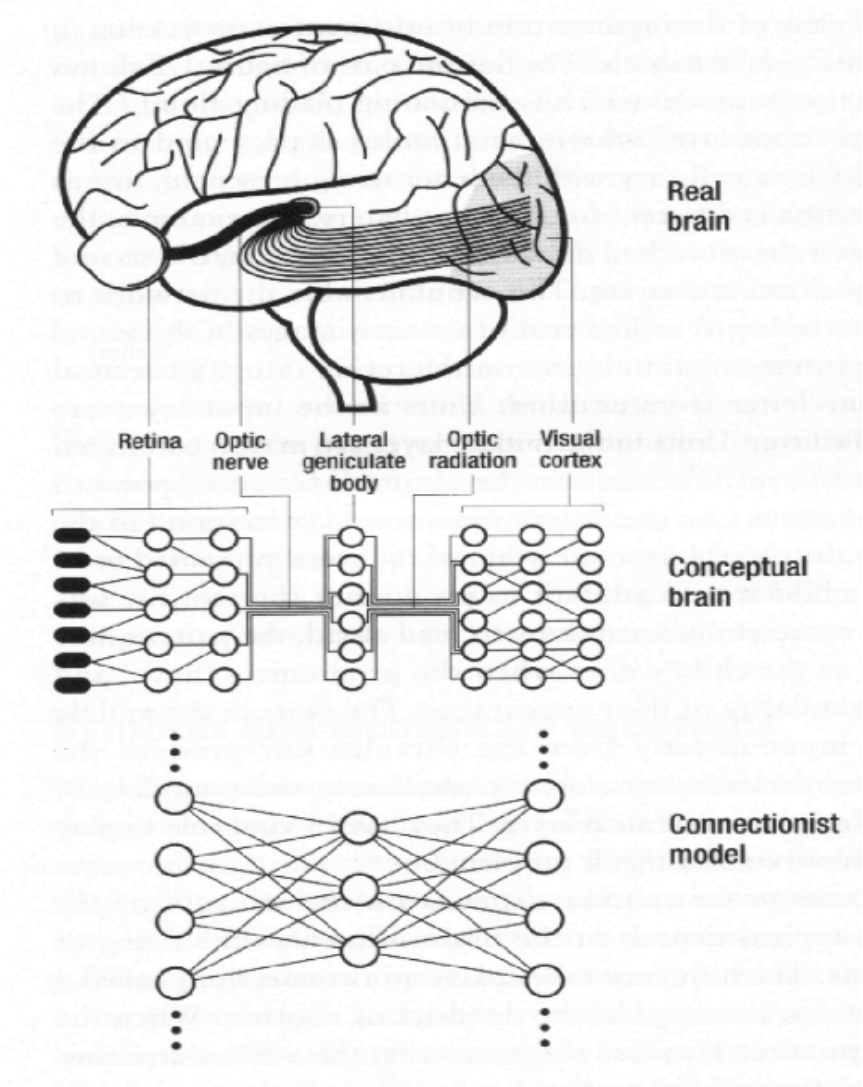
Neurons versus Nodes



- Neurons receive signals (excitatory or inhibitory) from other neurons via synaptic connections to its dendrites.
- If the sum of these signals exceeds a certain threshold, then the neuron fires, sending a signal along its axon.

Brain versus Network

- The human brain contains approximately 10^{10} - 10^{11} neurons
- Those neurons are densely interconnected:
  - 10^5 connections per neuron
  - Thus, 10^{15} - 10^{16} connections in total
- Connections can be both excitatory and inhibitory
- Learning involves modifying of synapses (connections)
- Connections can be both added and eliminated (pruning)



The “Connectionist” Perspective

- Rumelhart and McClelland (1987, p. 196):
  - *“... implicit knowledge of language may be stored among simple processing units organized into networks. While the behaviour of such networks may be describable (at least approximately) as conforming to some system of rules, we suggest that an account of the fine structure of the phenomena of language and language acquisition can be best formulated in models that make reference to the characteristics of the underlying networks”*

- Key ideas:
  - Neurologically based (but not true models of the brain)
  - Distributed, implicit representations
  - Dense connectivity
  - Communication of “real values” not “symbols”
  - Representations and processing are the same
  - Learning: supervised and unsupervised

