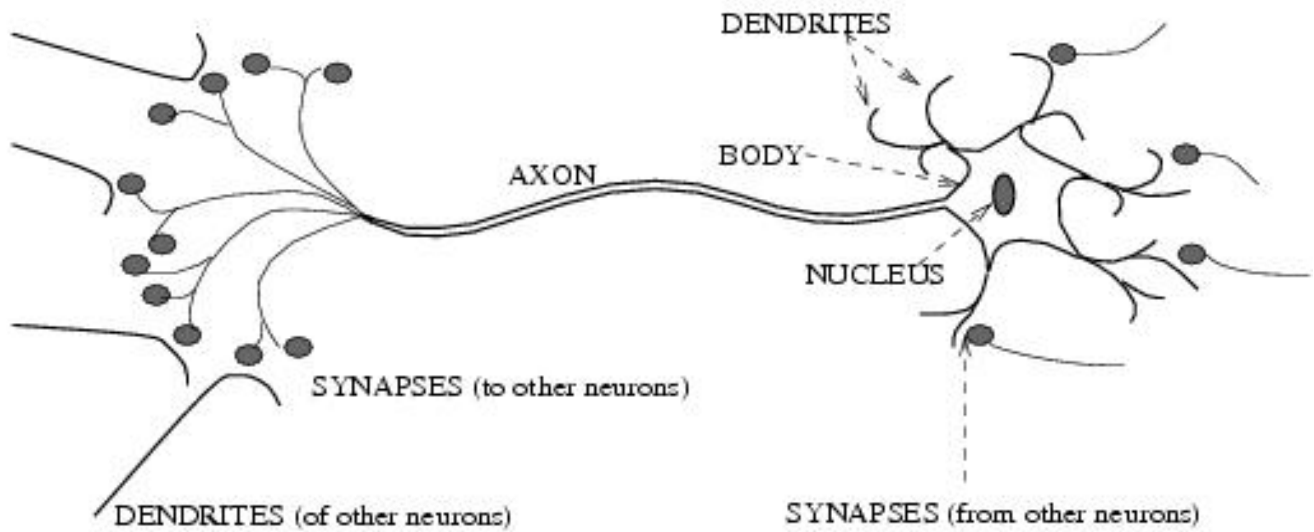
Neural Networks

Biological neural activity



- Each neuron has a *body*, an *axon*, and many *dendrites*
  - Can be in one of the two states: *firing* and *rest*.
  - Neuron fires if total incoming stimulus exceeds a threshold
- *Synapse*: thin gap between axon of one neuron and dendrite of another.
  - Signal exchange
  - Synaptic strength/efficiency

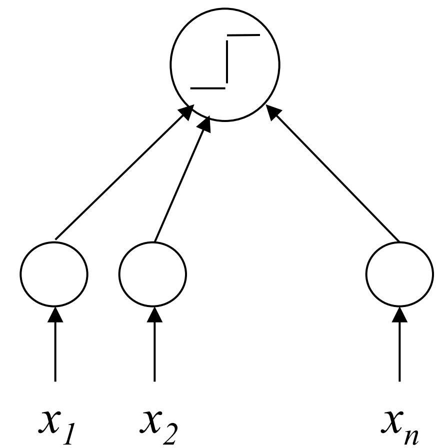
Artificial neural network

- Set of **nodes** (units, neurons, processing elements)
  - Each node has input and output
  - Each node performs a simple computation by its **node function**
- **Weighted connections** between nodes
  - Connectivity gives the structure/architecture of the net
  - What can be computed by a NN is primarily determined by the connections and their weights
- Simplified version of networks of neurons in animal nerve systems

History of NN

- **Pitts & McCulloch (1943)**
  - First mathematical model of biological neurons
  - All Boolean operations can be implemented by these neuron-like nodes
  - Competitor to Von Neumann model for general purpose computing device
  - Origin of automata theory
- **Hebb (1949)**
  - Hebbian rule of learning: increase the connection strength between neurons i and j whenever both i and j are activated.
  - Or increase the connection strength between nodes i and j whenever both nodes are simultaneously ON or OFF.

History: Early booming (50s – early 60s)



– Rosenblatt (1958)
  - Perceptron: network of threshold nodes for pattern classification
    Perceptron learning rule
  - Percenptron convergence theorem: everything that can be represented by a perceptron can be learned

– Widrow and Hoff (1960, 19062)
  - Learning rule based on gradient descent (with differentiable unit)

– Minsky' s attempt to build a general purpose machine with Pitts/McCullock units

History: setback in mid 60s – late 70s)

- Serious problems with perceptron model (Minsky's book 1969)
  - Single layer perceonptrons cannot represent (learn) simple functions such as XOR
  - Multi-layer of non-linear units may have greater power but there is no learning rule for such nets
  - Scaling problem: connection weights may grow infinitely
  - The first two problems overcame by latter effort in 80's, but the scaling problem persists
- Death of Rosenblatt (1964)
- Striving of Von Neumann machine and AI

