Machine Learning – COMS3007

Neural Networks

(Representations)

Benjamin Rosman

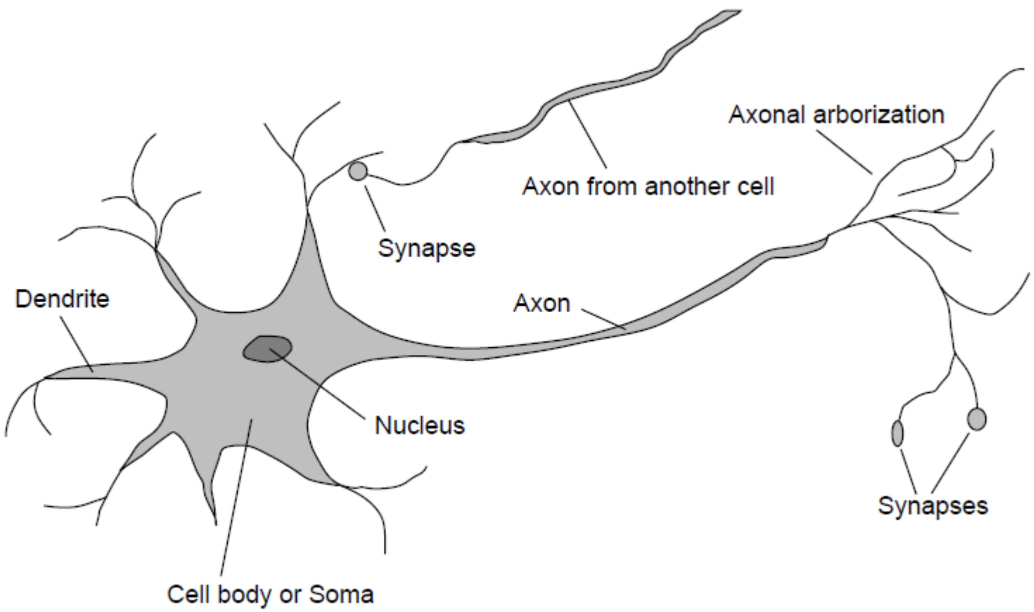
Benjamin.Rosman1@wits.ac.za / benjros@gmail.com

April 2018

Based heavily on course notes by Geoffrey Hinton, Chris Williams and Victor Lavrenko, Amos Storkey, Eric Eaton, and Clint van Alten

A biological perspective

- Interest in how the brain works (as a computing machine)
  - Human brain: 10^{11} neurons, 10^{14} synapses (connections)
  - Computers are much faster
    - But there are many more neurons
    - Neurons operate in parallel



A biological perspective

- Neurons send signals to each other

- Different parts of the brain do different things

- “One learning algorithm” hypothesis
  - Different areas learn in the same way

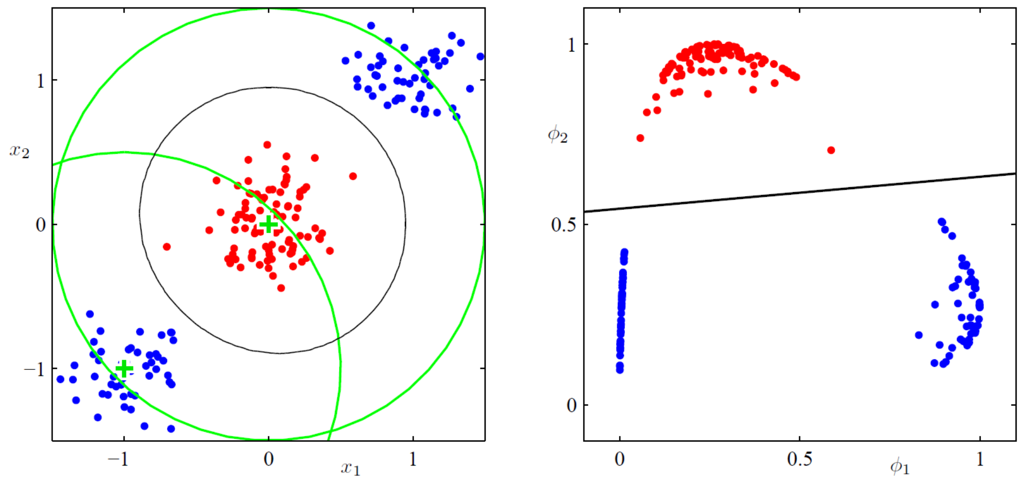
- Connection strength (synaptic weights) can change
  - Learning: perform useful computations
  - Strengths modulate the effects of signals between neurons
    - Excitatory vs inhibitory

Artificial neural networks

- Idealised models of how natural neurons work
- Allows us to study:
  - How the brain works
  - How to do computations in parallel rather than sequentially
  - How to solve complex problems

An engineering perspective

- We’ve spoken a lot about the importance of choosing the right features $\phi(x)$:

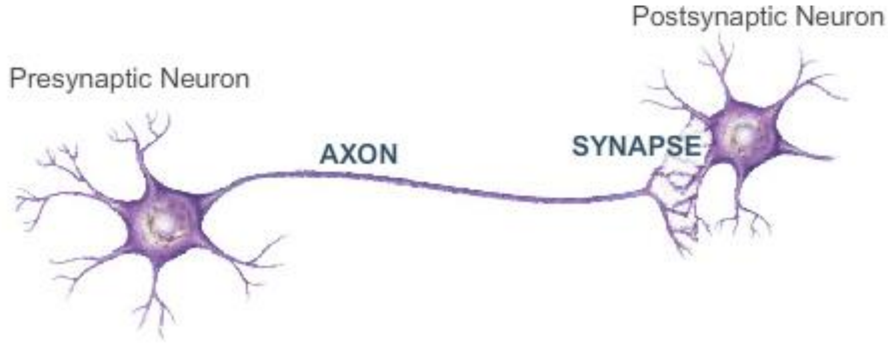


- Feature engineering (using domain expertise) has always been the **most important aspect**
  - But can’t we learn these features as well as the weights?

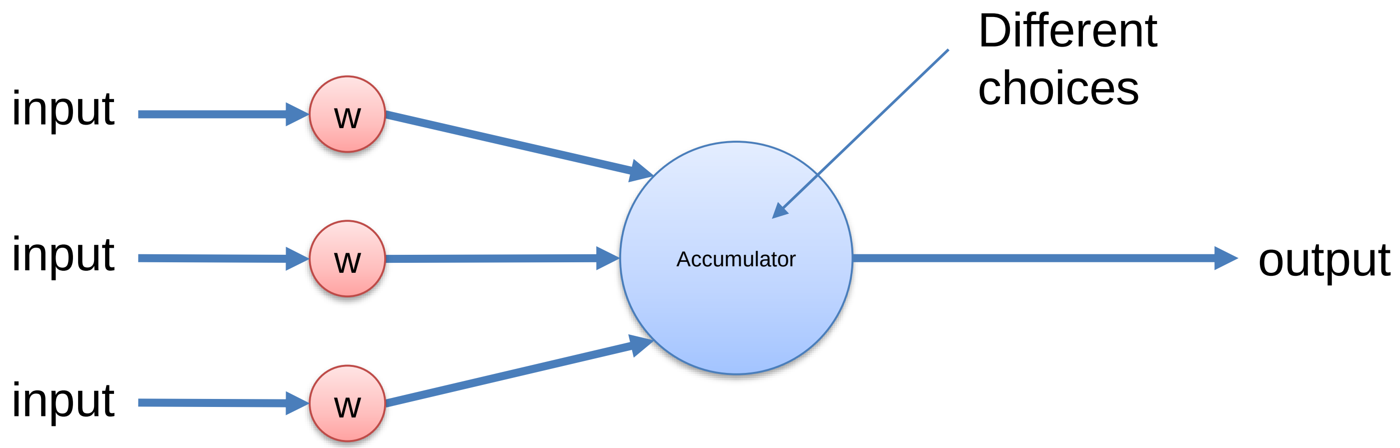
A brief history

- 1943: first artificial neuron: threshold logic unit
  - Warren McCulloch and Walter Pitts
- 1949: weights (synapses) adapt during learning
  - Donald Hebb
- 1957: the perceptron (Frank Rosenblatt)
- 1969: “Perceptrons” (Minsky and Papert)
- 1970s: the AI winter
- 1975: backpropagation – learning in networks (Paul Werbos)
- 1986: backprop rediscovered
  - David Rumelhart and Geoffrey Hinton
- 1980s – 2006: difficult to train networks
- 2006: new strategies for training networks
  - Geoffrey Hinton, Yann LeCun, Yoshua Bengio
- Since then: deep learning everywhere!