Properties of Connectionist Networks

- Learning
  - There is usually no predetermined (innate) knowledge of language, but ...
    - Input/output representations are often specified
    - The architecture of the network may be “suited” to a particular task
    - The learning mechanism and parameters provide degrees of freedom
  - Learning takes place in direct response to experience

- Generalisation
  - Networks are able to learn generalisations, not just by rote
  - More efficient representation of information
  - Novel inputs can be processed

- Representation
  - Learned automatically, and typically distributed

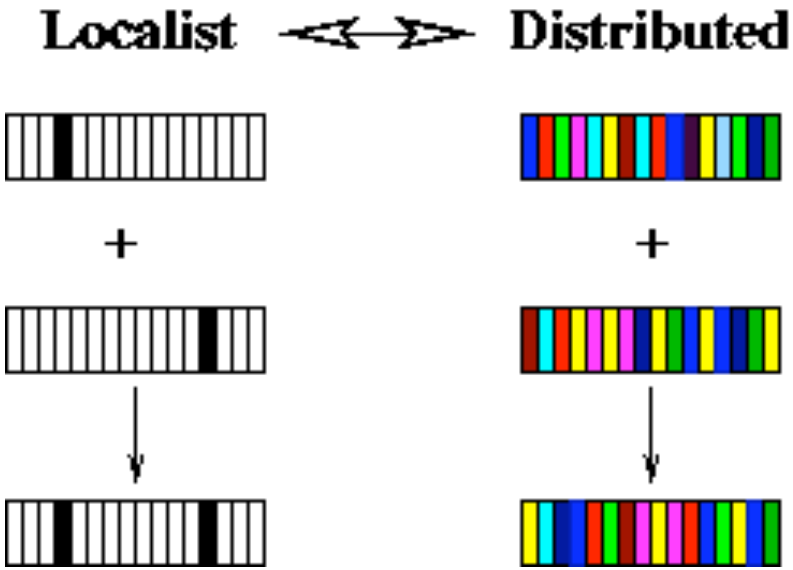
Properties continued

- Rules versus exceptions
  - Single mechanism to explain both general rules and also exceptions
- Graded:
  - Can often give a useful output to new, partial, noisy input (pattern completion)
  - Damage is distributed, and some performance is still possible:
    - Modelling of brain damage and neurological disorders
- Frequency effects
  - Model response time behaviours where high frequency inputs are recognised faster than low frequency ones

Computational Properties of Networks

- Neurally inspired:
  - Slow and parallel
  - Highly interconnected
  - Learning by changing connection strength
  - Processing is distributed/ “decentralized”
- Neuron is the basic processing unit
- Network configuration = “program”
- Local computation yields global behaviour
- Long-term memory is in the strength of connections (weights)
- Short-term memory is in the pattern of activity

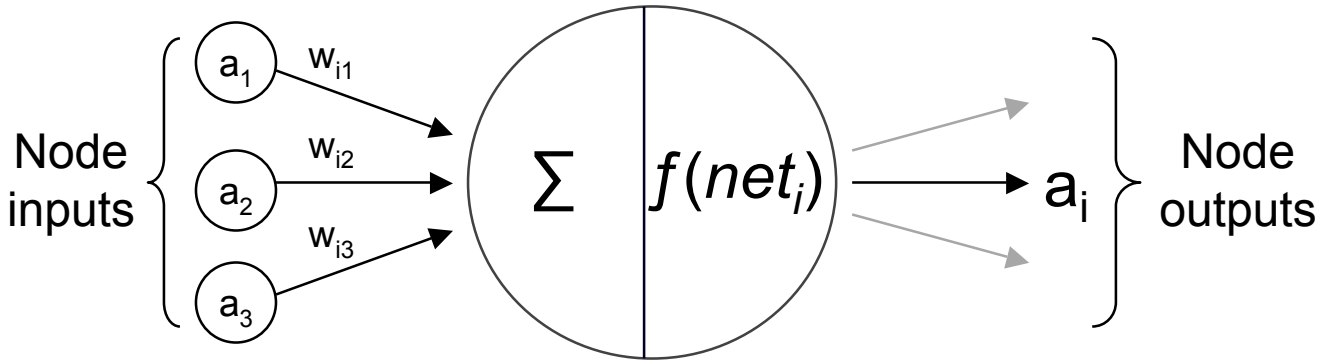
Representation



- One-to-one
- Discrete
- Fragile
- Literal
- Limited

- Many-to-one
- Continuous
- Robust
- Generalisable
- Unbounded

Basic Structure of Nodes



- A node can be characterised as follows:
  - Input connections representing the flow of activation from other nodes or some external source
  - Each input connection has its own *weight*, which determines how much influence that input has on the node
  - A node *i* has an output activation $a_i = f(net_i)$ which is a function of the weighted sum of its input activations, *net*.

