# Non-symbolic Machine Learning

## Neural network basics

- Russell and Norvig, Chapter 18 (18.7)

# Overview: aims

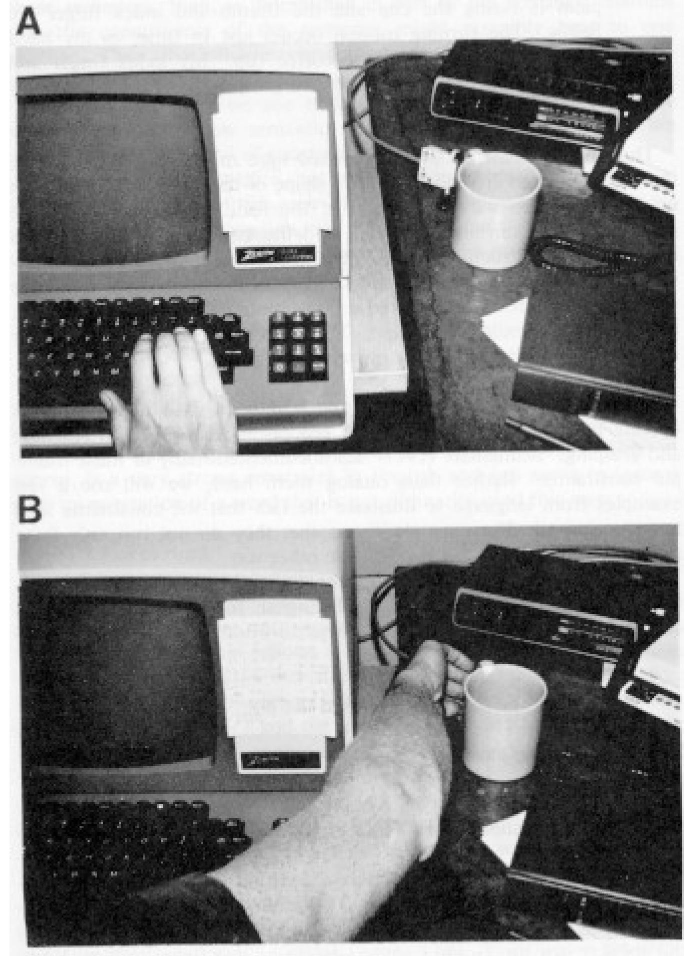
- have some idea of what neural networks are good at
- know what a neuron / unit is and how it can be formalised mathematically
- understand how and what a single-layer neural network computes, and how it stores information

# Overview: topics

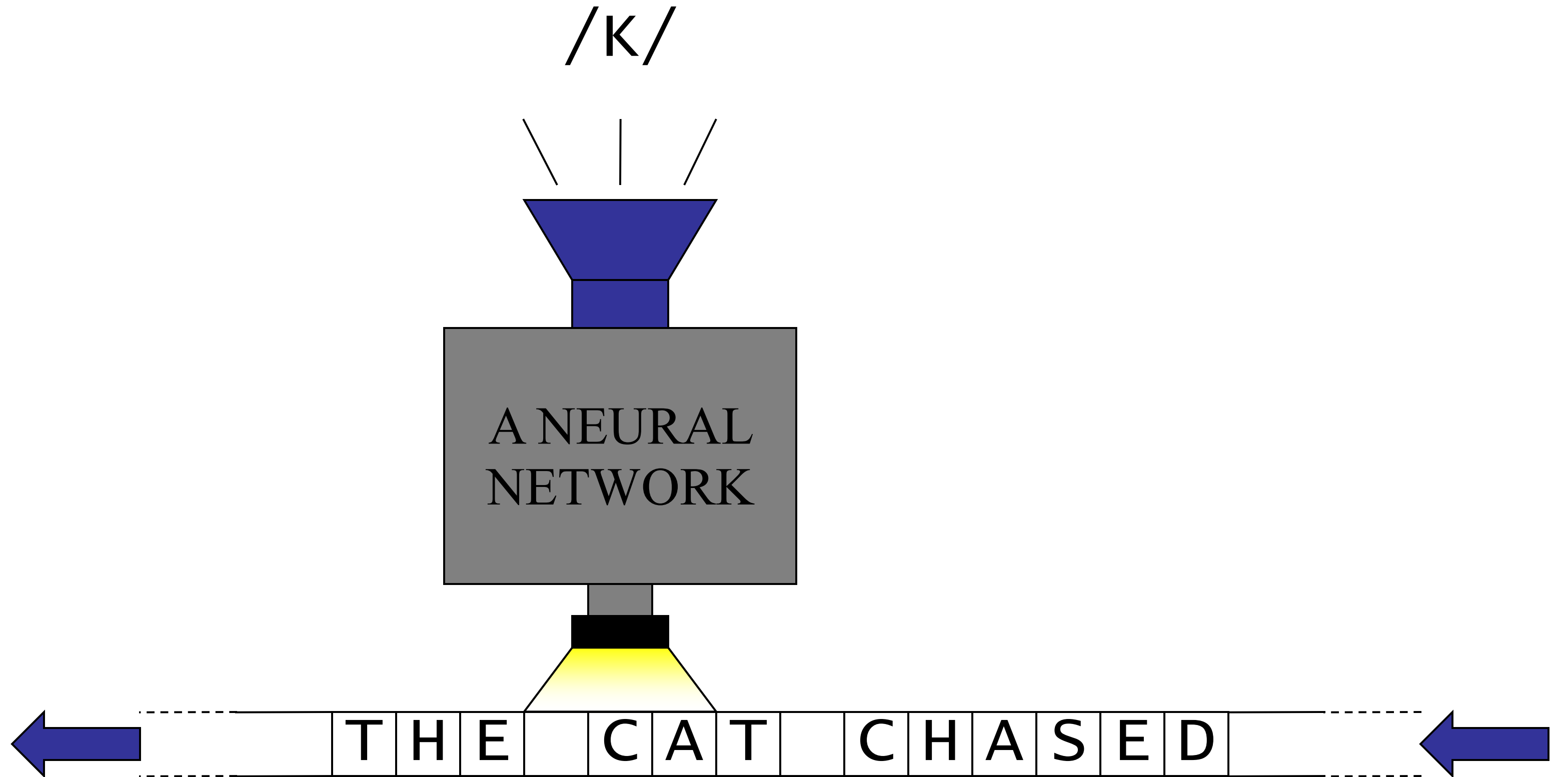
- Neural networks applications
- Brief history
- Units
- Network
- Decision boundaries
- Learning in neural networks

# Neural networks

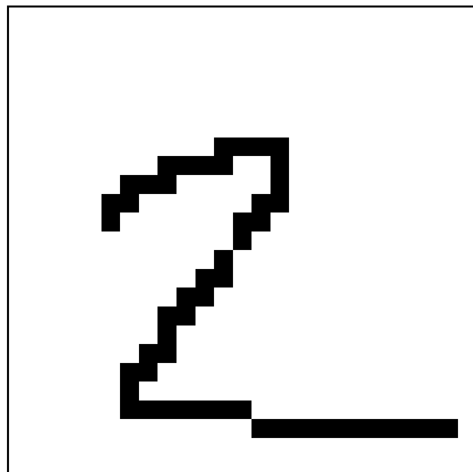
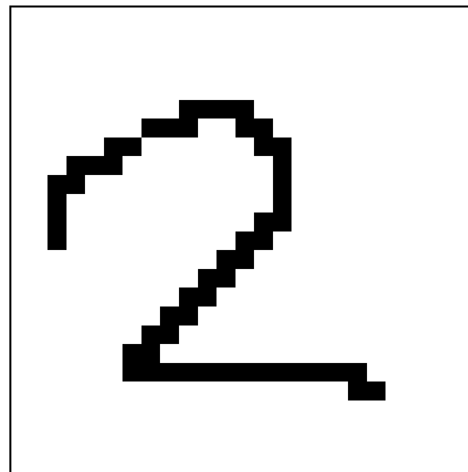
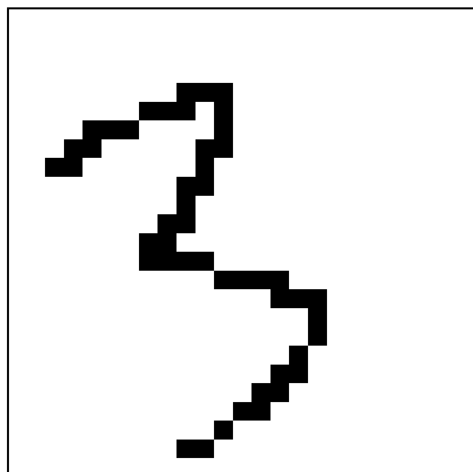
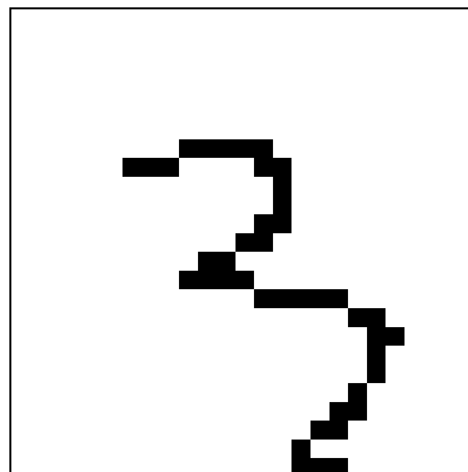
- Computational models, consisting of simple processing elements, for representing and learning functions and procedures from examples
- Crude mathematical descriptions of biological neural circuitry and functionality
- Tools for modelling cognitive phenomena
- Tools for statistical classification and regression applications