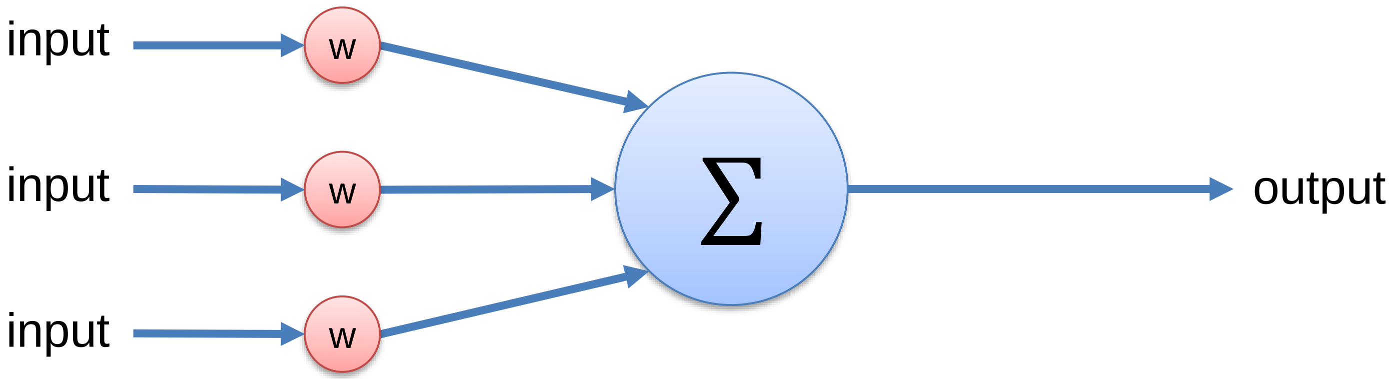
A simple neuron



- Artificial neuron:
  - Output y as a function of inputs x and weights w



Linear neurons



- Let one of these inputs be a bias $b = 1 = x_0$

Sum over all input connections (including bias)

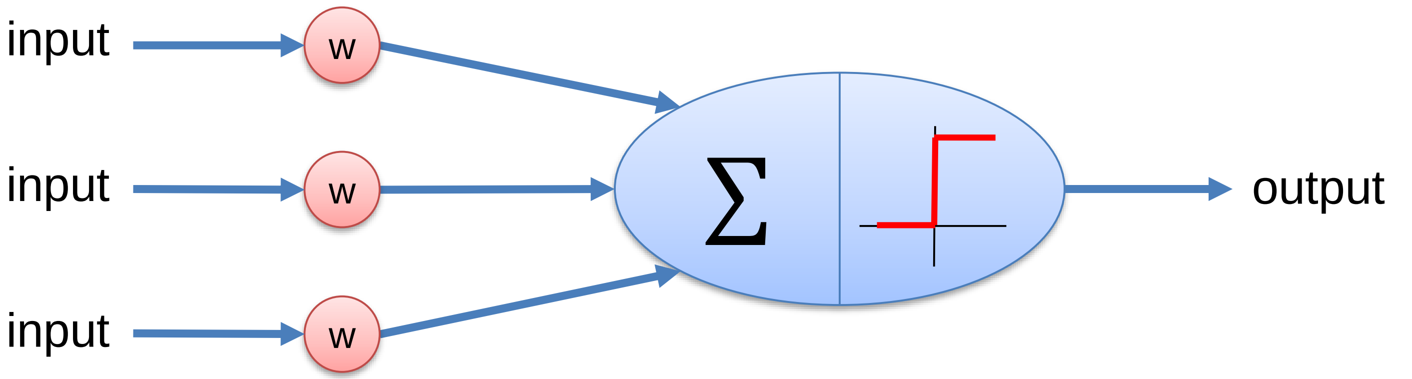
- $y = \sum_{i=0}^{d} x_i w_i$

Weight on i^{th} input

i^{th} input

- But this is just linear regression!

Binary threshold neurons



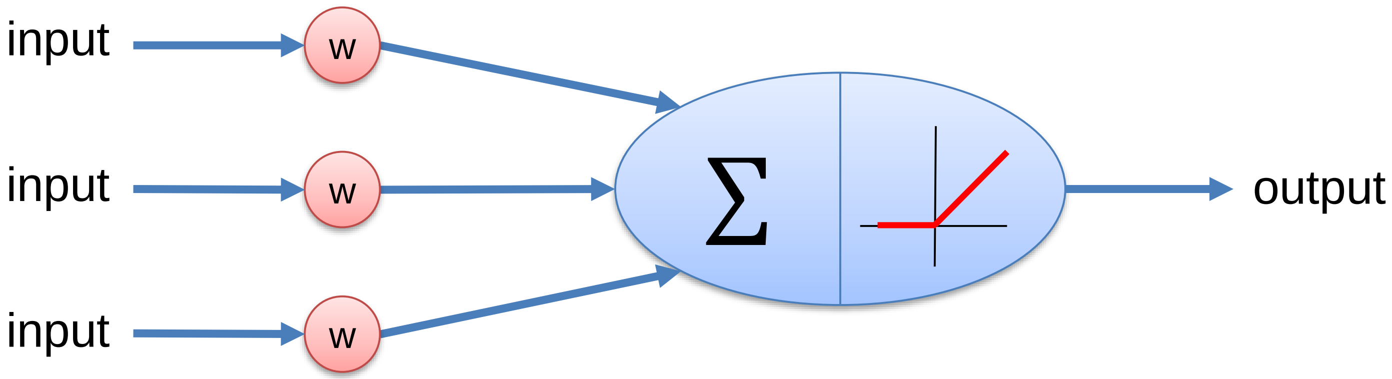
- Use a step function (threshold function)

- $z = \sum_{i=0}^{d} x_i w_i$
- $y = \begin{cases} 1 \; if \; z \geq \theta \\ 0 \; otherwise \end{cases}$

Threshold (could be 0)

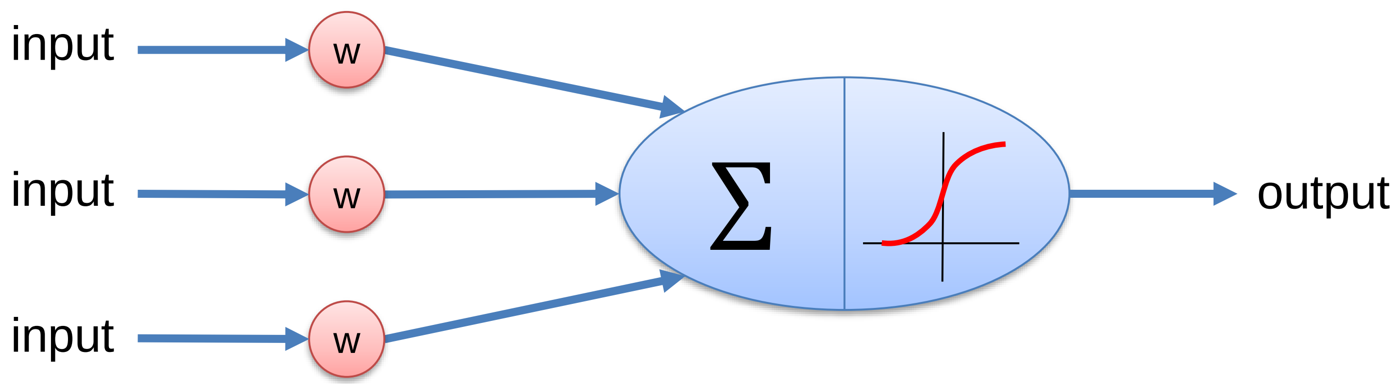
- But this is the perceptron!