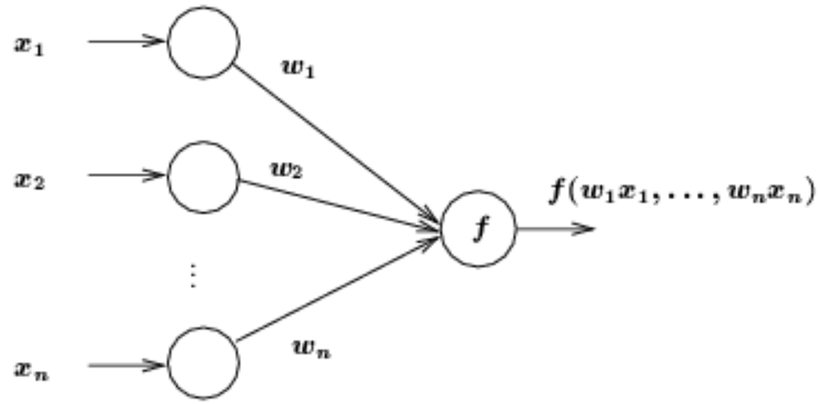
History of NN: Renewed enthusiasm

- New techniques
  - Backpropagation learning for multi-layer feed forward nets (with non-linear, differentiable node functions)
  - Thermodynamic models (Hopfield net, Boltzmann machine, etc.)
  - Unsupervised learning
- Impressive application (character recognition, speech recognition, text-to-speech transformation, process control, associative memory, etc.)
- Traditional approaches face difficult challenges
- Caution:
  - Don't underestimate difficulties and limitations
  - Poses more problems than solutions

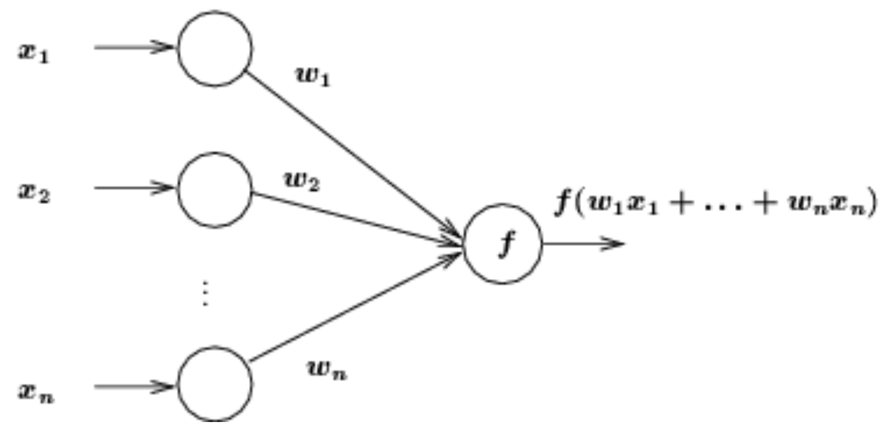
ANN Neuron Models

- Each node has one or more inputs from other nodes, and one output to other nodes
- Input/output values can be
  - Binary {0, 1}
  - Bipolar {-1, 1}
  - Continuous (bounded or not)
- All inputs to a node come in at same time and remain activated until output is produced
- Weights associated with links
- Node function