# Reading text aloud: NETtalk

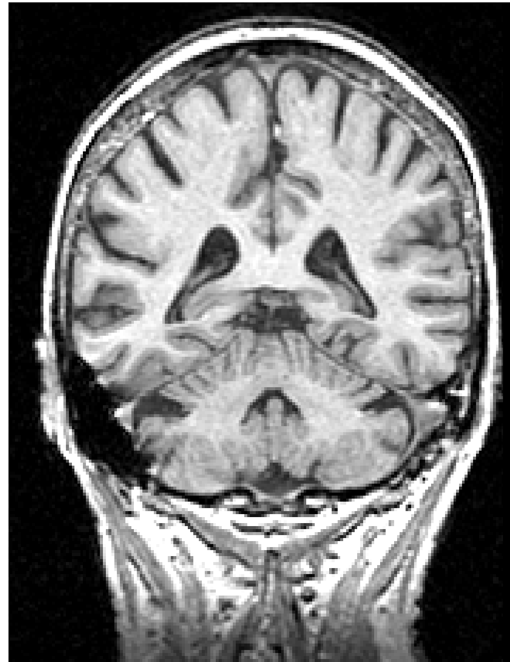


# Reading handwritten zip-codes

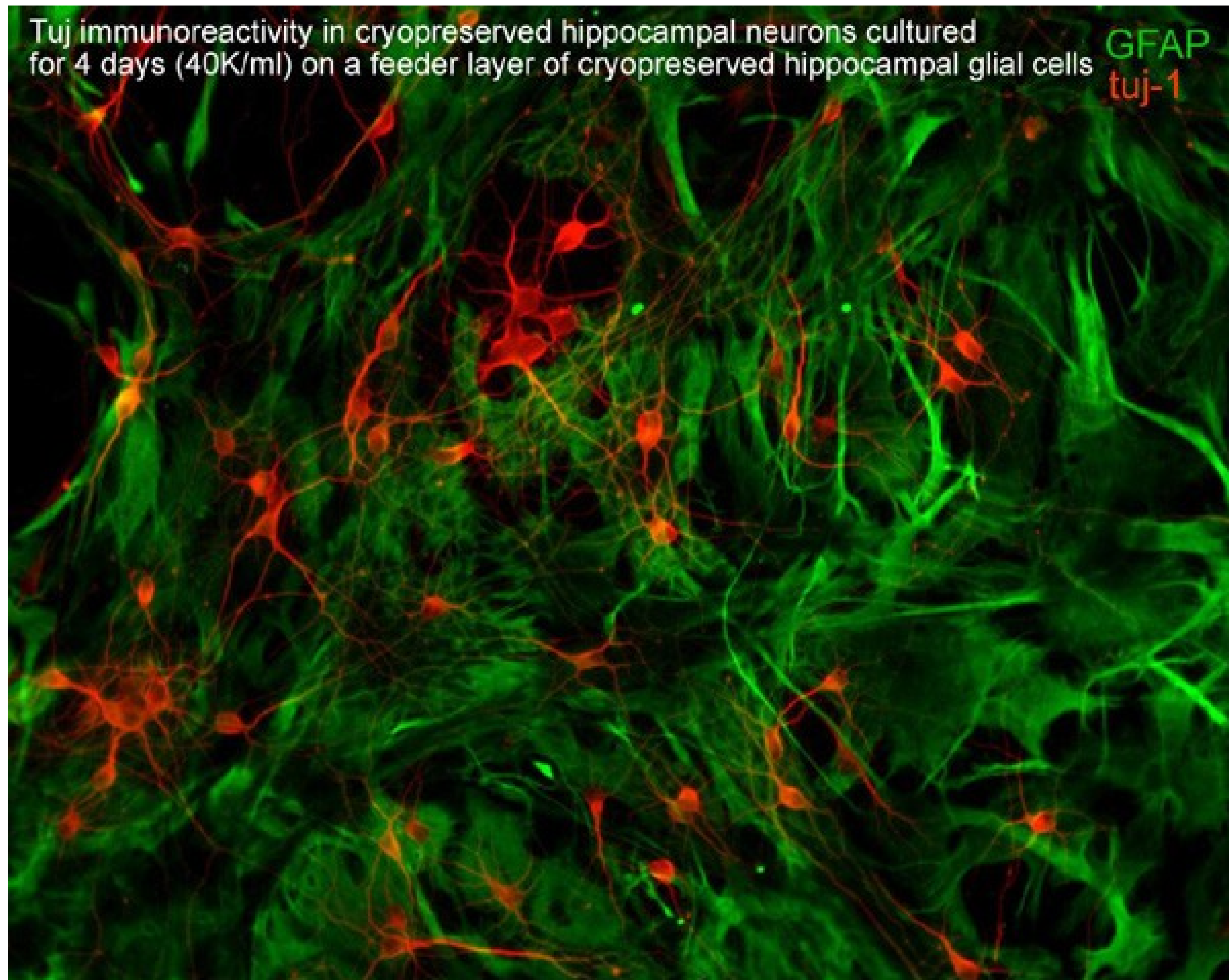





# Driving a vehicle: NavLab/ALVINN

# The brain

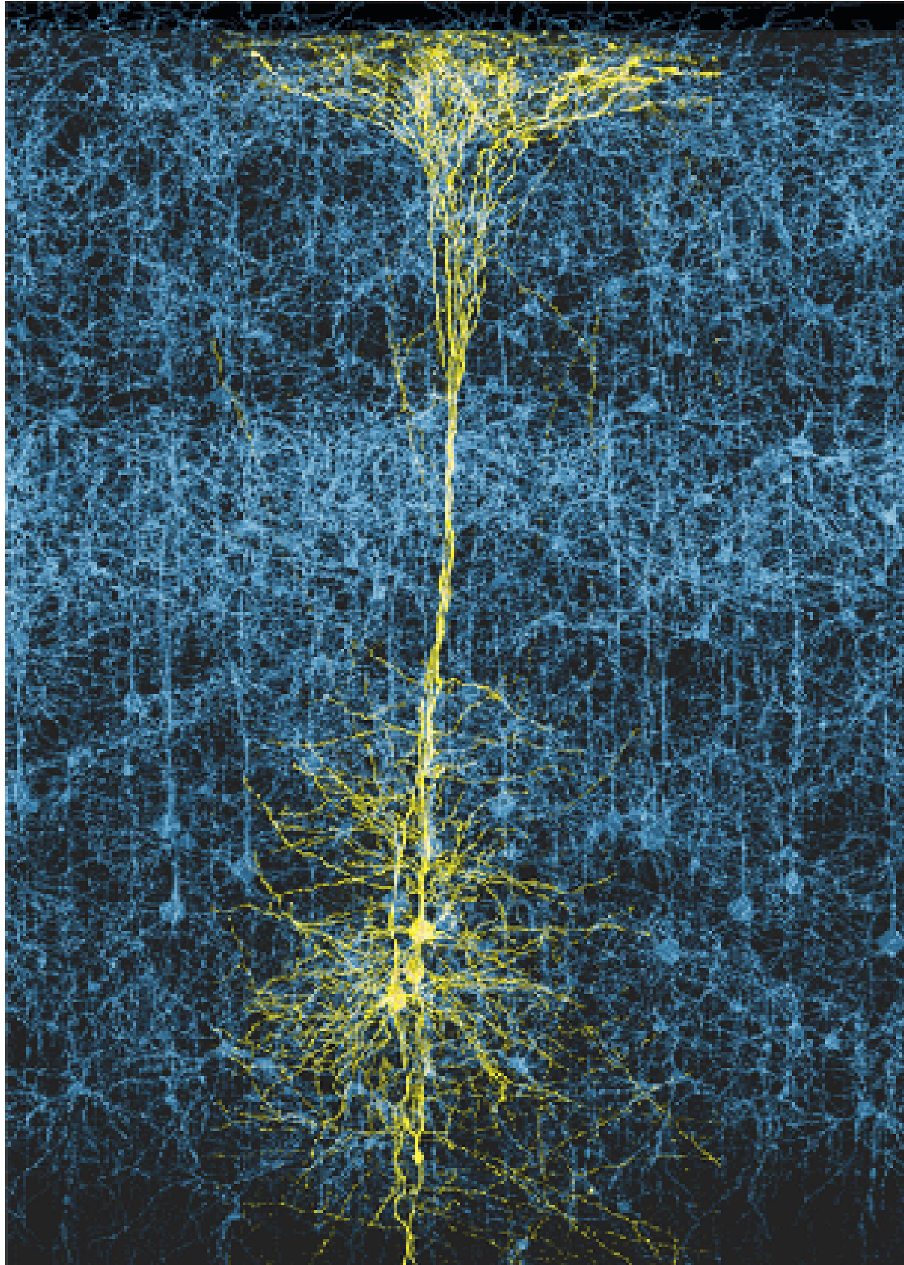


# Rat hippocampal neurons



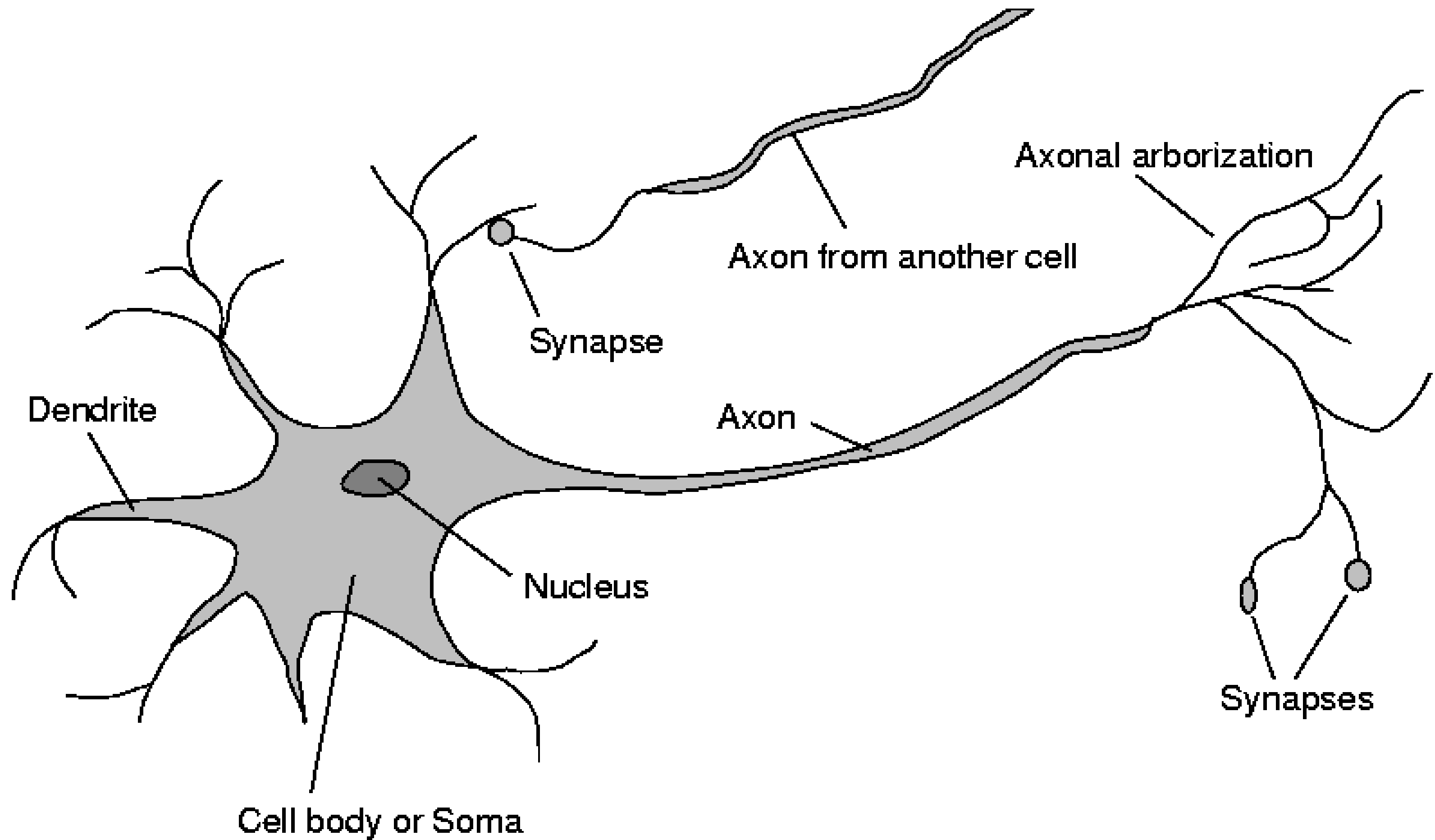

©QBM Cell Science.

# Pyramidal and other neurons



©HPC@EPFL

# The neuron



# Brain as a computational engine

| | **Personal Computer** | **Human Brain** |
|---|---|---|
| Computational units | 1 CPU, $3x10^8$ transistors (Intel Core 2 Duo) | $10^{11}$ neurons |
| Storage units | $10^{10}$ bits RAM, $10^{12}$ bits | $10^{11}$ neurons |
| Cycle time | $3x10^{-10}$ seconds | $10^{-3}$ seconds |
| Bandwidth | $10^{10}$ bits / second | $10^{14}$ bits / second |

Neurons and their signals are more complex than bits and logic gates, but how much more is arguable. However, it is clear that computers are closing the gap in performance.

# Spot of History

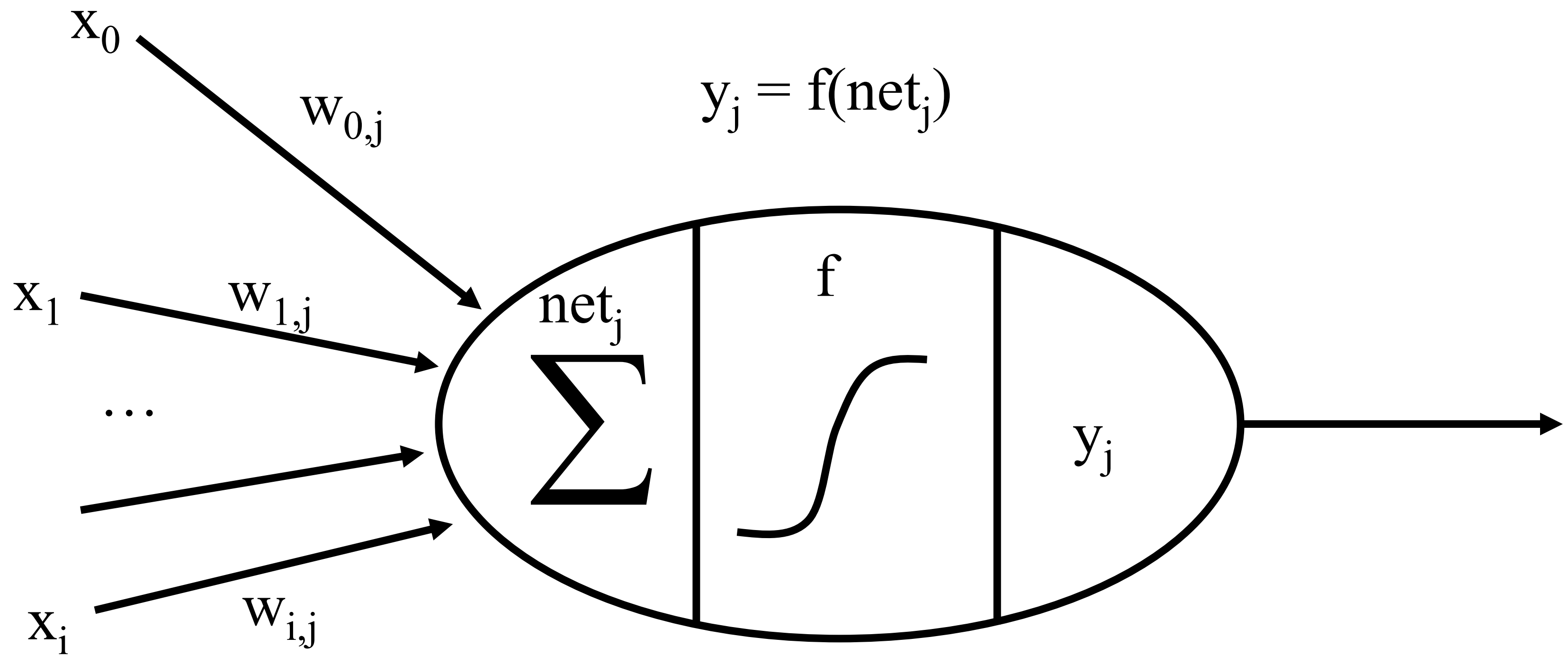