Rectified linear neurons (ReLU)



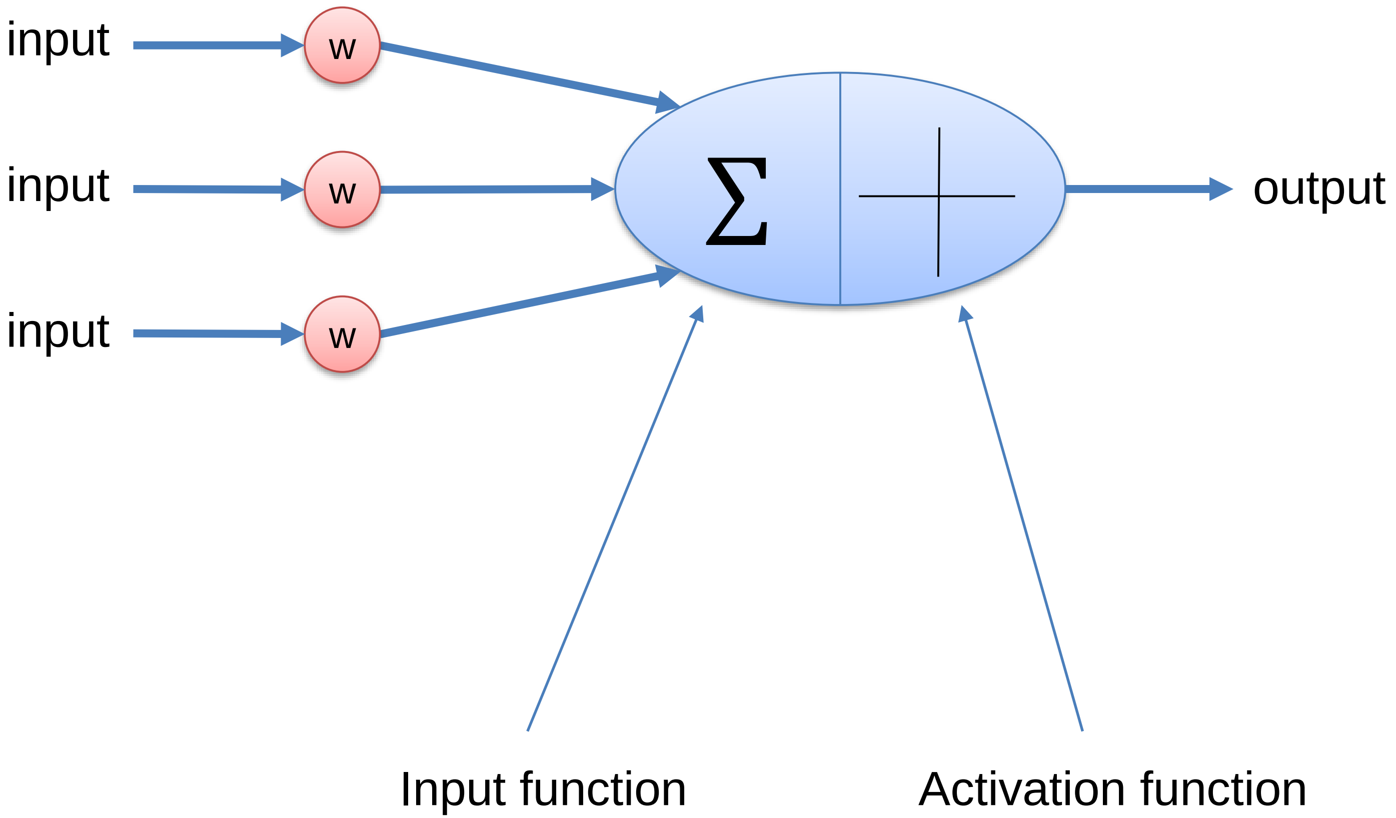
- Somewhere between a linear and threshold unit
- $z = \sum_{i=0}^{d} x_i w_i$
- $y = \begin{cases} z \ if \ z \geq 0 \\ 0 \ otherwise \end{cases}$

Sigmoid/logistic neurons



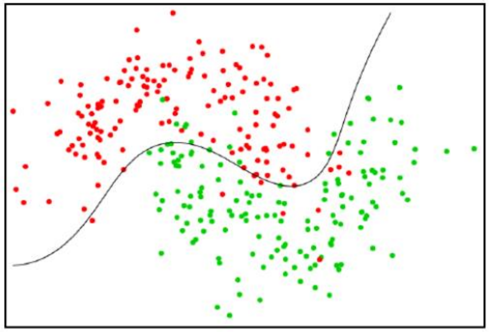
- Use a logistic function
- $z = \sum_{i=0}^{d} x_i w_i$
- $y = \frac{1}{1+\mathrm{e}^{-z}}$
- But this is logistic regression!

Neuron anatomy



- Intuition: stack these neurons to learn features!

Learning features



- Logistic regression performs classification
  - Only with linear decision boundaries!

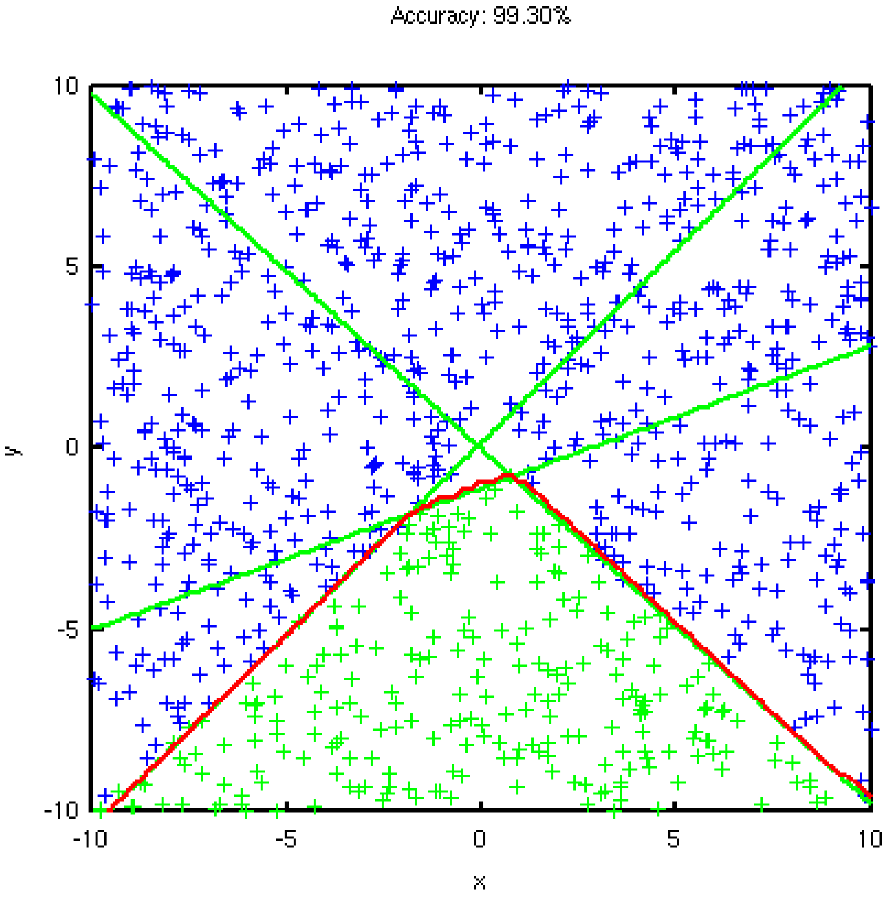
- To model more complicated systems, we need nonlinear class boundaries