  $f(net)$ is the most popular node function where $net = \sum_{i=1}^{n} w_i x_i$

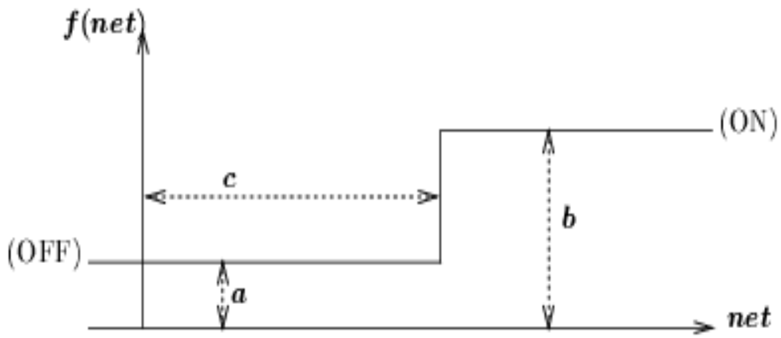


General neuron model

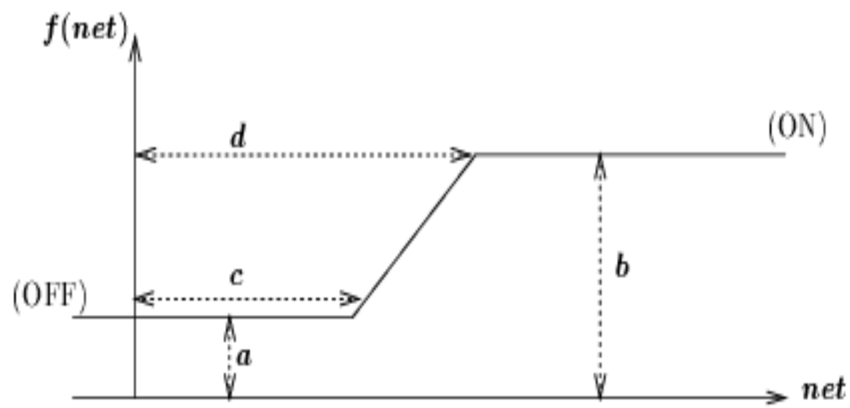


Weighted input summation

Node Function



Step function



Ramp function

Node Function

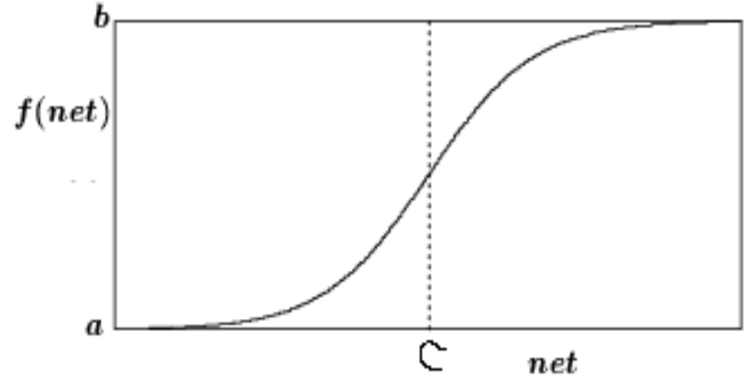
- **Sigmoid function**
  - S-shaped
  - Continuous and everywhere differentiable
  - Rotationally symmetric about some point (*net = c*)
  - Asymptotically approaches saturation points

$$\lim_{\text{net}\to-\infty} f(\text{net}) = a \quad \lim_{\text{net}\to\infty} f(\text{net}) = b$$

  - Examples:

$$f(\text{net}) = z + \frac{1}{1+\exp(-x\cdot\text{net}+y)}$$

$$f(\text{net}) = \tanh(x\cdot\text{net}-y)+z,$$



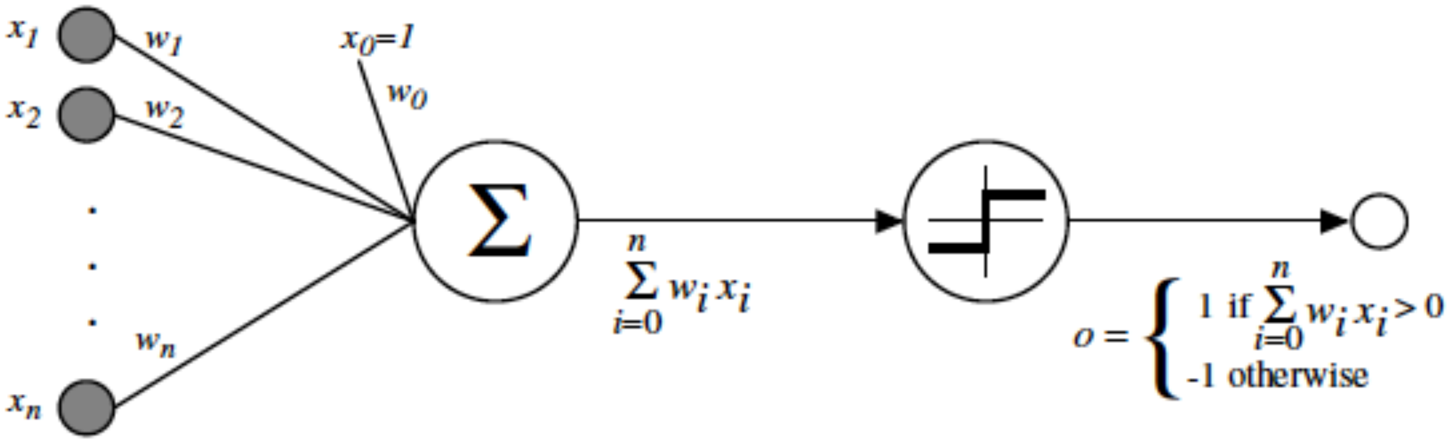
Sigmoid function

When $y = 0$ and $z = 0$:
$a = 0$, $b = 1$, $c = 0$.
When $y = 0$ and $z = -0.5$
$a = -0.5$, $b = 0.5$, $c = 0$.