- This first artificial neuron was invented by Warren McCulloch and Walter Pitts in 1943
- They were colleagues of Norbert Wiener (invented “Cybernetics”), John von Neumann (helped invent modern computer architecture) and Claude Shannon (information theory)
- Their neuron takes input values (on artificial dendrites), multiplies by a weight (synaptic strength), processes these weighted inputs in a cell, and produces a 1 or 0 signal along the output (artificial axon)

# Task

- Classification
- Regression

- Both are possible for Neural Networks
- Classification is the focus of this lecture

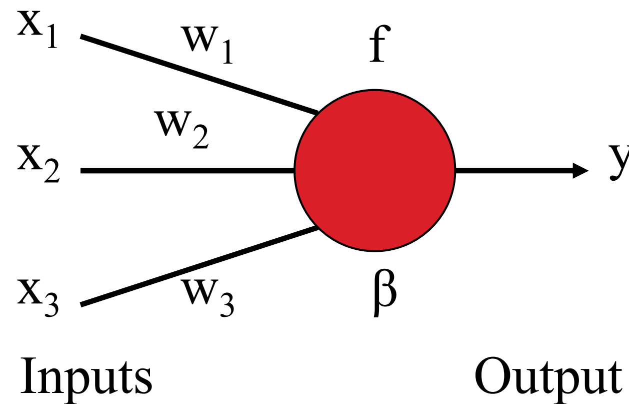
# The unit




Input Links | Input Function | Activation Function | Output | Output Links

# The unit: weighting input (1)

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$




*n* inputs $x_i$ , *n* weights $w_i$, bias $\beta$, output $y$.

```
public double sum(double[] x, double[] w,
double bias){
  double sum=bias;
  for (int i=0; i<x.length; i++)
    sum+=x[i]*w[i];
  return sum;
}
```