- Use features $\phi(x)$ to get these nonlinear decision boundaries. But what features?
  - Ideally: features that apply to different parts of the input space

- Logistic regression model divides input space into two!
  - Let each feature $\phi_j(x)$ be a logistic regression model (neuron)

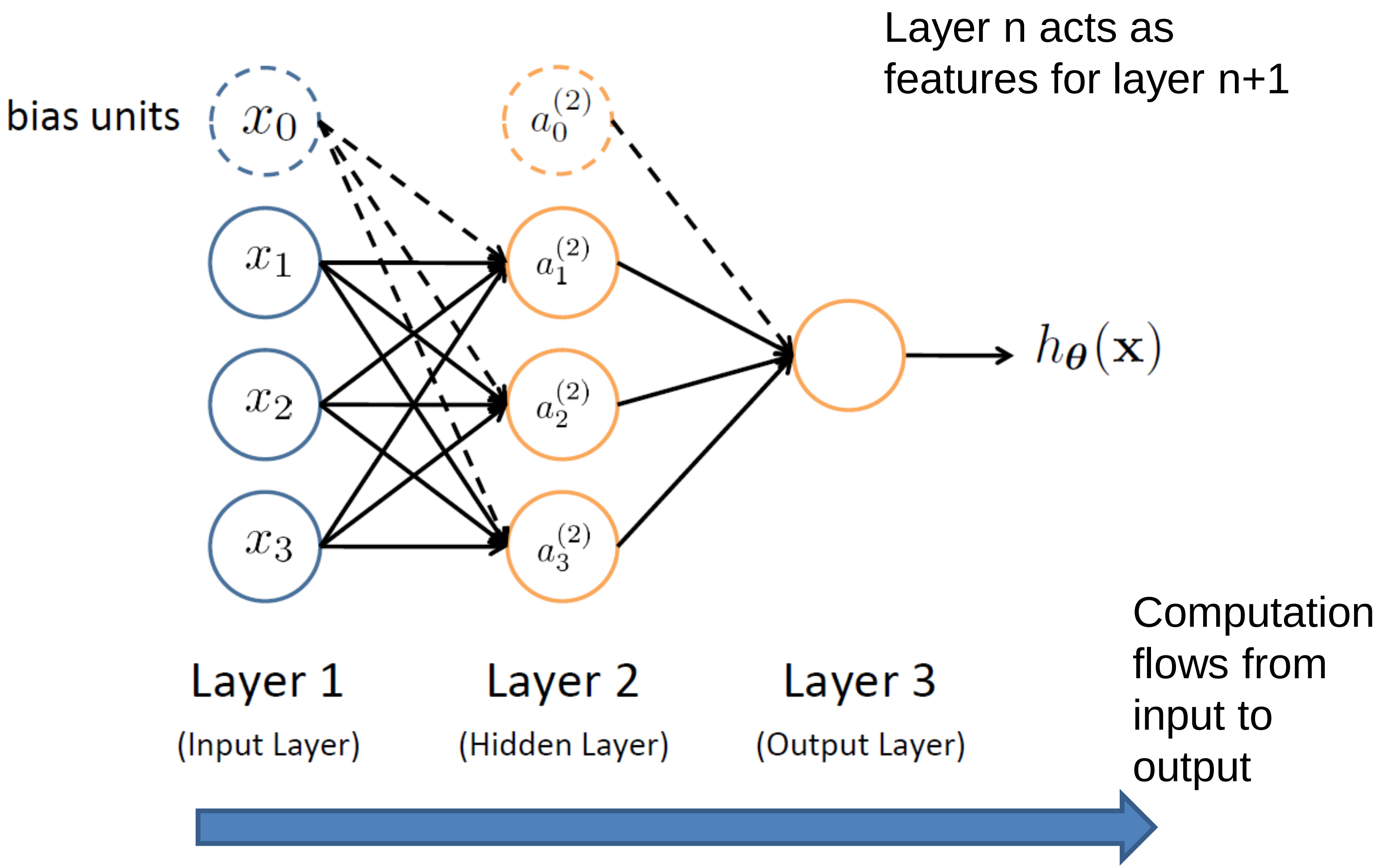
Learning features

- Logistic regression model divides input space into two!
    - Let each feature $\phi_i(x)$ be a logistic regression model (neuron)



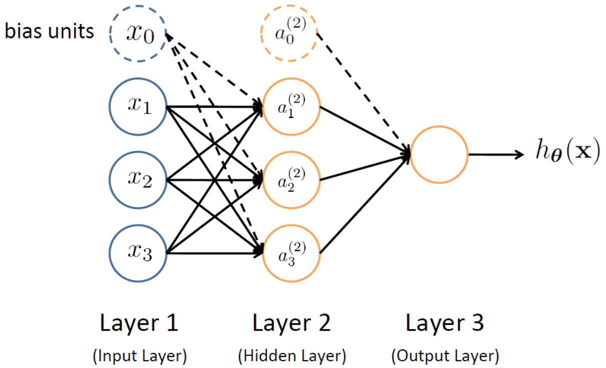
Stacking neurons

- Neurons arranged in layers



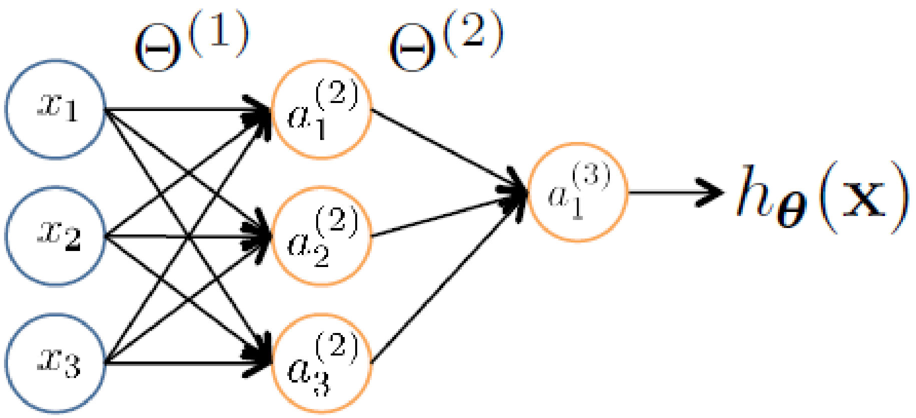
Feed-forward networks

Sometimes called multilayer perceptrons (MLPs)



- Input layers
  - Raw data
  - As provided by sensor measurements
- Feed-forward networks (most common)
  - Outputs from one layer become inputs to the next
- Working forward through the network:
  - Apply input function to compute total input
    - Usually just the sum of inputs
  - Activation function transforms input to final value
    - Usually nonlinear function
- Output layer: computation target

Computations



- $a_i^{(j)}$ = activation of unit i in layer j
- $\Theta^{(j)}$ = weight matrix: mapping from layer j to j+1

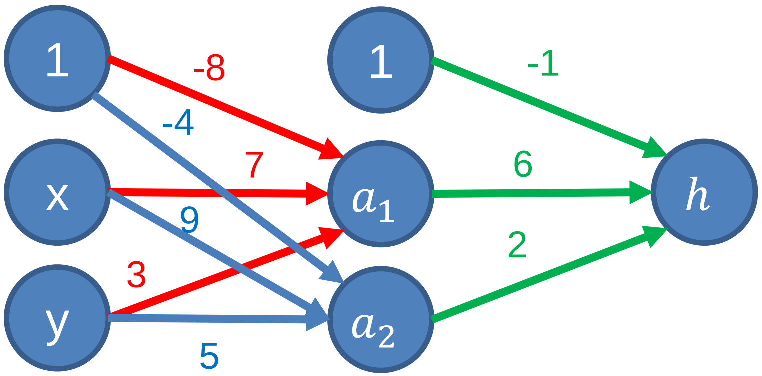
Bias = 1

- $a_1^{(2)} = g\left(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3\right)$
- $a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right)$
- $a_3^{(2)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right)$