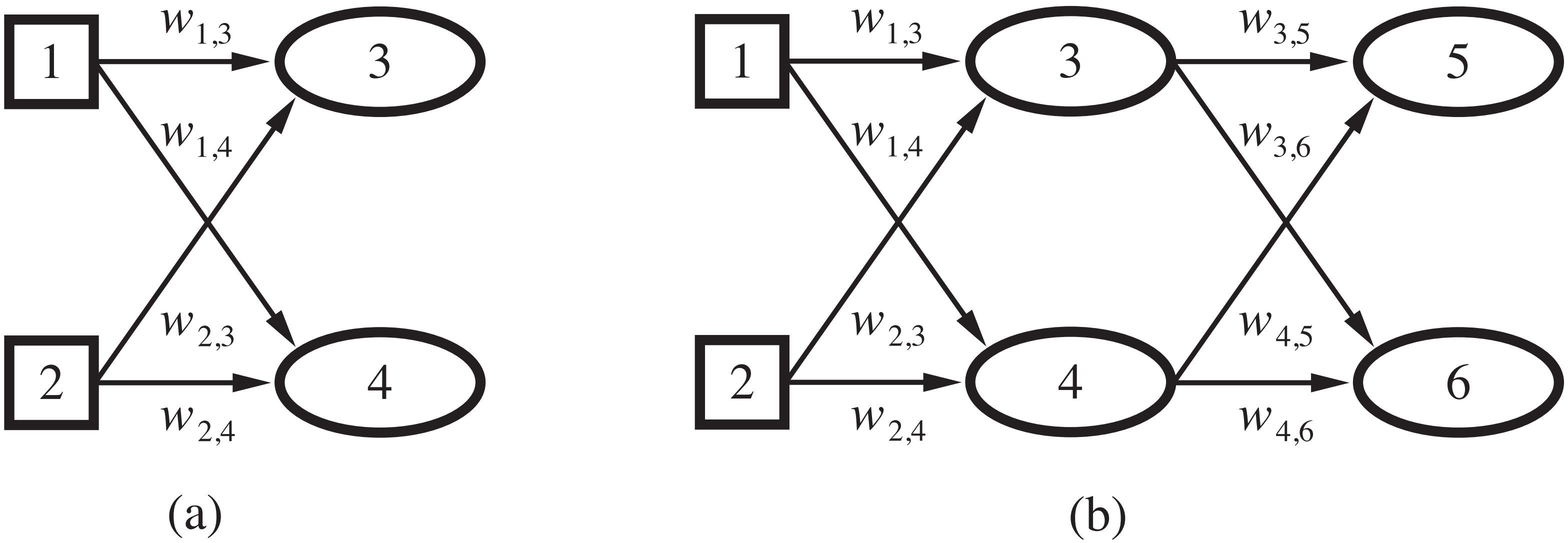
Larger x gives steeper curve

Perceptron

A single layer neural network

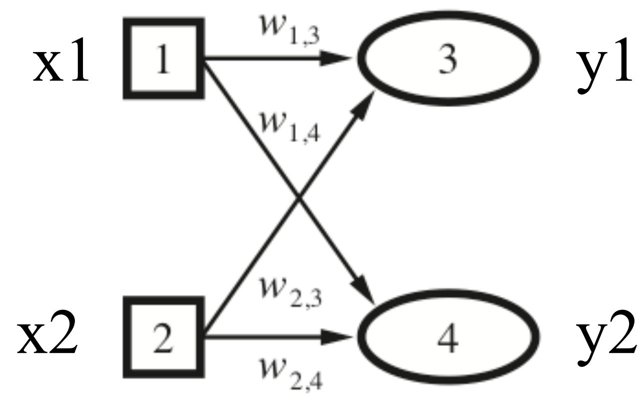


Simple architectures



Can we make a two bit adder?

- Inputs are bits x1 and x2
- Outputs: carry bit (y1), sum bit (y2)
- Two NNs, really



X1	X2	Y1 (carry)	Y2 (sum)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

Perceptron training rule

Adjust weights slightly to reduce error between perceptron output **o** and target value **t;** repeat

$$w_i \leftarrow w_i + \Delta w_i$$

where

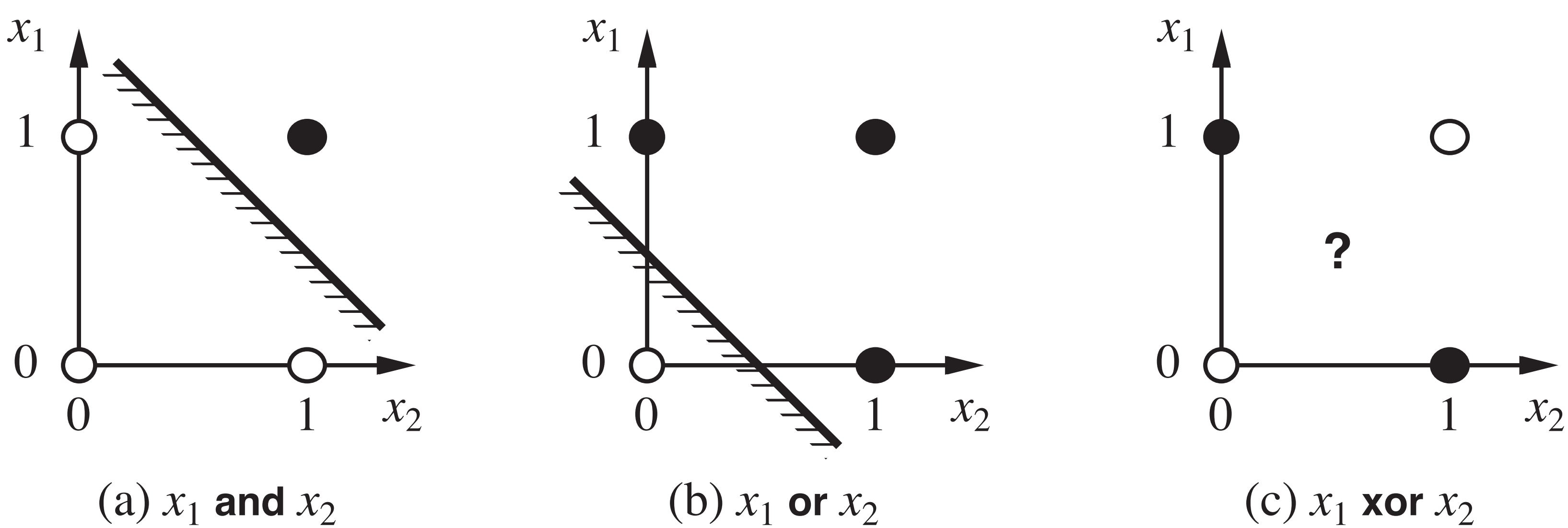
$$\Delta w_i = \eta(t - o)x_i$$

Where:

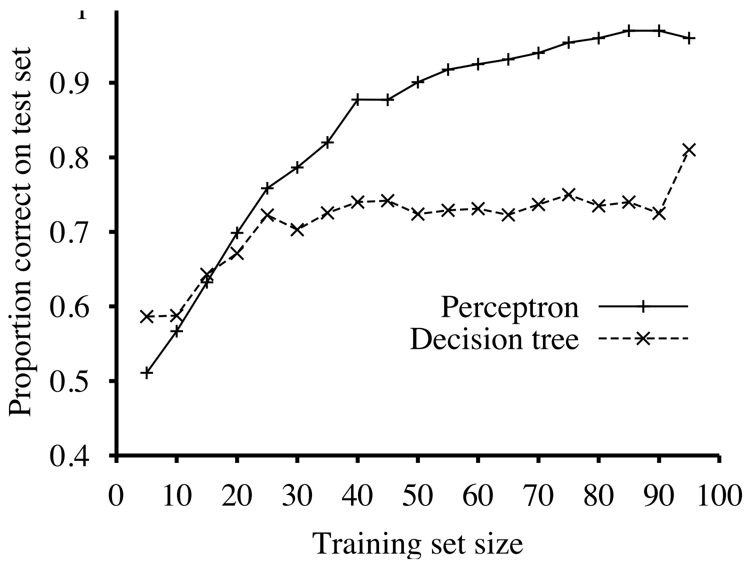
- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

Not with a perceptron ☹

Training examples are not linearly separable for one case: *sum=1 iff x1 xor x2*

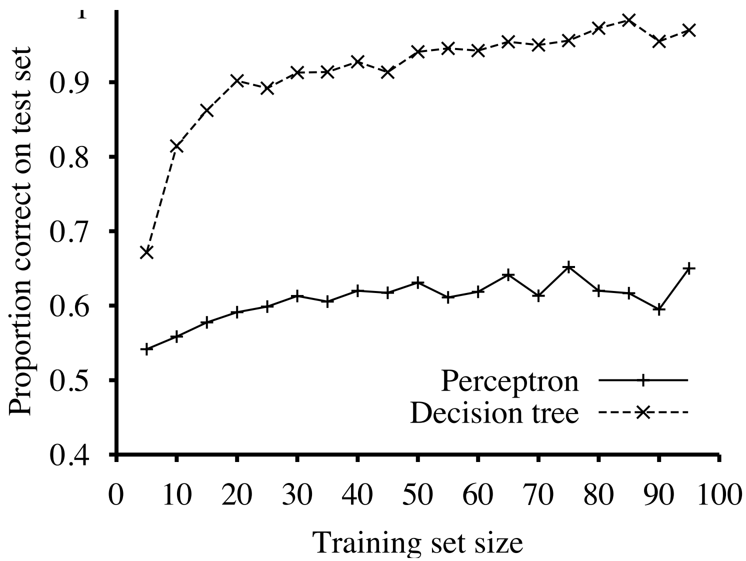


Works well on some problems



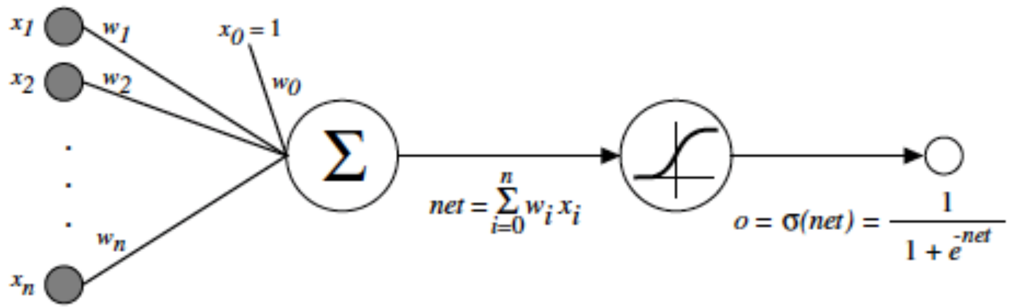
Learning curves

Are majority of inputs 1?



Restaurant example: WillWait?