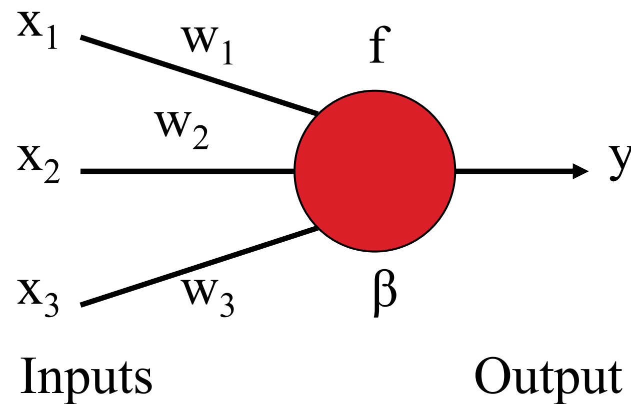
# The unit: weighting input (2)

$x_1 = 2.5$ $w_1 = 1.0$

$x_2 = 0.5$ $w_2 = 3.0$

$x_3 = 1.0$ $w_3 = -2.0$

$\beta = -1.5$




$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + x_3 w_3 + \beta$$

$$= 2.5 \times 1 + 0.5 \times 3 + 1.0 \times (-2) - 1.5$$

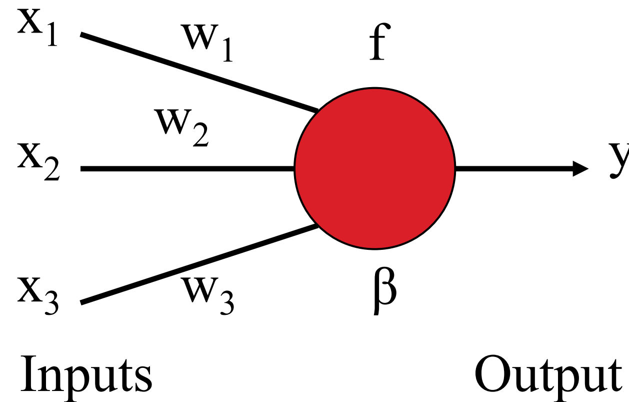
$$= 2.5 + 1.5 - 2.0 - 1.5$$

$$= 0.5$$

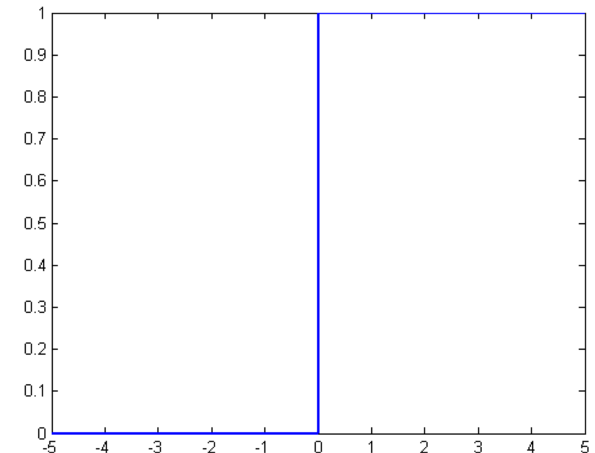
# The unit: forming output: threshold (1)

$$y = f(net)$$

$$f(net) = \begin{cases} 0 \ \textbf{if} \ net < 0 \\ 1 \ \textbf{if} \ net \geq 0 \end{cases}$$




```
public double outputFunction(double net) {
  // threshold function
  return (net>=0.0?1.0:0.0);
}
```



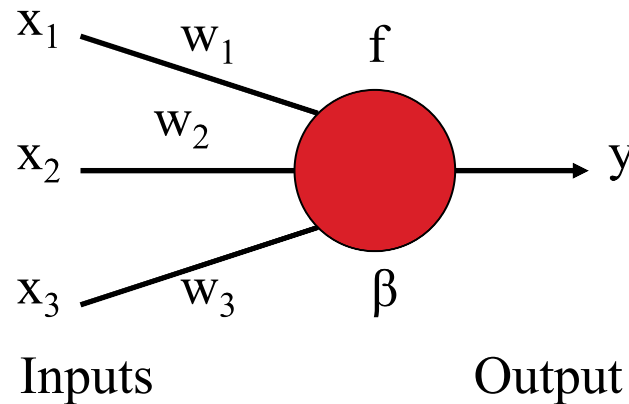
# The unit: forming output: threshold (2)

$x_1 = 2.5$ $w_1 = 1.0$

$x_2 = 0.5$ $w_2 = 3.0$

$x_3 = 1.0$ $w_3 = -2.0$

$\beta = -1.5$

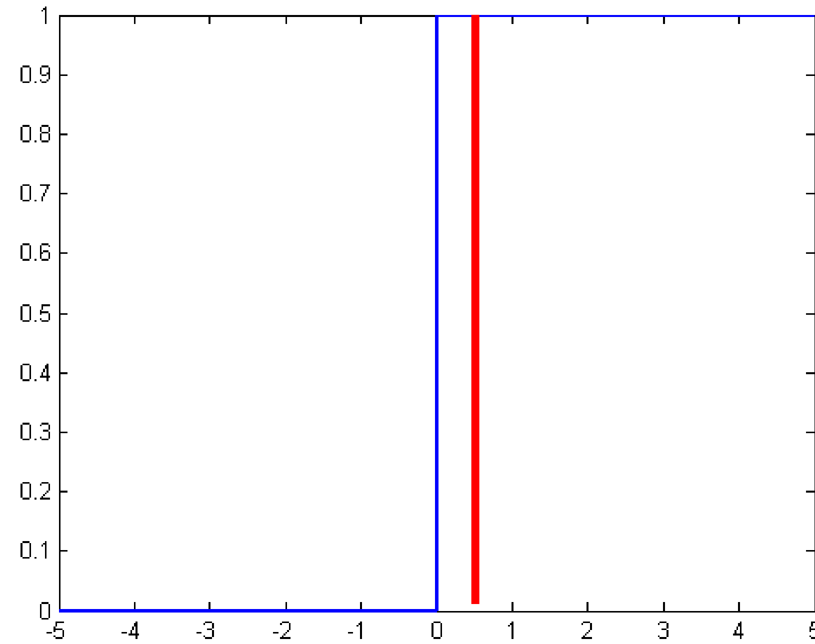



$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= 0.5$$

$$f(net) = \begin{cases} 0 \text{ if } net < 0 \\ 1 \text{ if } net \geq 0 \end{cases}$$

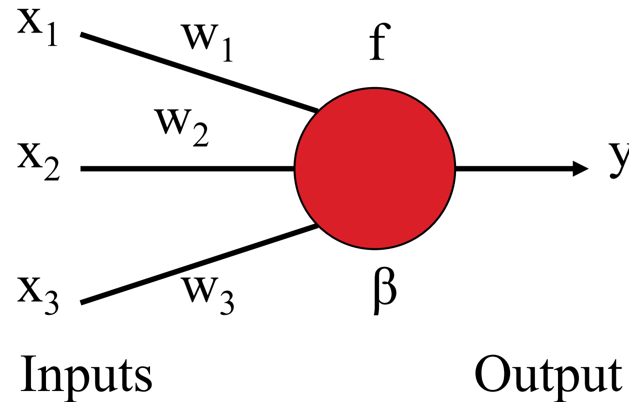
$$= 1$$



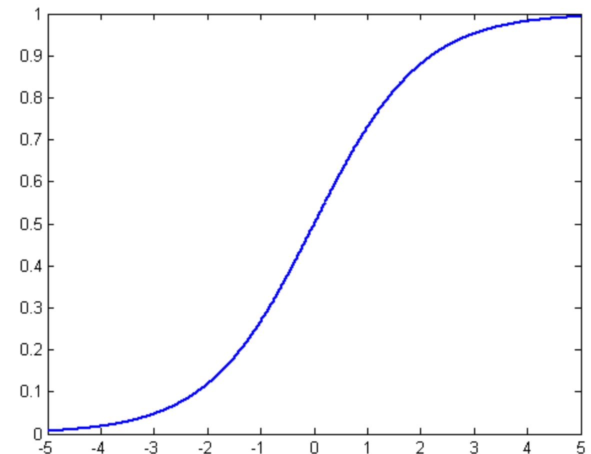
# The unit: forming output: sigmoid (1)

$$y = f(net)$$

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




```
public double outputFunction(double net) {
  // a sigmoid function
  return 1.0/(1.0+Math.exp(-net));
}
```



# The unit: forming output: sigmoid (2)

$x_1 = 2.5$ $w_1 = 1.0$

$x_2 = 0.5$ $w_2 = 3.0$

$x_3 = 1.0$ $w_3 = -2.0$

$\beta = -1.5$

x1, x2, x3 (Inputs) → w1, w2, w3 → f, β → y (Output)

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= 0.5$$

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

$$= \frac{1}{1+e^{-0.5}}$$

$$= 0.6225$$

# What can a unit compute?

- Regression
  - E.g. continuous output
- Classification
  - E.g. logic functions (AND, OR, NOT etc.)