- The net input is determined as follows: $$net_i = \sum_j w_{ij} a_j$$

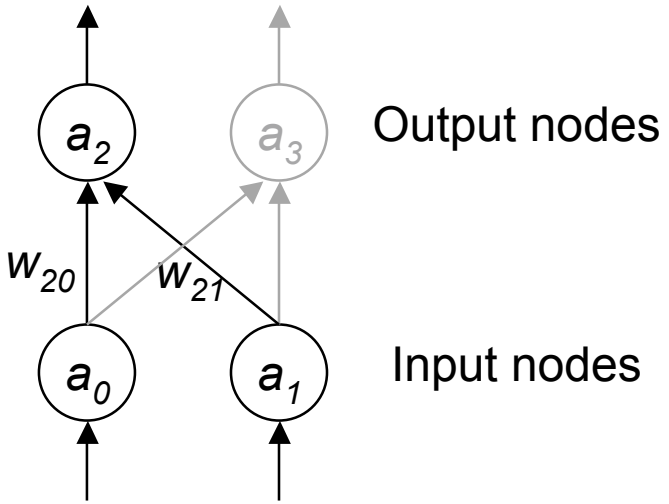
An example

- A one-layer feed-forward network:

$$net_i = \sum_j w_{ij} a_j$$

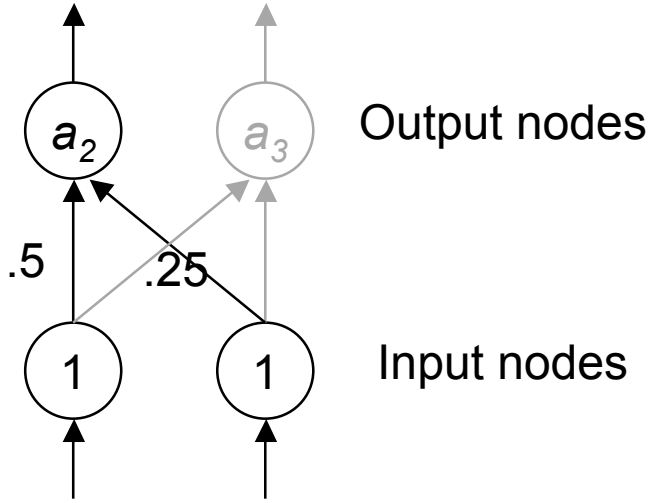
- So the net input for a_2 is:

$$\text{net input } a_2 = w_{20} \cdot a_0 + w_{21} \cdot a_1$$



- Consider the network with the following inputs and weights:

- The net input for node a_2 is:
  - 1 x .5 + 1 x .25 = 0.75



About weights

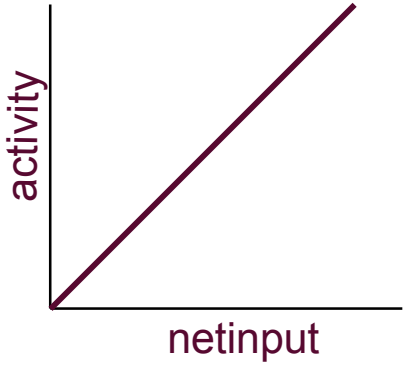
- Node *j* influences node *i* by passing information about its activity level.
- The degree of influence it has is determined by the weight connecting node *j* to node *i*.
  - A smaller weight corresponds to reduced influence of one particular node on another
  - A larger weight emphasises the influence of the node's activation
- Weights can be either positive or negative
  - Positive weights contribute activation to the net input
  - Negative weights lead to a reduction of the net input activation
  - Brain: *excitatory* versus *inhibitory* connections

Activation functions

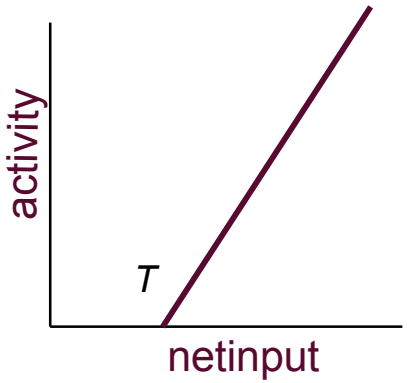
- The activation function determines the activation a_i for node *i* from the net input (net_i) to the node: f(net_i)