- $h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right)$

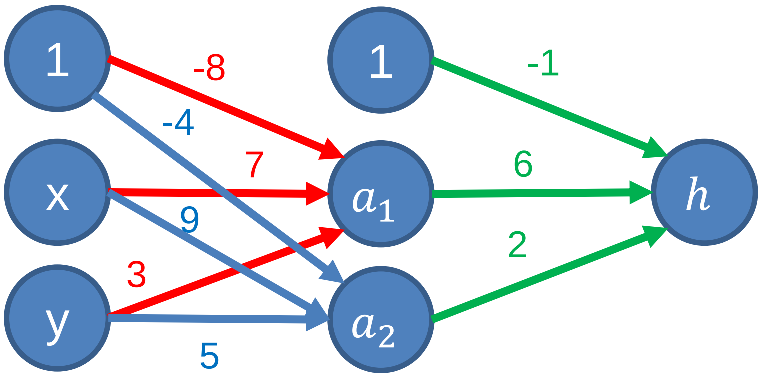
- If s_j units in layer j, s_{j+1} in layer j+1: $\dim\left(\Theta^{(j)}\right) = s_{j+1} \times (s_j + 1)$

Example



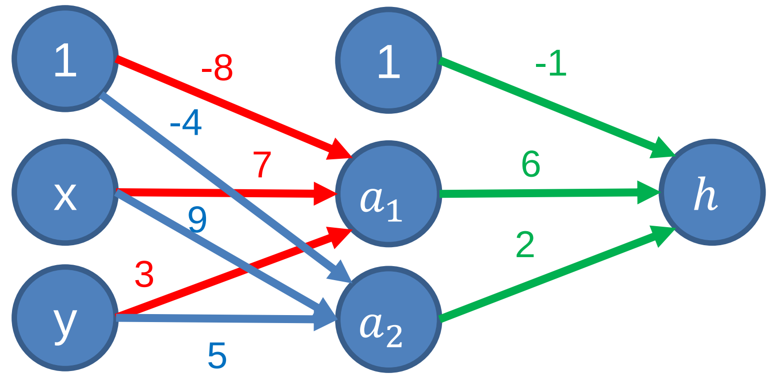
- Assume all activation functions are $g(\,)$
- Output?
- $h = g(6a_1 + 2a_2 - 1)$
- $h = g(6g(7x + 3y - 8) + 2g(9x + 5y - 4) - 1)$

Example



- Weight matrix (layer 1) = $W^{(1)} = \begin{bmatrix} -8 & 7 & 3 \\ -4 & 9 & 5 \end{bmatrix}$
- Weight matrix (layer 2) = $W^{(2)} = \begin{bmatrix} -1 & 6 & 2 \end{bmatrix}$
- Assume all activation functions are ReLUs
- E.g. Input: $x = (1.5, -1)^T$
- Output?
  - Compute a_1, a_2 and then propagate forward to h

Example

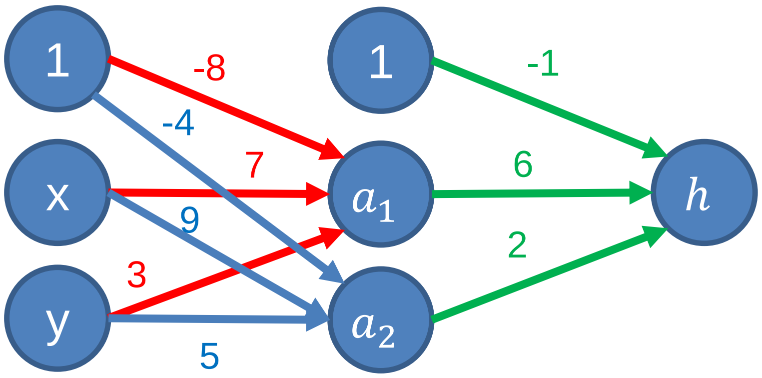


- $W^{(1)} = \begin{bmatrix} -8 & 7 & 3 \\ -4 & 9 & 5 \end{bmatrix}$, $W^{(2)} = \begin{bmatrix} -1 & 6 & 2 \end{bmatrix}$
- All activation functions are ReLUs
- Input: $x = (1.5, -1)^T$ → augment to $x = (1, 1.5, -1)^T$

- Total input to a_1:
  - $\sum_{i=1}^{3} w_{1i}^{(1)} x_i = (-8)(1) + (7)(1.5) + (3)(-1) = -0.5$
- Activation at a_1 (ReLU):
  - $-0.5 < 0$ so output $a_1 = 0$

- Total input to a_2:
  - $\sum_{i=1}^{3} w_{2i}^{(1)} x_i = (-4)(1) + (9)(1.5) + (5)(-1) = 4.5$
- Activation at a_2 (ReLU):
  - $4.5 > 0$ so output $a_2 = 4.5$

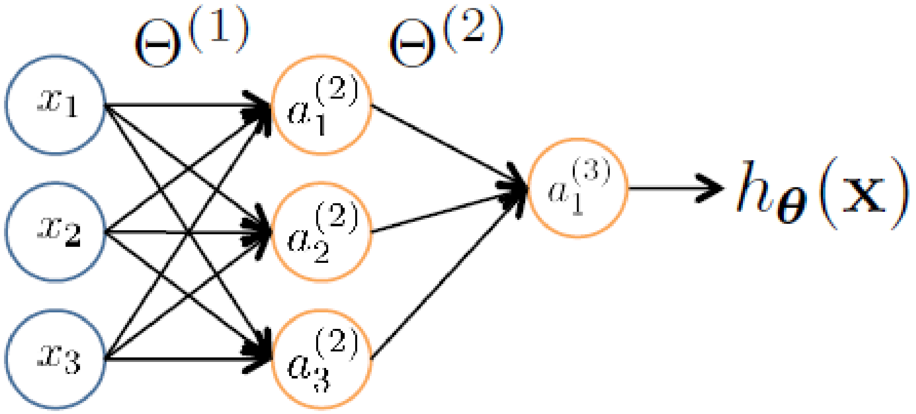
Example



- $W^{(1)} = \begin{bmatrix} -8 & 7 & 3 \\ -4 & 9 & 5 \end{bmatrix}, W^{(2)} = \begin{bmatrix} -1 & 6 & 2 \end{bmatrix}$
- All activation functions are ReLUs
- $a_1 = 0$
- $a_2 = 4.5$
- Augment with $a_0 = 1$: $a = (1, 0, 4.5)^T$

- Total input to h:
  - $\sum_{i=1}^{3} w_i^{(2)} a_i = (-1)(1) + (6)(0) + (2)(4.5) = 8$
- Activation at h (ReLU):
  - $8 > 0$ so output $h = 8$

Vectorization

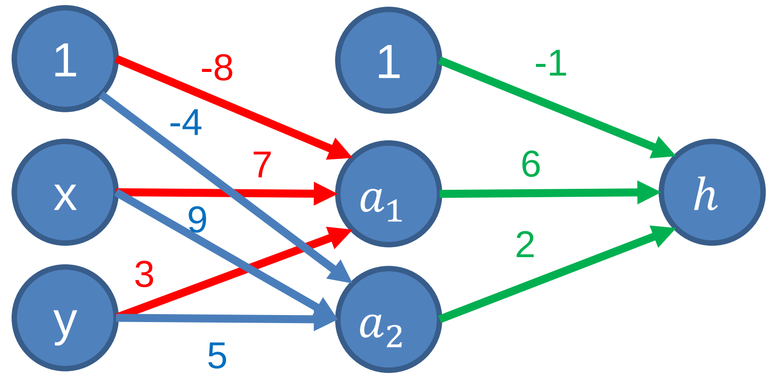


- $a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g(z_1^{(2)})$
- $a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g(z_2^{(2)})$
- $a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g(z_3^{(2)})$
- $h_\Theta(x) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g(z_1^{(3)})$