Sigmoid Unit



$\sigma(x)$ is the sigmoid function

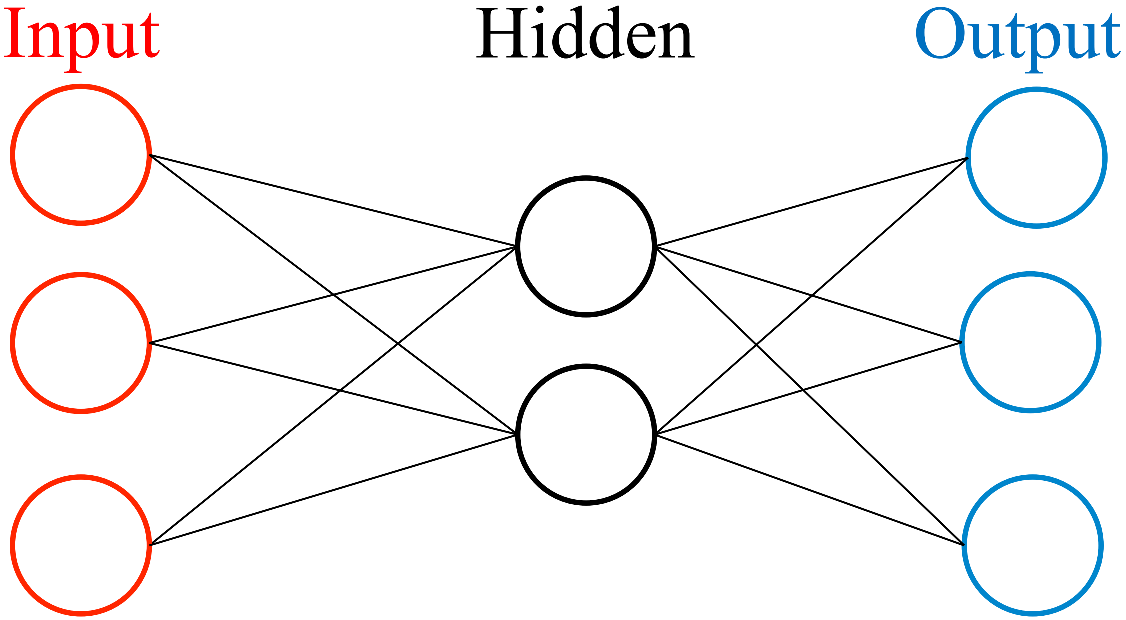
$$\frac{1}{1 + e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

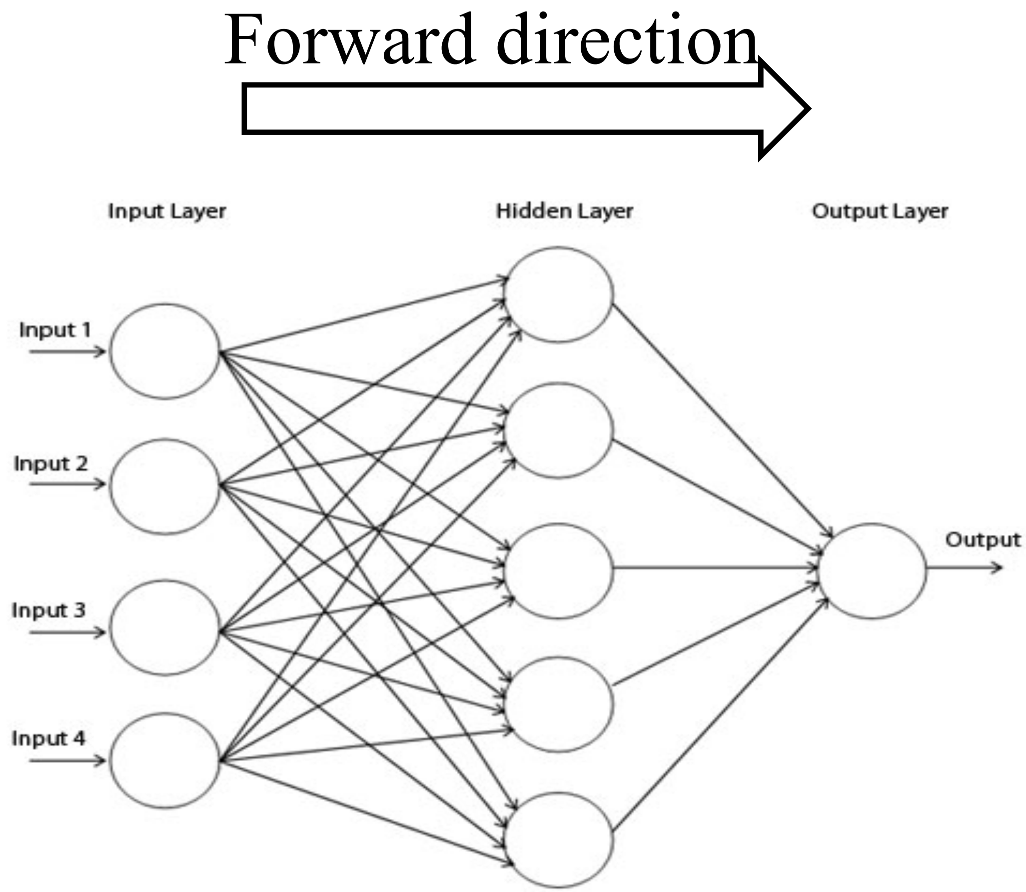
We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units $\rightarrow$ Backpropagation