- Linear activation function
  - (McCulloch-Pitts neurode, perceptron)
  - Identity: the $a_i = net_i$

$$f(net_i) = net_i$$
$$f(0.75) = 0.75$$

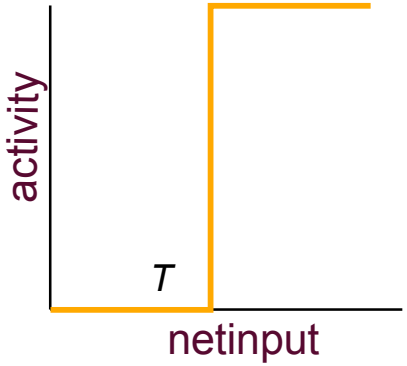


- Threshold activation function:
  - IF $net_i > T$ THEN $a_i := net_i - T$
  - ELSE $a_i := 0$



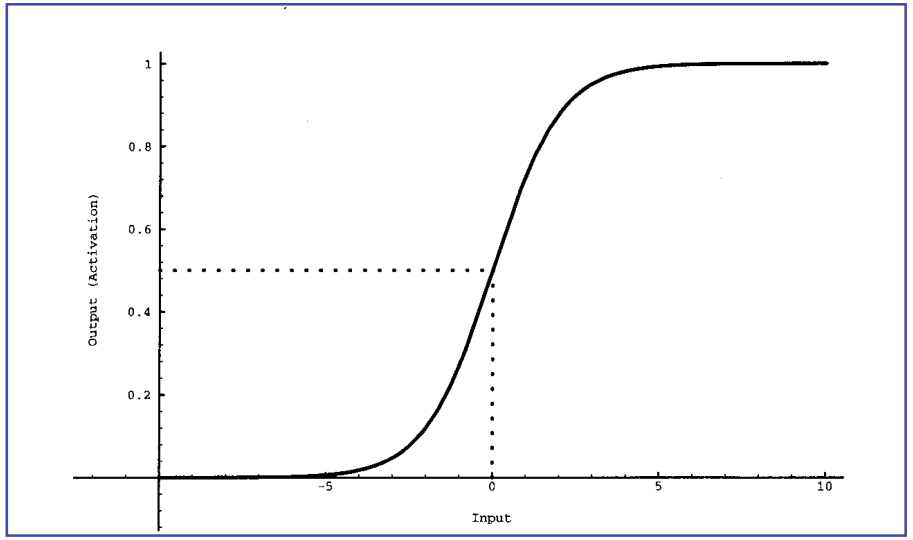
More Activation Functions

- Binary threshold activation function:
  - IF $net_i > T$ THEN $a_i := 1$
  - ELSE $a_i := 0$



- Nonlinear activation function
  - It is often more useful to use the "sigmoidal" logistic function:

$$a_i = f(net_i) = \frac{1}{1 + e^{-net_i}}$$

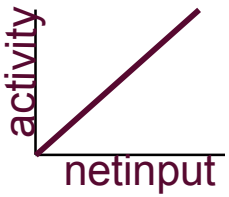


Calculating the activation: net_i is 1.25

- Linear activation:

$$f : \Re \rightarrow \Re$$

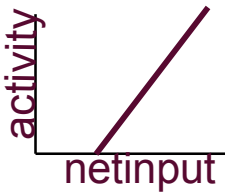
$$f(net_i) = net_i$$
$$f(1.25) = 1.25$$



- Linear threshold: T=0.5

$$f : \Re \rightarrow \Re$$

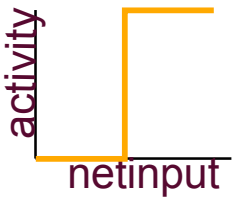
$$\text{IF } net_i > T \text{ then } f(net_i) = net_i - T$$
$$\text{ELSE } f(net_i) = 0$$
$$f(1.25) = 1.25 - 0.5 = 0.75$$



- Binary threshold: T=0.5

$$f : \Re \rightarrow [0,1]$$

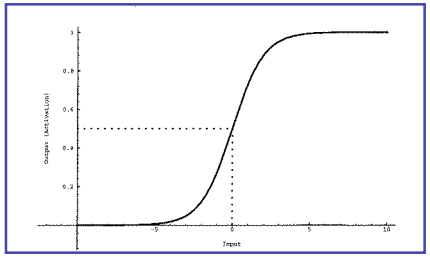
$$\text{IF } net_i > T \text{ then } f(net_i) = 1$$
$$\text{ELSE } f(net_i) = 0$$
$$f(1.25) = 1$$