Vectorized steps:

- $\boldsymbol{z}^{(2)} = \Theta^{(1)}\boldsymbol{x}$
- $\boldsymbol{a}^{(2)} = g\left(\boldsymbol{z}^{(2)}\right)$
- Add $a_0^{(2)} = 1$
- $\boldsymbol{z}^{(3)} = \Theta^{(2)}\boldsymbol{a}^{(2)}$
- $h_\Theta(\boldsymbol{x}) = \boldsymbol{a}^{(3)} = g(\boldsymbol{z}^{(3)})$

Example

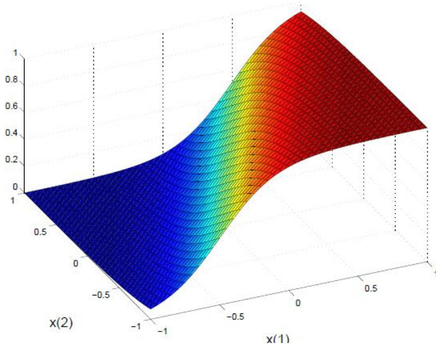


- $W^{(1)} = \begin{bmatrix} -8 & 7 & 3 \\ -4 & 9 & 5 \end{bmatrix}$, $W^{(2)} = \begin{bmatrix} -1 & 6 & 2 \end{bmatrix}$
- All activation functions are ReLUs
- Input: $x = (1.5, -1)^T$ → augment to $x = (1, 1.5, -1)^T$

- Total input $z^{(2)}$ at layer 2:
  - $z^{(2)} = W^{(1)}x = \begin{bmatrix} -8 & 7 & 3 \\ -4 & 9 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1.5 \\ -1 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 4.5 \end{bmatrix}$
- Apply ReLU operator: $a^{(2)} = g\left(z^{(2)}\right) = (0, 4.5)^T$
- Augment with $a_0 = 1$: $a = (1, 0, 4.5)^T$
- Total input $z^{(3)}$ at layer 3:
  - $z^{(3)} = W^{(2)}a = \begin{bmatrix} -1 & 6 & 2 \end{bmatrix}(1, 0, 4.5)^T = 8$
- Apply ReLU operator: $h = g\left(z^{(3)}\right) = 8$

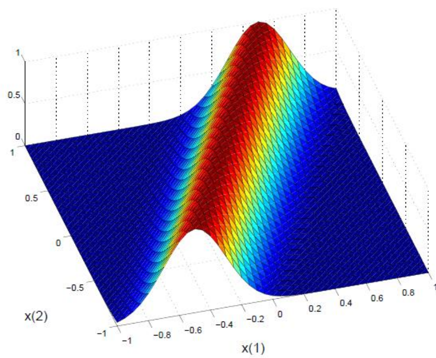
Hidden neurons

- If each neuron produces a sigmoid:

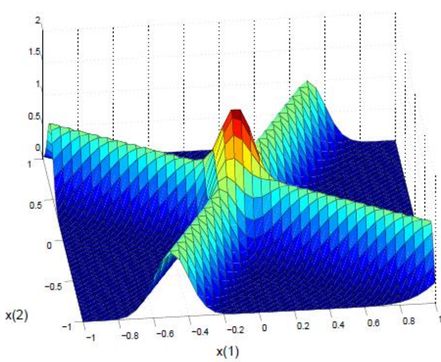


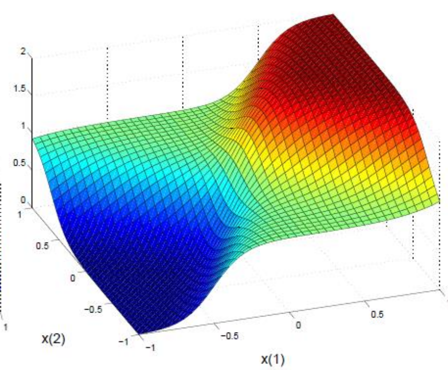
- Two can combine to give a ridge:

  (in the next layer)



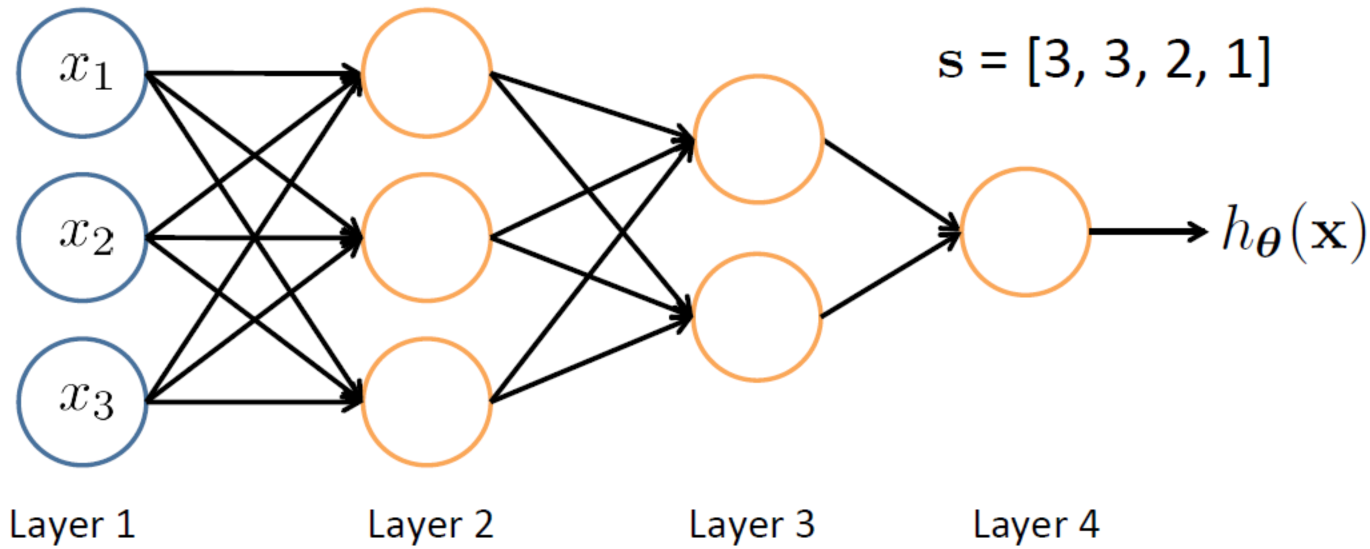
- These can create more complex structures:





Architectures

- Can build many different architectures
- E.g.