# Logic functions

Y = X1 OR X2

| X1 | X2 | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Y = X1 AND X2

| X1 | X2 | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Y = X1 NOR X2

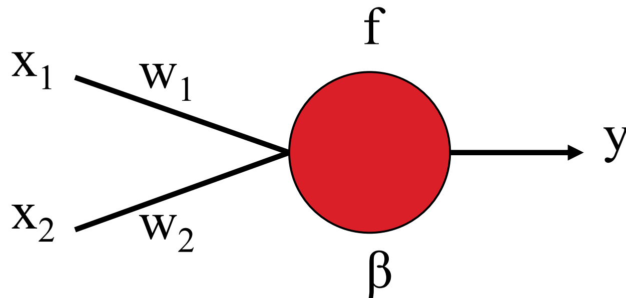
| X1 | X2 | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Y = X1 NAND X2

| X1 | X2 | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Logic functions: OR (1)

| | X1 | X2 | Y |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 1 |
| 3 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 |




$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 0 \textbf{ if } net < 0 \\ 1 \textbf{ if } net \geq 0 \end{cases}$$

$x_1 w_1 + x_2 w_2 + \beta = net$

1. $w_1 * 0 + w_2 * 0 + \beta < 0$

   $\beta < 0$

2. $w_1 * 0 + w_2 * 1 + \beta \geq 0$

   $w_2 + \beta \geq 0$

   $w_2 \geq -\beta$

3. $w_1 * 1 + w_2 * 0 + \beta \geq 0$

   $w_1 + \beta \geq 0$

   $w_1 \geq -\beta$

4. $w_1 * 1 + w_2 * 1 + \beta \geq 0$

   $w_1 + w_2 + \beta \geq 0$

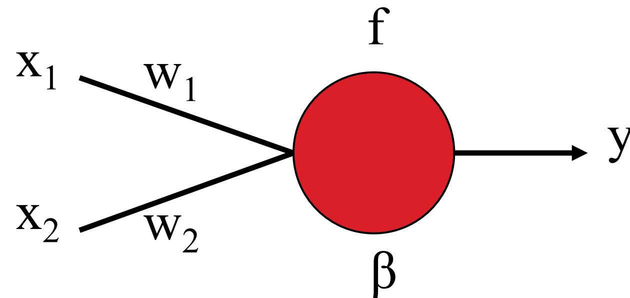
   $w_1 + w_2 \geq -\beta$

# Logic functions: OR (2)

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 0 \text{ if } net < 0 \\ 1 \text{ if } net \geq 0 \end{cases}$$




$w_1 = 1.0$

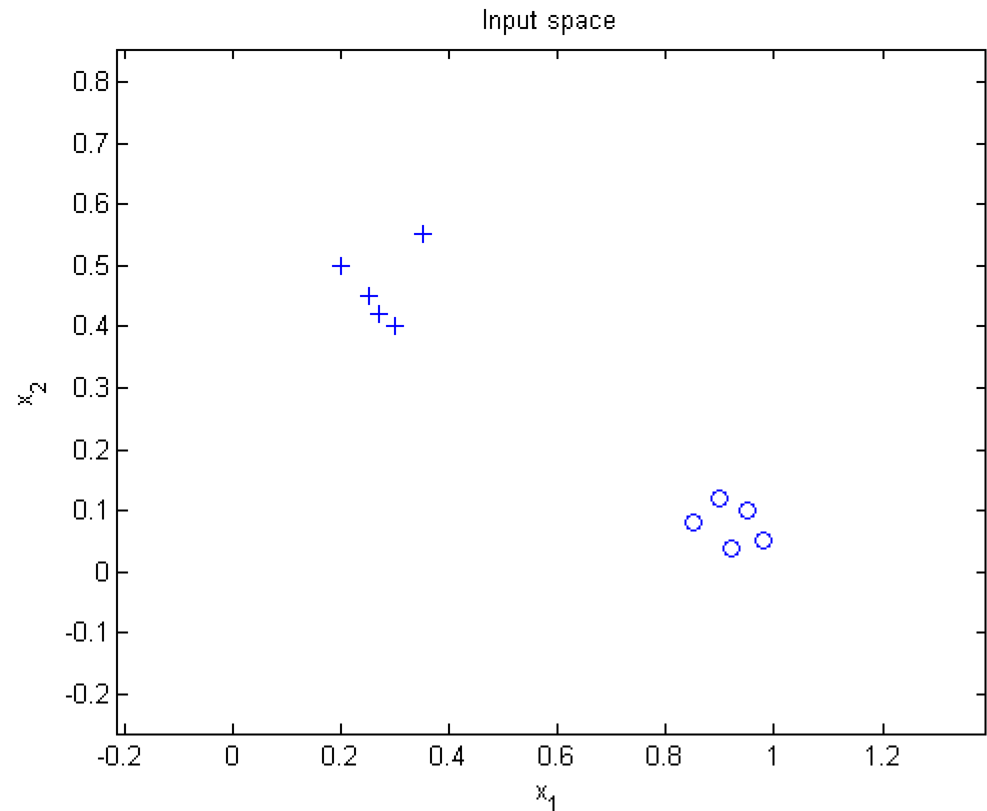
$w_2 = 1.0$

$\beta = -0.5$

| x1 | x2 | net | f(net) |
|---|---|---|---|
| 0 | 0 | -0.5 | 0 |
| 0 | 1 | 0.5 | 1 |
| 1 | 0 | 0.5 | 1 |
| 1 | 1 | 1.5 | 1 |

# What a network "computes"?

- Weight values are coefficients in a function
- Consider a classification problem: which group does a new point belong to ?




A network with two inputs:
$x_1$ and $x_2$
and one output:
0 or 1

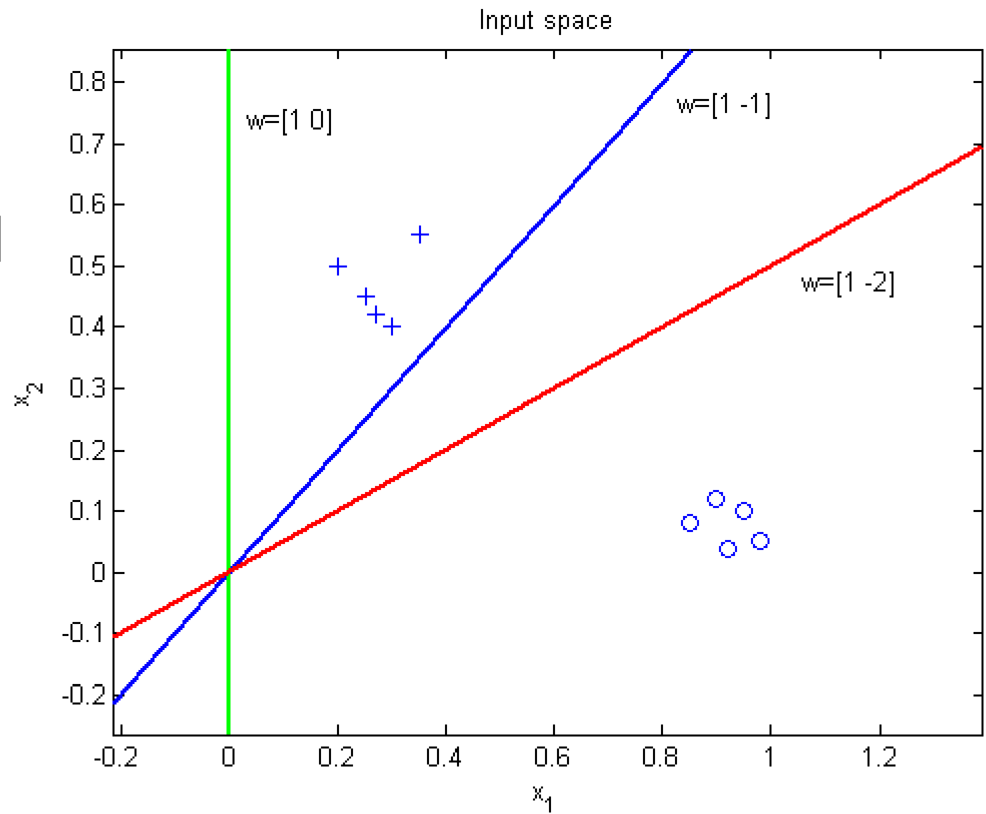
Assume bias $\beta$=0 for now.

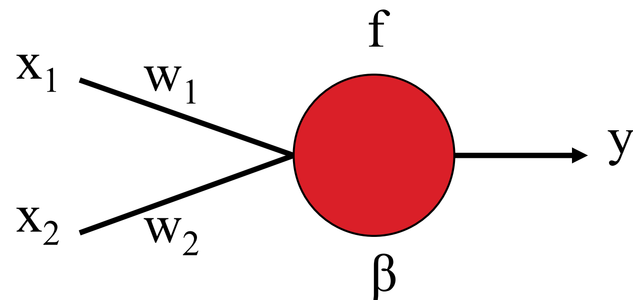
$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$f(net) = \begin{cases} 0 \textbf{ if } net < 0 \\ 1 \textbf{ if } net \geq 0 \end{cases}$$