- Nonlinear activation:
  - Sigmoid or "logistic" function

$$f : \Re \rightarrow [0,1]$$

$$f(net_i) = \frac{1}{1 + e^{-net_i}}$$
$$f(1.25) = 0.777$$



About activation functions

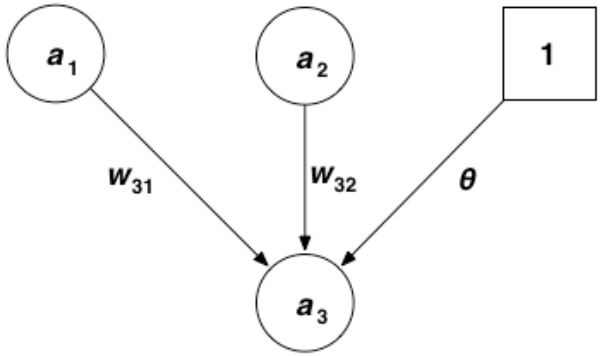
- The activation function defines the relationship between the net input to a node, and its activation level (which is also its output).
- Neurons in the brain have thresholds, only fire with sufficient net input.
- Nonlinearity (i.e. a small change in input can result in large change in output) can be useful to reduce the effects of spurious inputs, noise.
- Most common in connectionist modelling: sigmoid/logistic
  - Activation ranges between 0 and 1
  - Rate of activation change is highest for net inputs around 0
  - Models neurons by implementing thresholding, a maximum activity, and smooth transition between states.
- The sigmoid function also has nice mathematical properties