- Let L be the number of layers, s_i = nodes on layer i
  - Hyperparameters
- Usually: $s_0 = d$ (# input features), $s_{L-1} = K$ (# classes)

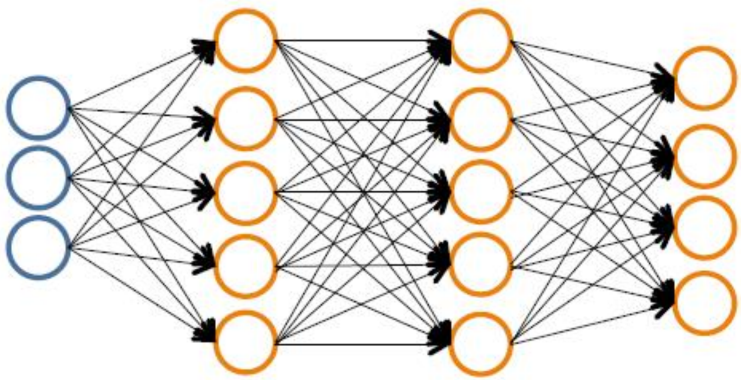
Multiclass classification

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

when car

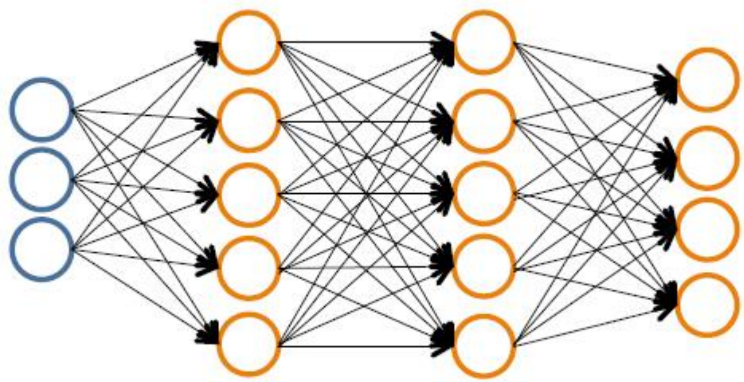
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

when motorcycle

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when truck

Multiclass classification



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

when pedestrian   when car   when motorcycle   when truck

- Given data $\{(\mathbf{x}_1, y_1),\ (\mathbf{x}_2, y_2),\ \ldots,\ (\mathbf{x}_n, y_n)\}$
- Convert labels to 1-of-K representation
  - E.g. $\mathbf{y}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ when motorcycle, $\mathbf{y}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ when car, etc.

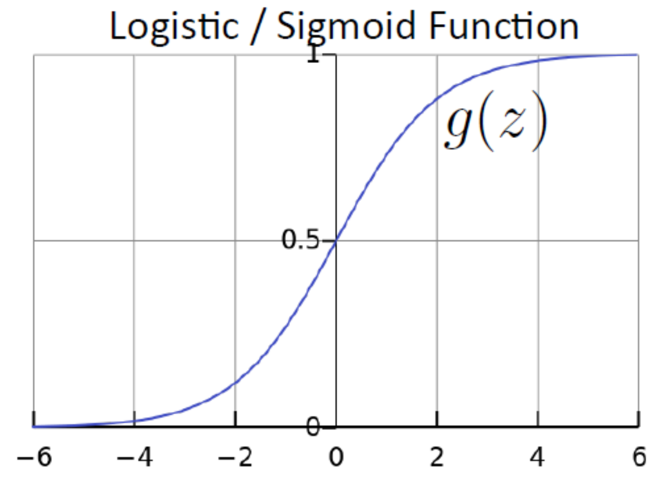
Representation power

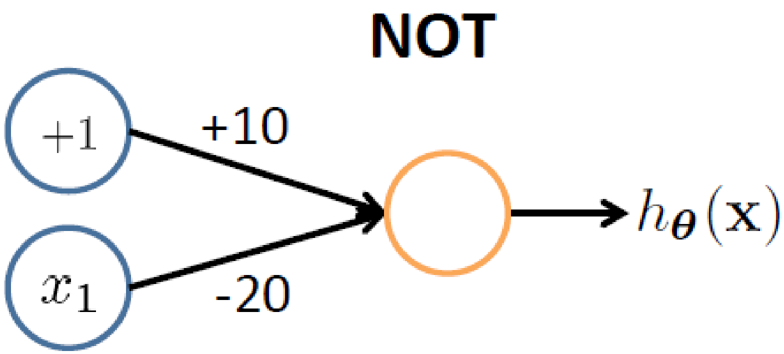
- Every Boolean function can be represented by a network with a single hidden layer
- Every bounded continuous function can be approximated with arbitrarily small error by a network with one hidden layer
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers
- Neural networks are **universal function approximators**

- But:
  - A function being representable does not tell us how many hidden units would be required
    - May be exponential!
  - Nor how easily this can be learned!

Boolean functions – NOT

- Consider NOT
- $x_1 \in \{0,1\}$
- $y = NOT\ x_1$





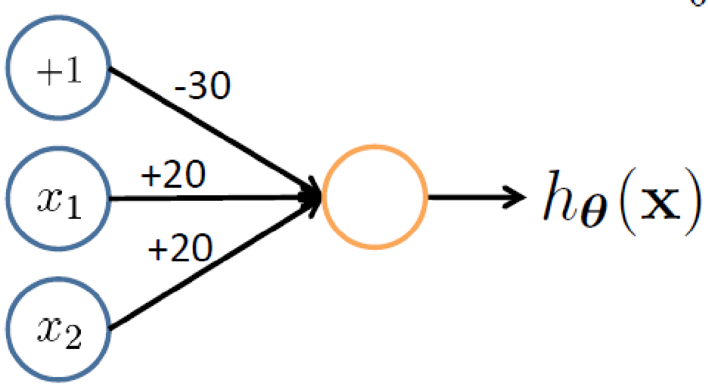
x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

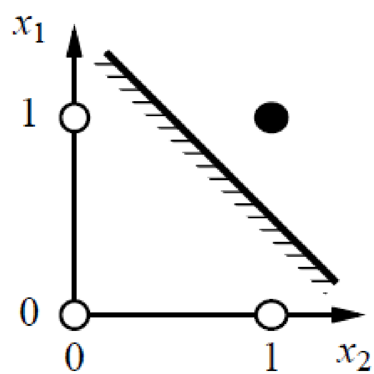
Boolean functions – AND

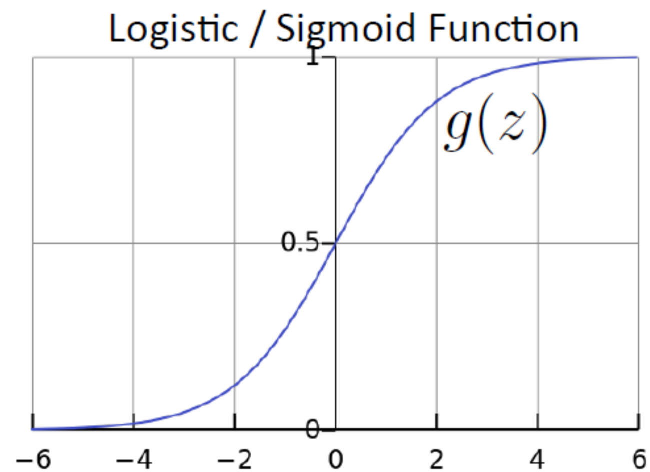
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$





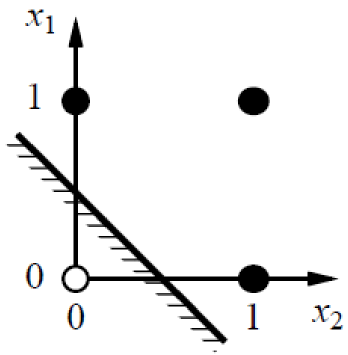


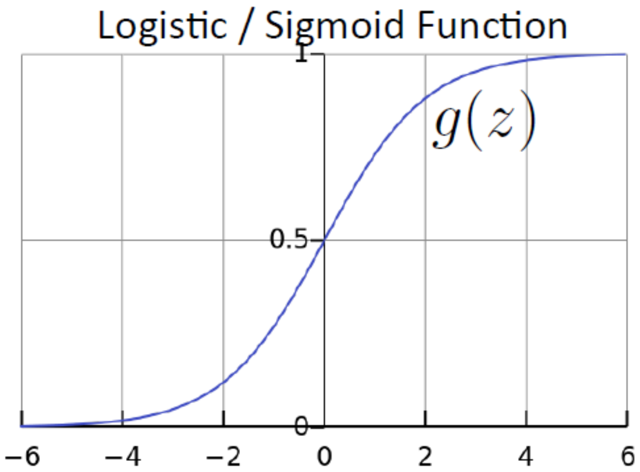
$$h_\Theta(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