# Decision boundaries (1)

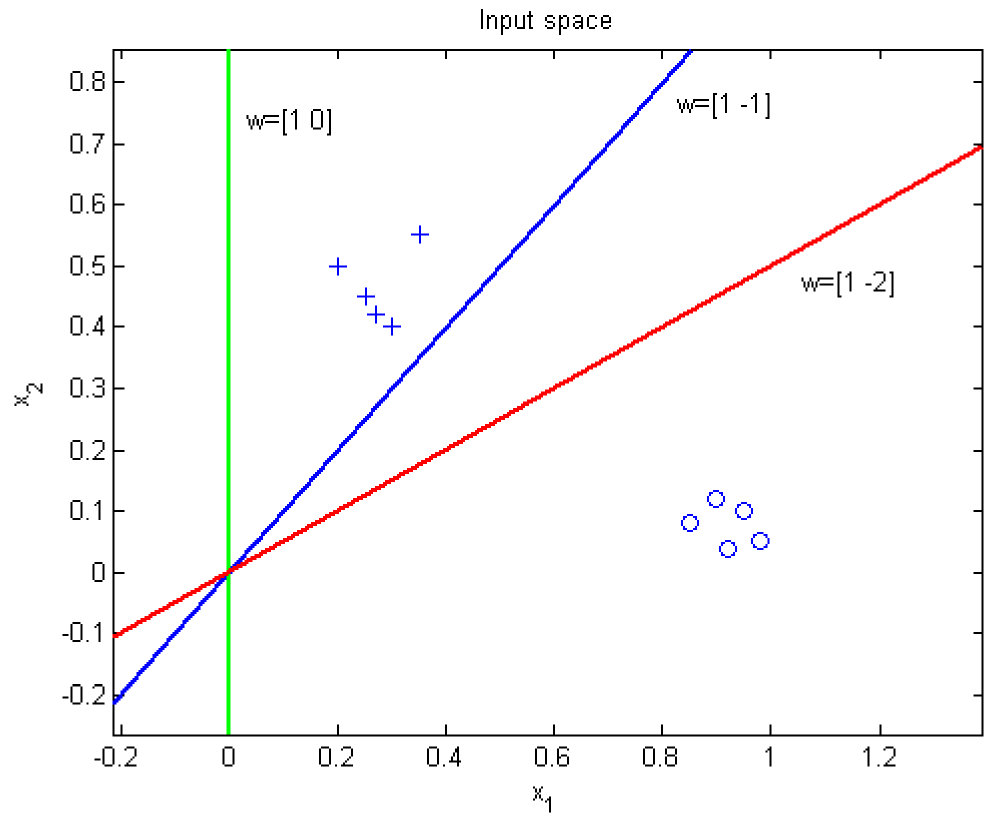
- Decision boundary separates those input values that produce a 1 from those that produce 0
- It is the task of any classifier to partition the input space into classes via decision boundaries.

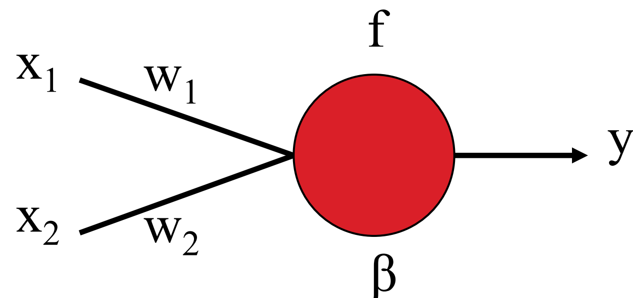


# Decision boundaries (2)

- The decision boundary contains all the values of the vector *x* for which this sum = 0
- For these weights, it is zero when $x_1 w_1 + x_2 w_2 = 0$
- Here, that's $x_1 - 2x_2 = 0$, ie: the line $x_2 = 0.5x_1$ (red line)



# Decision boundaries (3)

- For a threshold unit, the decision boundary contains all the values for which net = 0
- If bias = 0, the decision boundary must pass through the origin
- The bias value frees the decision boundary from crossing the origin

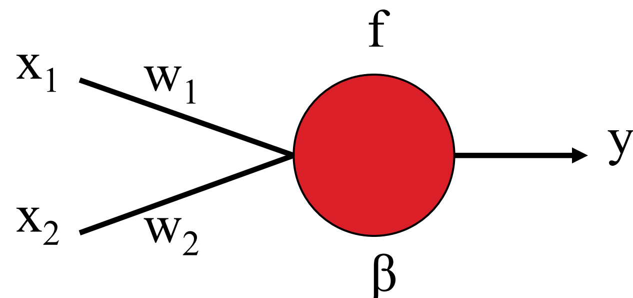
$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$net = x_1 w_1 + x_2 w_2 + \beta$$

$$0 = x_1 w_1 + x_2 w_2 + \beta$$

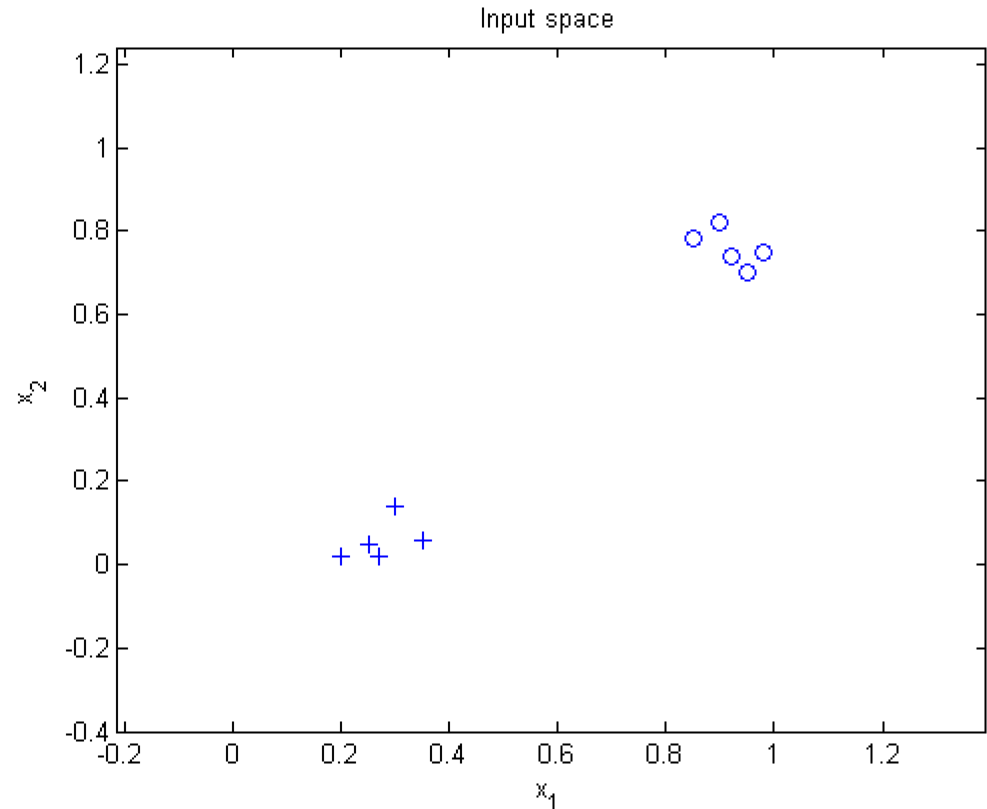
$$x_1 w_1 = -x_2 w_2 - \beta$$

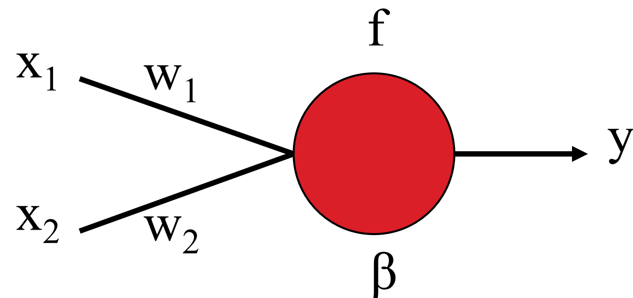
$$x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1}$$



# Decision boundaries (4)

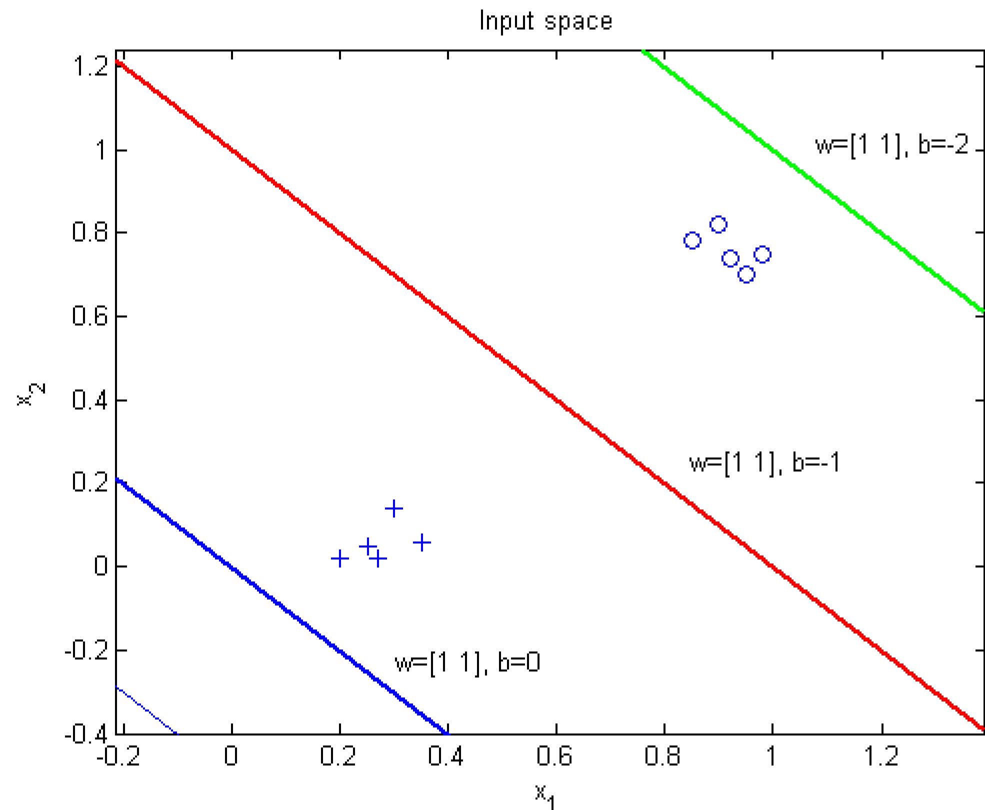
- What if the samples are distributed differently…?