Logistic Function: bias and gain

- Weighted sum of inputs

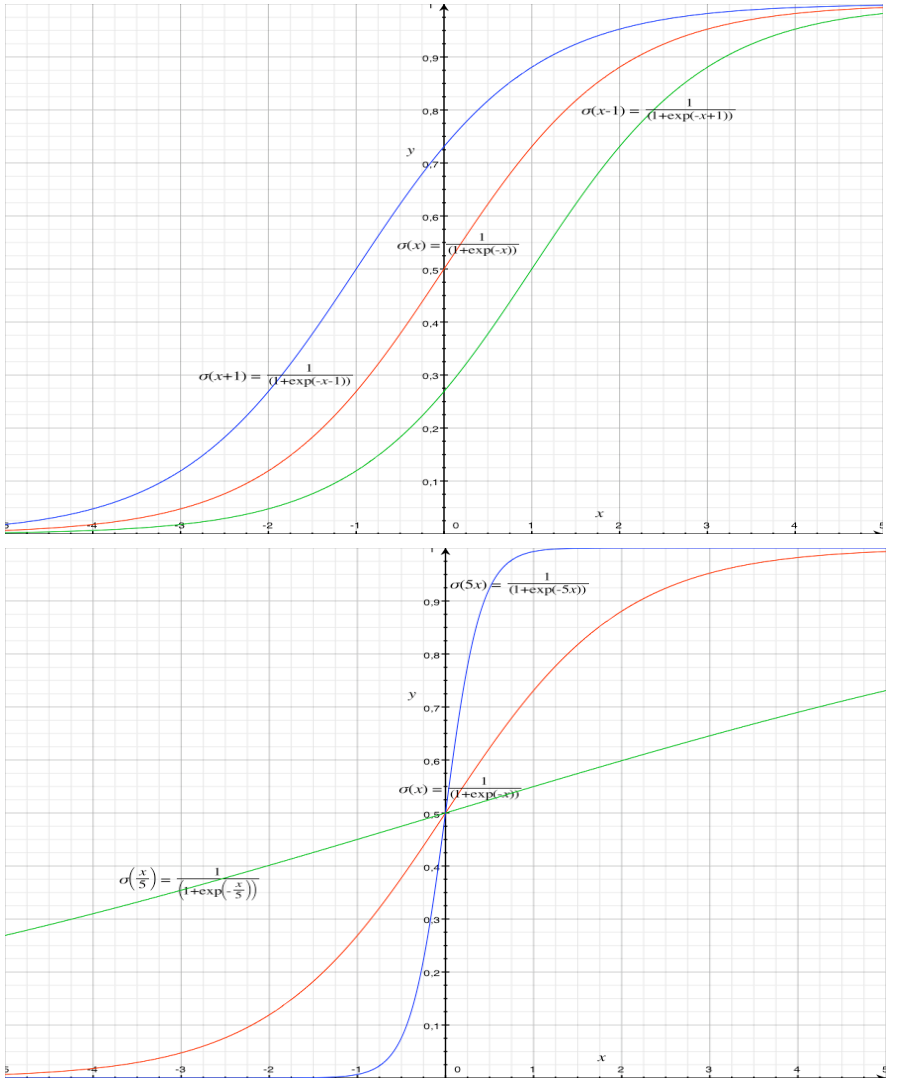
$$net_i = \sum_j w_{ij} a_j$$

- Bias θ: threshold weight



- Gain γ: slope of logistic function

$$a_3 = \sigma(\gamma\, net_3 + \theta) = \frac{1}{1 + e^{-(\gamma\, net_3 + \theta)}}$$



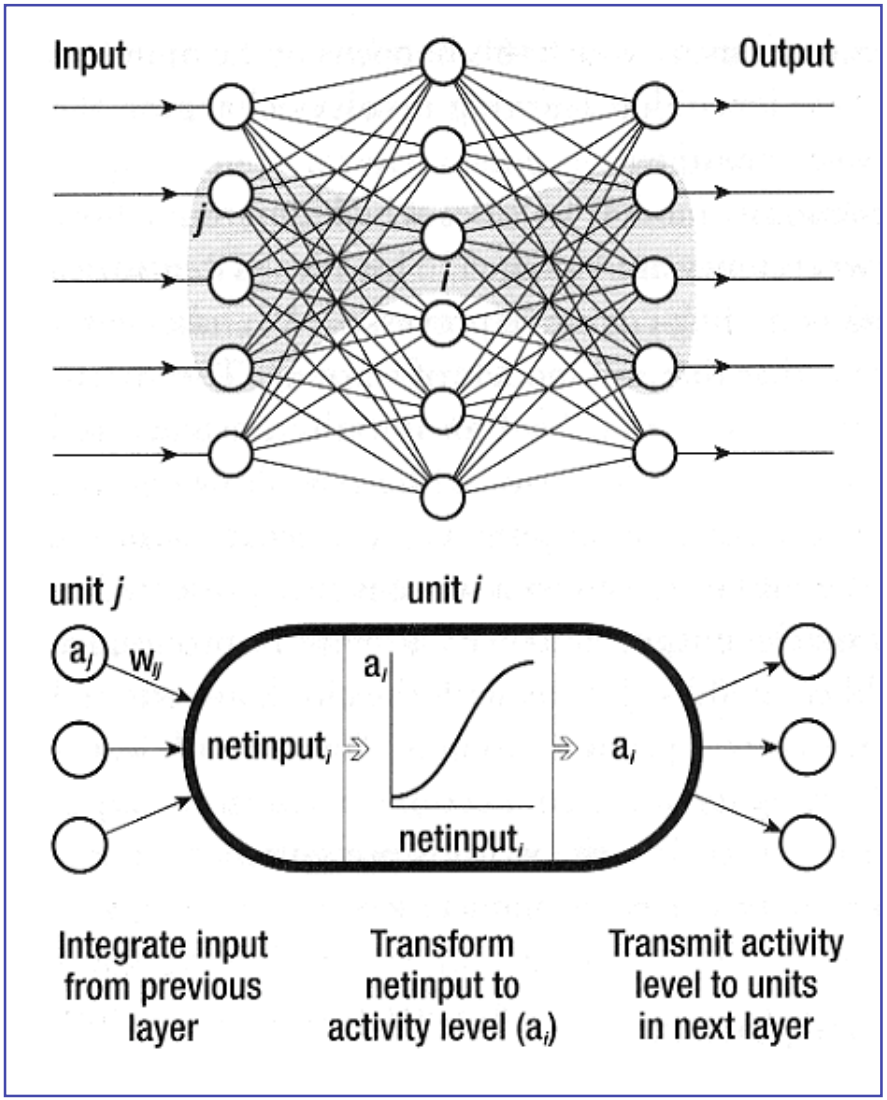
Summary of network architecture

- The activation of a unit *i* is represented by the symbol a_i.
- The extent to which unit *j* influences unit i is determined by the weight w_{ij}
- The input from unit *j* to unit *i* is the product: $a_j * w_{ij}$
- For a node *i* in the network:

$$net_i = \sum_j w_{ij} a_j$$

- The output activation of node *i* is determined by the activation function, e.g. the logistic:

$$a_i = f(net_i) = \frac{1}{1 + e^{-net_i}}$$



Learning in connectionist networks

- **Supervised learning** in connectionist networks involves successively adjusting connection weights to reduce the discrepancy between the *actual output activation* and the *target output activation*
  - An input is presented to the network
  - Activations are propagated through the network to its output
  - Outputs are compared to "correct" outputs: difference is called *error*
  - Weights are adjusted

- The Delta Rule:

$$\Delta w_{ij} = \left(t_i - a_i\right) a_j \varepsilon$$

  - (t_i - a_i) is the difference between the target output activation and the actual activation produced by the network
    - What is the "error"?
  - a_j is the activity of the contributing unit j
    - How much activation is this unit responsible for? (Credit/blame assignment)
  - ε is the learning rate parameter.
    - How rapidly do we want to make changes?

Training the Network

- Consider the AND function
  - Present stimulus: 0 0
  - Compute output activation
  - Compared with desired output (0)
  - Use Delta rule to change weights
  - Present next stimulus: 0 1
  - ...

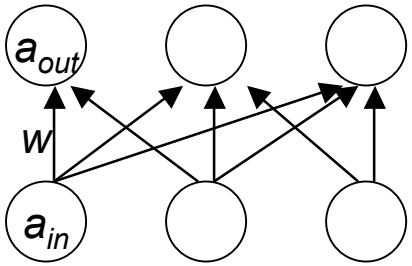
Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

- An Epoch, consists of a single presentation of all training examples
  - Here there are 4 such examples

- A Sweep, is a presentation of a single training example
  - So, 250 epochs consists of 1000 sweeps

"Perceptrons" [Rosenblatt 1958]

- Perceptron: a simple, one-layer, feed-forward network:



$$\text{net}_{out} = \sum_{in} w \cdot a_{in}$$

- Binary threshold activation function:

$$\begin{aligned} a_{out} &= 1 \text{ if net}_{out} > \theta \\ &= 0 \text{ otherwise} \end{aligned}$$

- Learning: the perceptron convergence rule
  - Two parameters can be adjusted:
    - The threshold
    - The weights

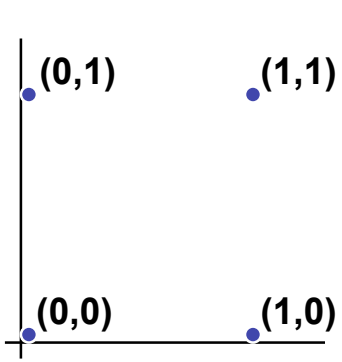
$$\text{The error, } \delta = (t_{out} - a_{out})$$

$$\Delta\theta = -\varepsilon\delta$$

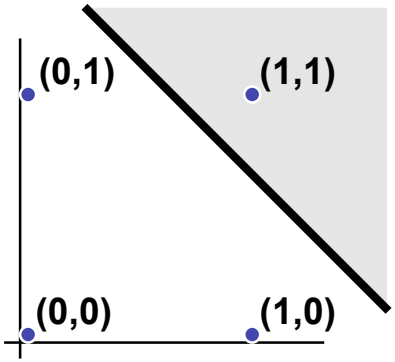
$$\Delta w = \varepsilon\delta a_{in}$$

2-D Representation of Boolean Functions

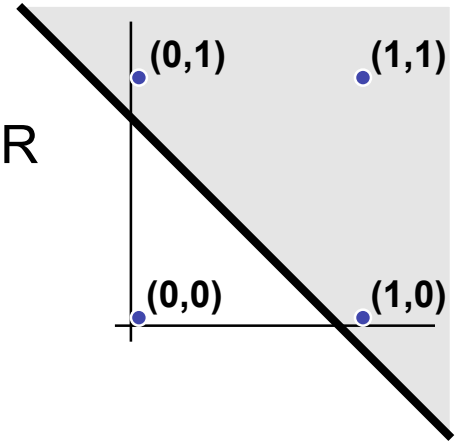
- We can visual the relationship between inputs (plotted in 2-D space) and the desired output (represented as a line dividing the space):