Multilayer Networks



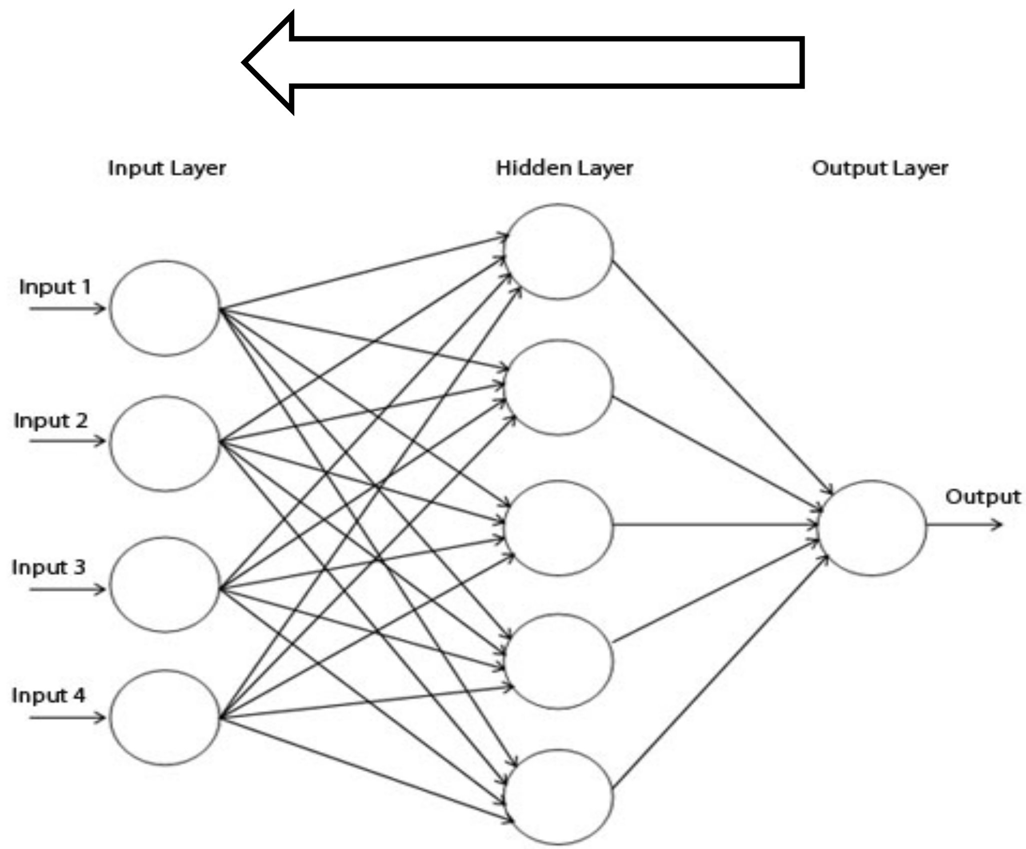
Backpropagation Algorithm



Calculate network and error

Backpropagation Algorithm

Backward direction

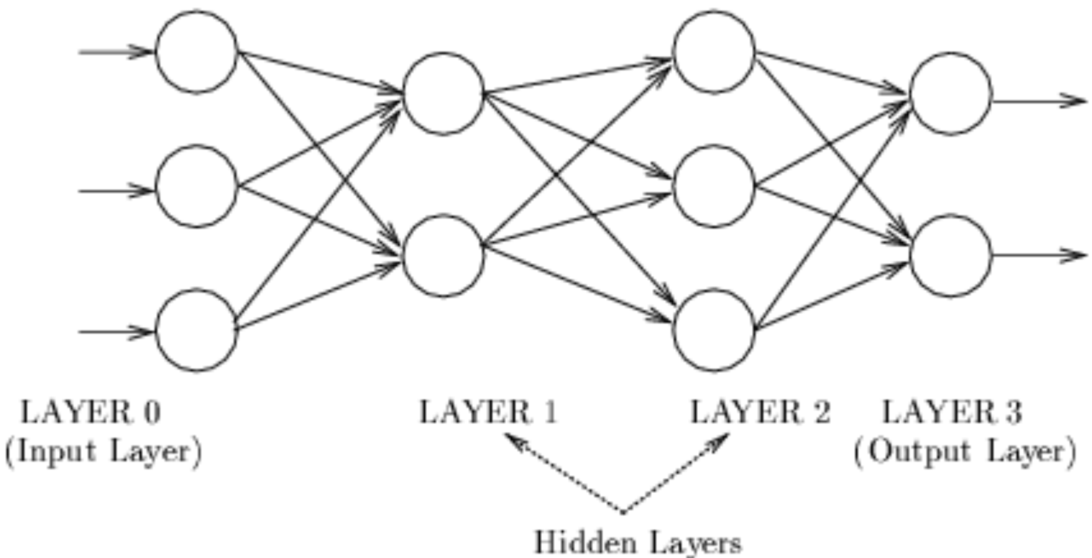


Backpropagate: from output to input, recursively compute $\partial E/\partial w_{ij} = \nabla_w E$ and adjust weights