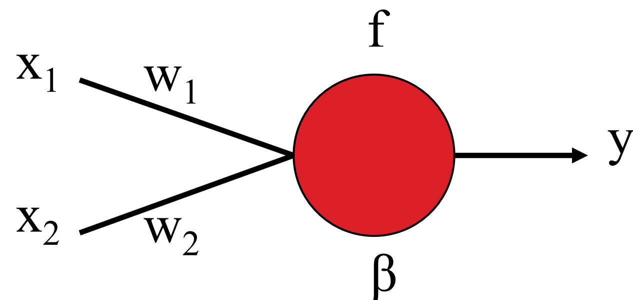


# Decision boundaries (5)

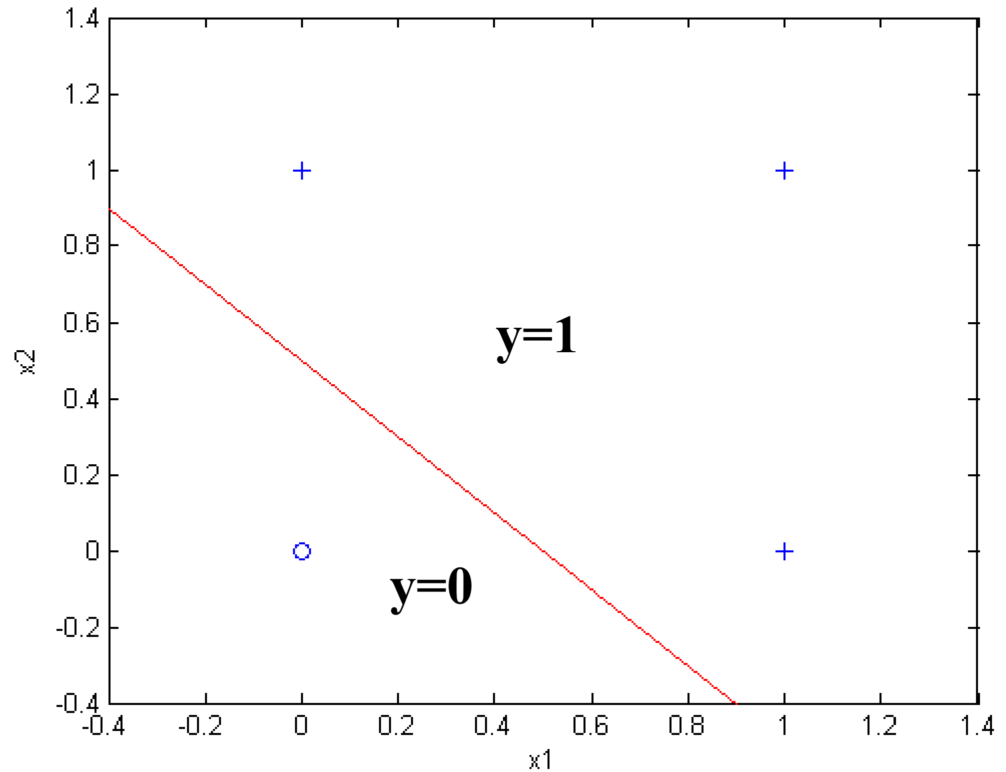
- The bias value frees the decision boundary from crossing the origin
- Allows the threshold unit to realize any linear separator



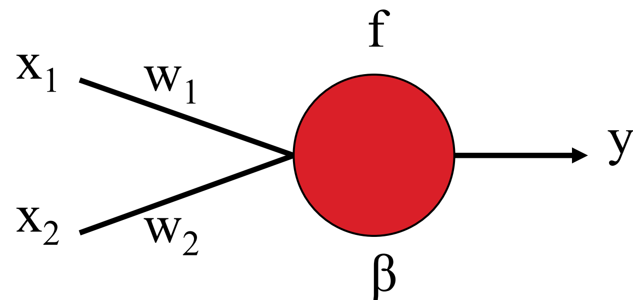
# Decision boundary: OR

$w_1 = 1.0$
$w_2 = 1.0$
$\beta = -0.5$



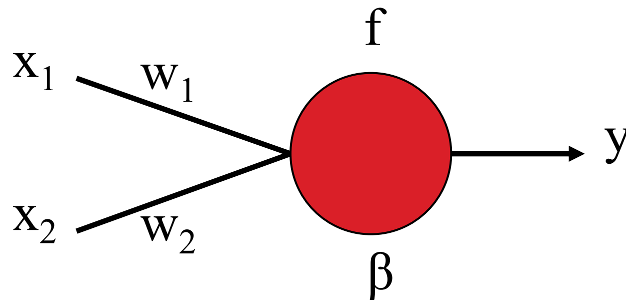

$$x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1}$$

$$x_1 = -x_2 + 0.5$$



# Logic functions: AND (1)

| | X1 | X2 | Y |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 |
| 3 | 1 | 0 | 0 |
| 4 | 1 | 1 | 1 |




$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 0 \textbf{ if } net < 0 \\ 1 \textbf{ if } net \geq 0 \end{cases}$$

$x_1 w_1 + x_2 w_2 + \beta = net$

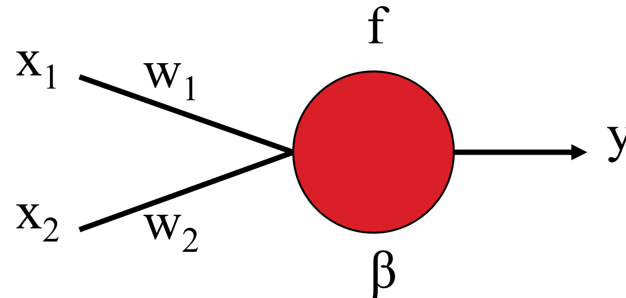
1. $w_1 * 0 + w_2 * 0 + \beta < 0$
   $\beta < 0$
2. $w_1 * 0 + w_2 * 1 + \beta < 0$
   $w_2 + \beta < 0$
   $w_2 < -\beta$
3. $w_1 * 1 + w_2 * 0 + \beta < 0$
   $w_1 + \beta < 0$
   $w_1 < -\beta$
4. $w_1 * 1 + w_2 * 1 + \beta \geq 0$
   $w_1 + w_2 + \beta \geq 0$
   $w_1 + w_2 \geq -\beta$

# Logic functions: AND (2)

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 0 \textbf{ if } net < 0 \\ 1 \textbf{ if } net \geq 0 \end{cases}$$




$w_1 = 1.0$

$w_2 = 1.0$

$\beta = -1.5$

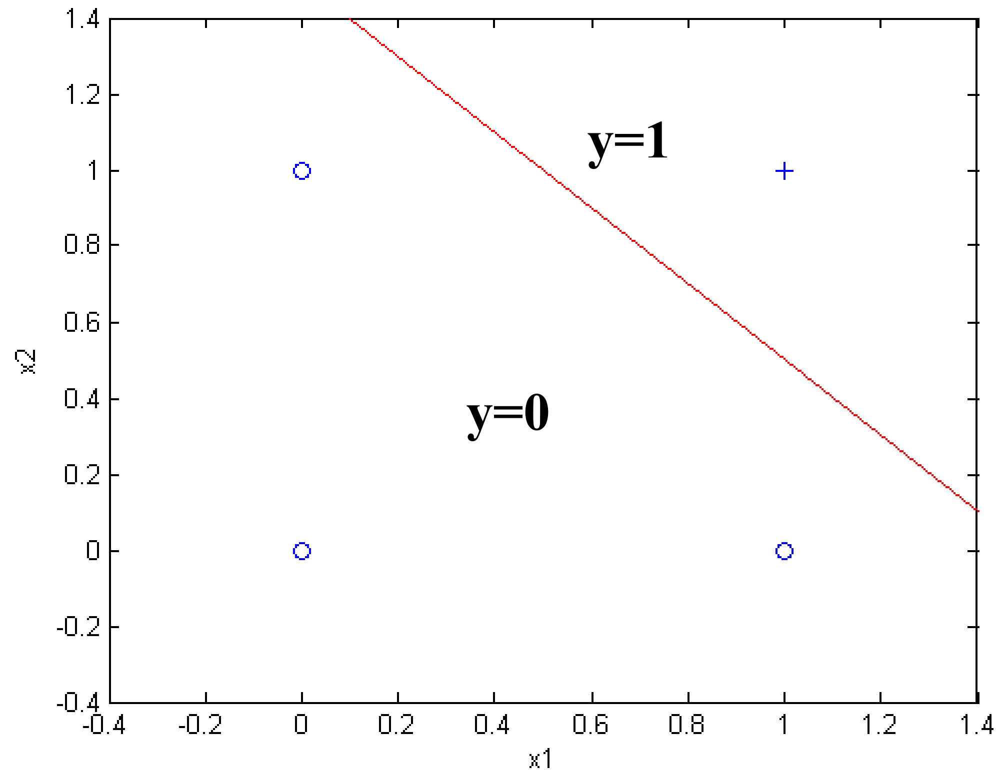
| x1 | x2 | net | f(net) |
|---|---|---|---|
| 0 | 0 | -1.5 | 0 |
| 0 | 1 | -0.5 | 0 |
| 1 | 0 | -0.5 | 0 |
| 1 | 1 | 0.5 | 1 |