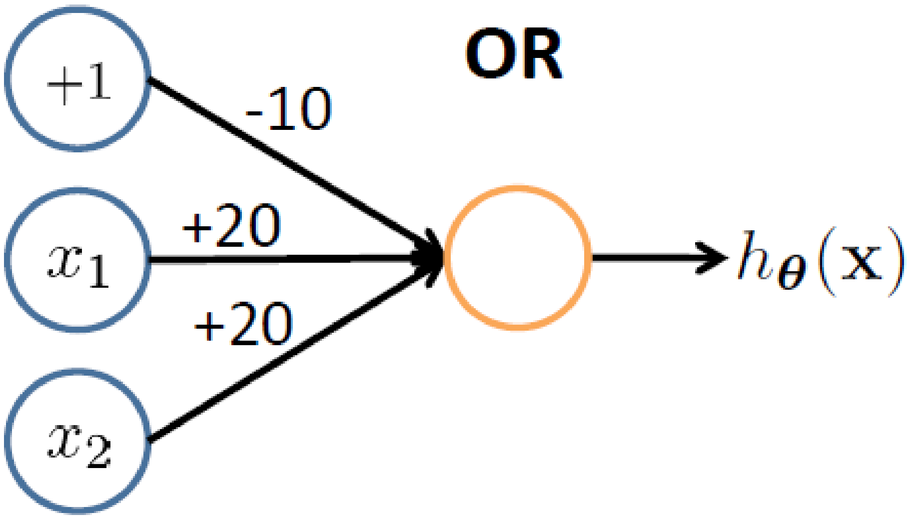
x_1	x_2	$h_\Theta(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Boolean functions – OR

- What about OR
- $x_1, x_2 \in \{0,1\}$
- $y = x_1\ OR\ x_2$

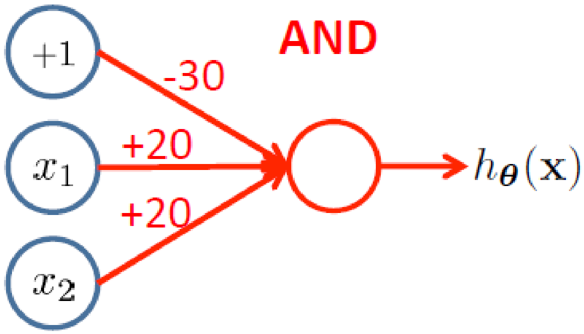


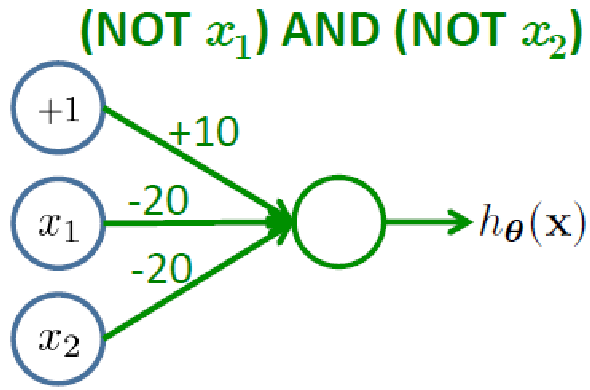


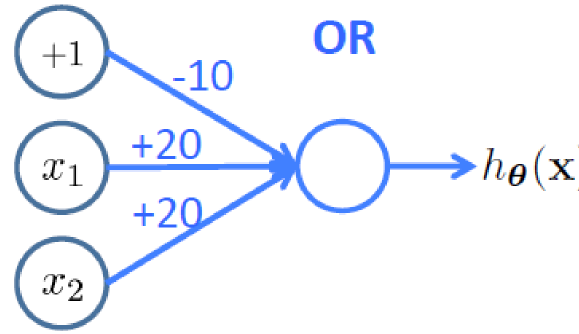


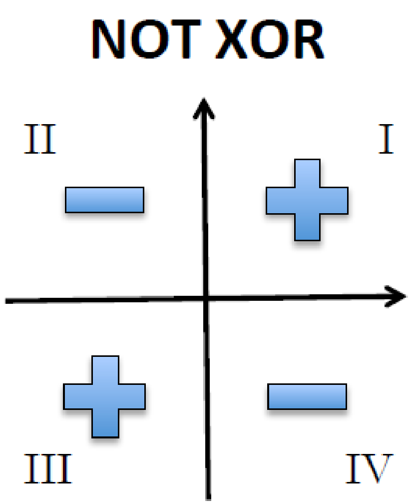
Boolean functions – NOT XOR

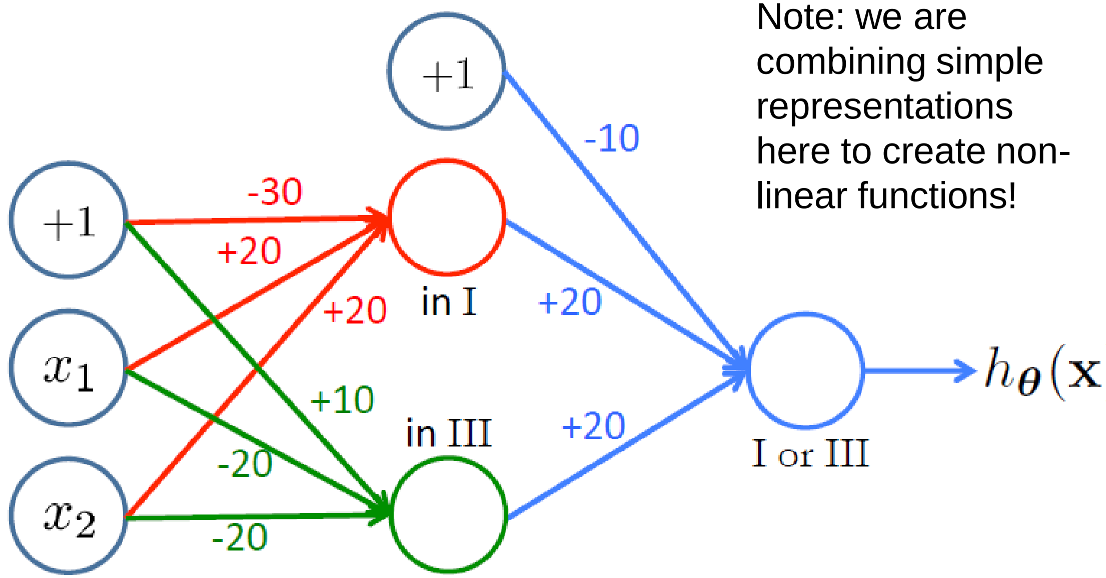
- What about NOT XOR: $y = NOT\ (x_1\ XOR\ x_2)$









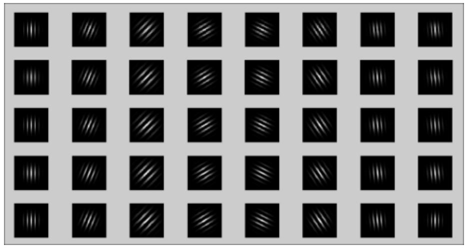


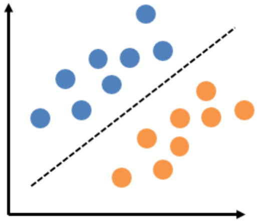
Note: we are combining simple representations here to create non-linear functions!

Feature learning: images

Traditionally:

Write a set of feature extractors:
e.g. texture and edge detectors



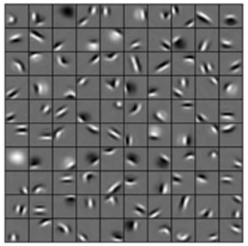


Insert into
a classifier

Neural network:

Feed raw pixels
into network

Each layer has
increasingly
abstract
representations



Final layer classifies

Recap

- Brains as motivation
- Different types of neurons
  - Relation to linear regression, logistic regression, perceptrons
- Feed-forward networks
- Hidden neurons
- Multiclass classification
- Boolean functions

- Next time: training a neural network