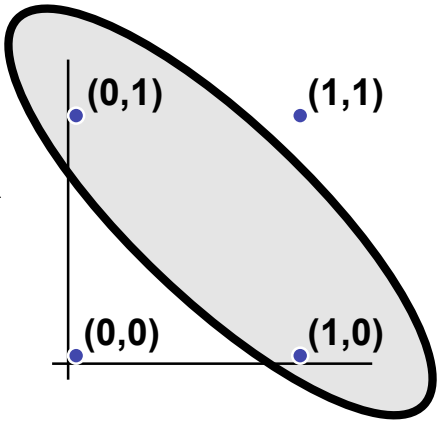
AND



OR



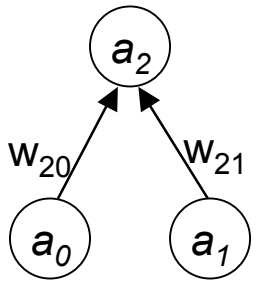
XOR



Learning OR

- Consider the following simple perceptron:
  - Recall the convergence rule:

$$\text{The error, } \delta = (t_{out} - a_{out})$$
$$\Delta\theta = -\varepsilon\delta$$
$$\Delta w = \varepsilon\delta a_{in}$$



Classification problem

a_0	a_1	a_2
0	0	0
0	1	1
1	0	1
1	1	1

- We want to train this to learn boolean OR:
  - Note: changes have opposite signs
    - E.g if activity is less than target, δ is positive
      - Threshold is decreased
      - Weight is increased
  - If δ is non-zero, threshold is always changed
    - But if a_{in} is zero, the weight is not changed
  - The changes can be calculated straightforwardly, but do they lead to convergence on a solution to a problem?

Learning OR continued …

- Recall the convergence rule:

$$a_{out} = a_2 = \sum_j w_{2j} a_j$$

The error, $\delta = (t_{out} - a_{out})$

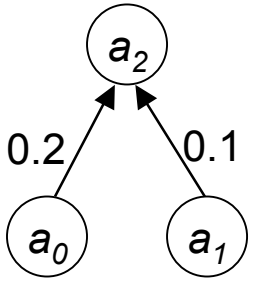
$$\Delta\theta = -\varepsilon\delta$$

$$\Delta w = \varepsilon\delta a_{in}$$

And the net:

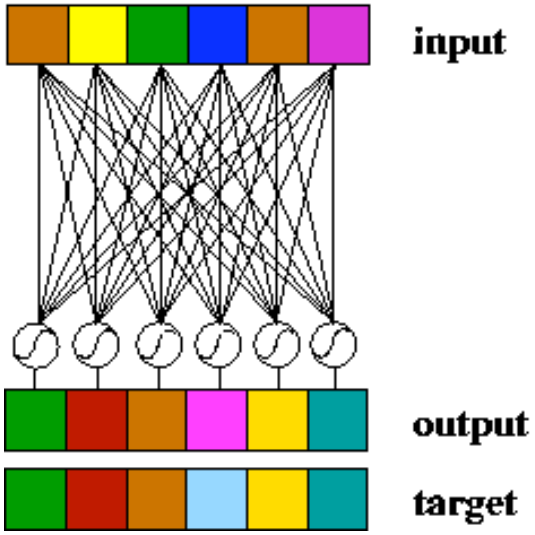
$$\theta = 1$$

$$\varepsilon = 0.5$$



a_0 a_1	w_{20}	w_{21}	Σ	θ	a_2	t_2	δ	$\Delta\theta$	Δw_{20}	Δw_{21}

Learning in a nutshell



- Patterns are vectors on [0,1]
- Input pattern is passed through a weight matrix
- Net values are summed and squashed to [0,1]
- Output pattern is compared to target pattern
- Error between output and target is propagated back through weight matrix
- Weights are changed to minimize error

Summary

- Connectionism is inspired by information processing in the brain
- Models typically contain several layers of processing units
  - Units correspond to a neuron (or group of neurons)
  - Units sum weighted inputs from previous layers, and compute activation
  - Output activation is passed to units of the next layer
- An input stimulus causes a “pattern of activation” on the first layer
  - Activations are then propagated through the network
  - The influence of one unit upon another is determined by the weight
  - The output response is the “pattern of activation” on the final layer
- Learning aims to reduce the discrepancy between actual and desired output patterns of activation
  - The Delta rule iteratively changes the weights of successive epochs
  - Training is complete when error is sufficiently reduced