Network Architecture: Feedforward net

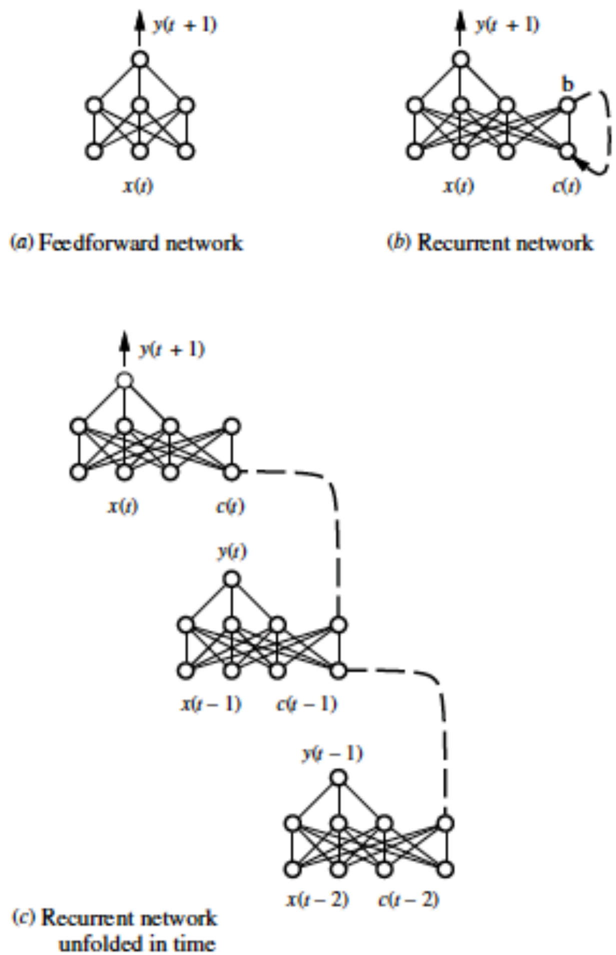
– A connection is allowed from a node in layer *i* only to nodes in layer *i* + 1.

– Most widely used architecture.



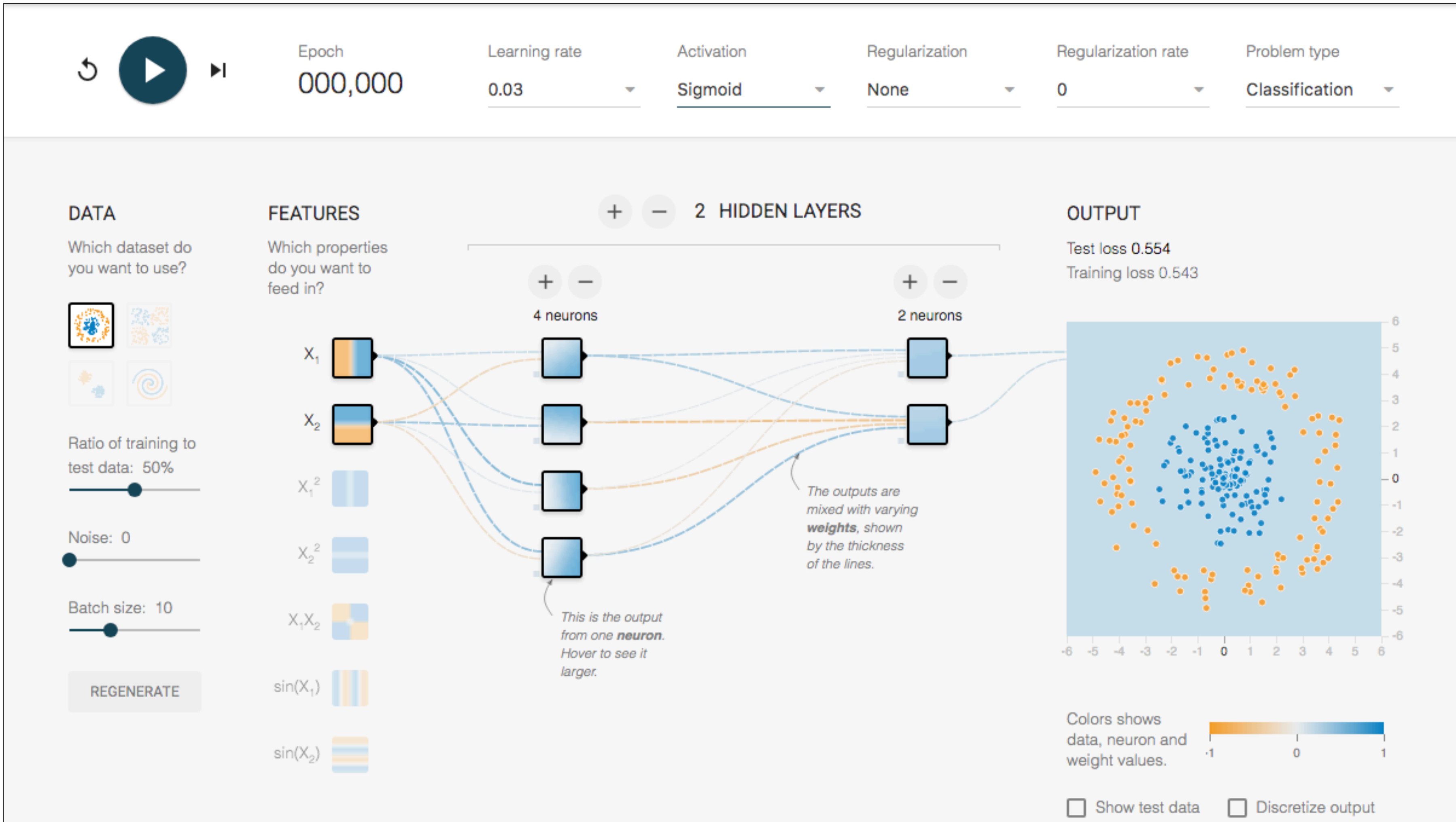
Conceptually, nodes at higher levels successively abstract features from preceding layers

Recurrent neural networks



- Good for learning sequences of data
- e.g., text
- Lots of variations today: convoluted NNs, LSTMs, …

Neural network playground



http://playground.tensorflow.org/