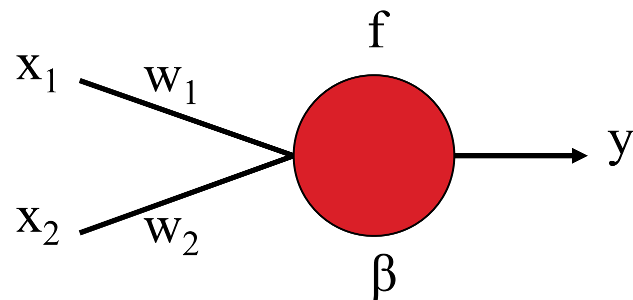
# Decision boundary: AND

$w_1 = 1.0$
$w_2 = 1.0$
$\beta = -1.5$

$$x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1}$$

$$x_1 = -x_2 + 1.5$$



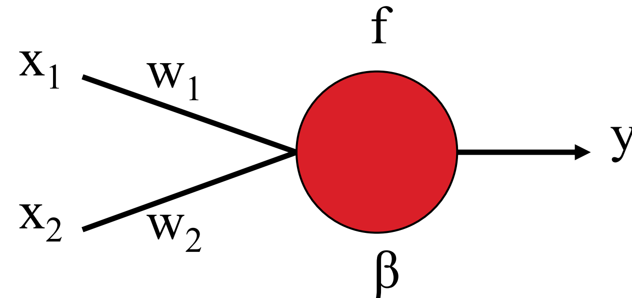



# Logic functions: NOR

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 0 \textbf{ if } net < 0 \\ 1 \textbf{ if } net \geq 0 \end{cases}$$




$w_1 = -1.0$
$w_2 = -1.0$
$\beta = 0.5$

| x1 | x2 | net | f(net) |
|---|---|---|---|
| 0 | 0 | 0.5 | 1 |
| 0 | 1 | -0.5 | 0 |
| 1 | 0 | -0.5 | 0 |
| 1 | 1 | -1.5 | 0 |

# Decision boundary: NOR

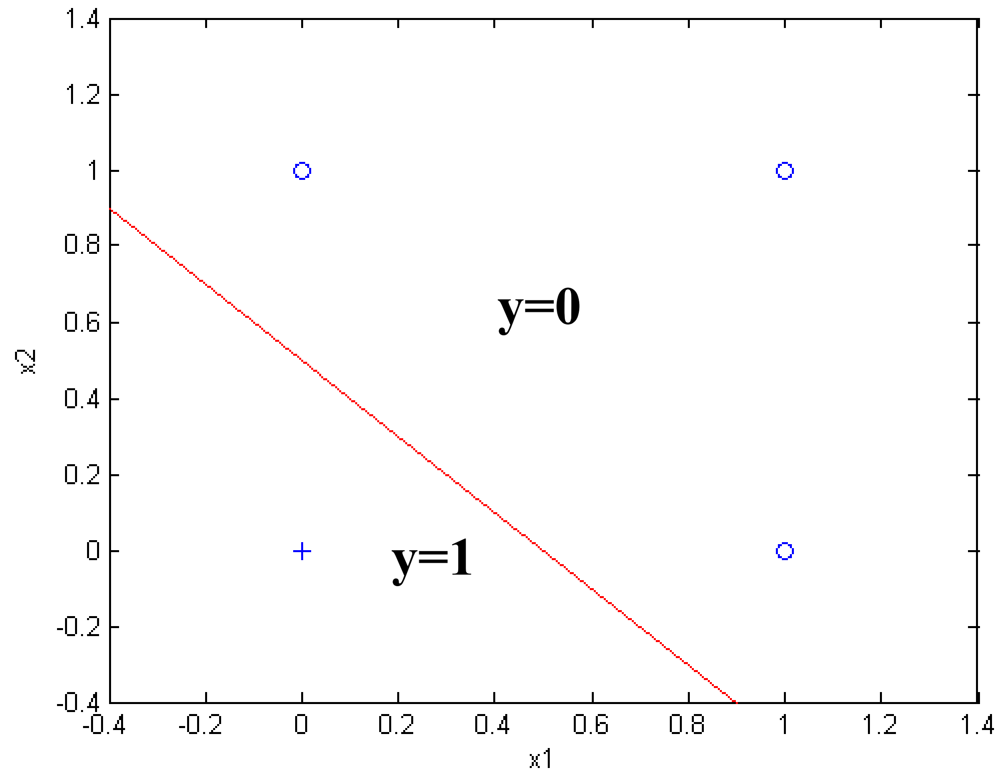
$w_1 = -1.0$

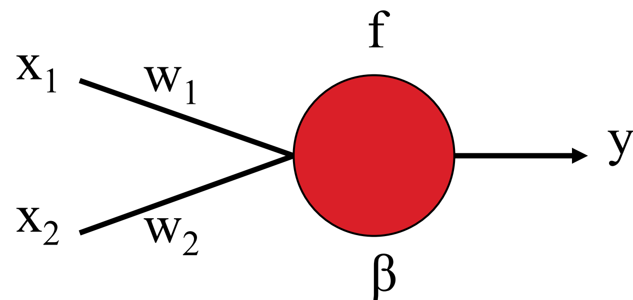
$w_2 = -1.0$

$\beta = 0.5$

$$x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1}$$

$$x_1 = -x_2 + 0.5$$



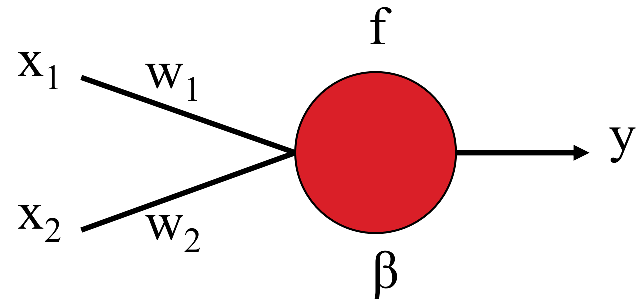



# Logic functions: NAND

$$net = \sum_{i=1}^{n} x_i w_i + \beta$$

$$= x_1 w_1 + x_2 w_2 + \beta$$

$$f(net) = \begin{cases} 0 \textbf{ if } net < 0 \\ 1 \textbf{ if } net \geq 0 \end{cases}$$




$w_1 = -1.0$

$w_2 = -1.0$

$\beta = 1.5$

| x1 | x2 | net | f(net) |
|---|---|---|---|
| 0 | 0 | 1.5 | 1 |
| 0 | 1 | 0.5 | 1 |
| 1 | 0 | 0.5 | 1 |
| 1 | 1 | -0.5 | 0 |

# Decision boundary: NAND

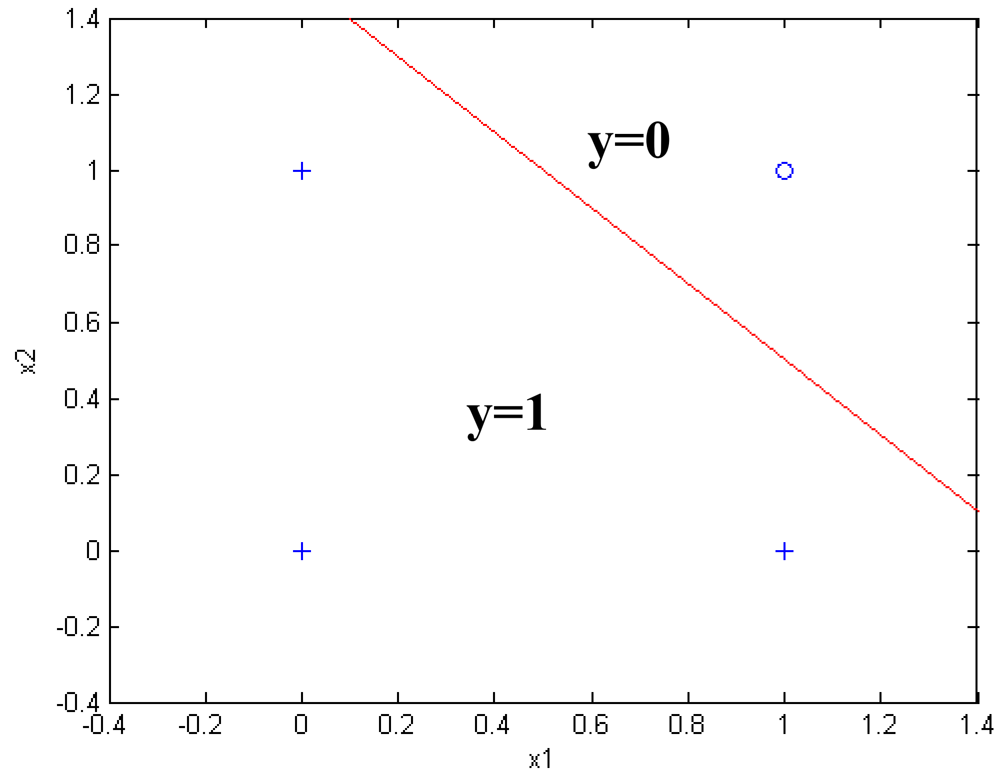
$w_1$ = -1.0

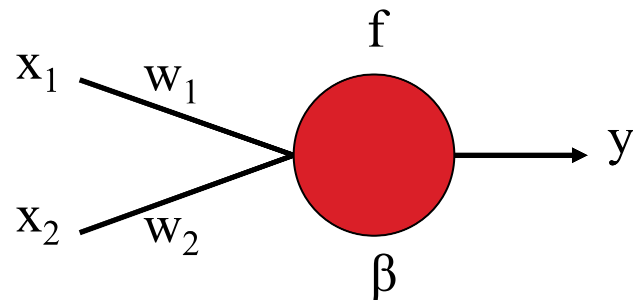
$w_2$ = -1.0

$\beta$ = 1.5

$$x_1 = -x_2 \frac{w_2}{w_1} - \frac{\beta}{w_1}$$

$$x_1 = -x_2 + 1.5$$

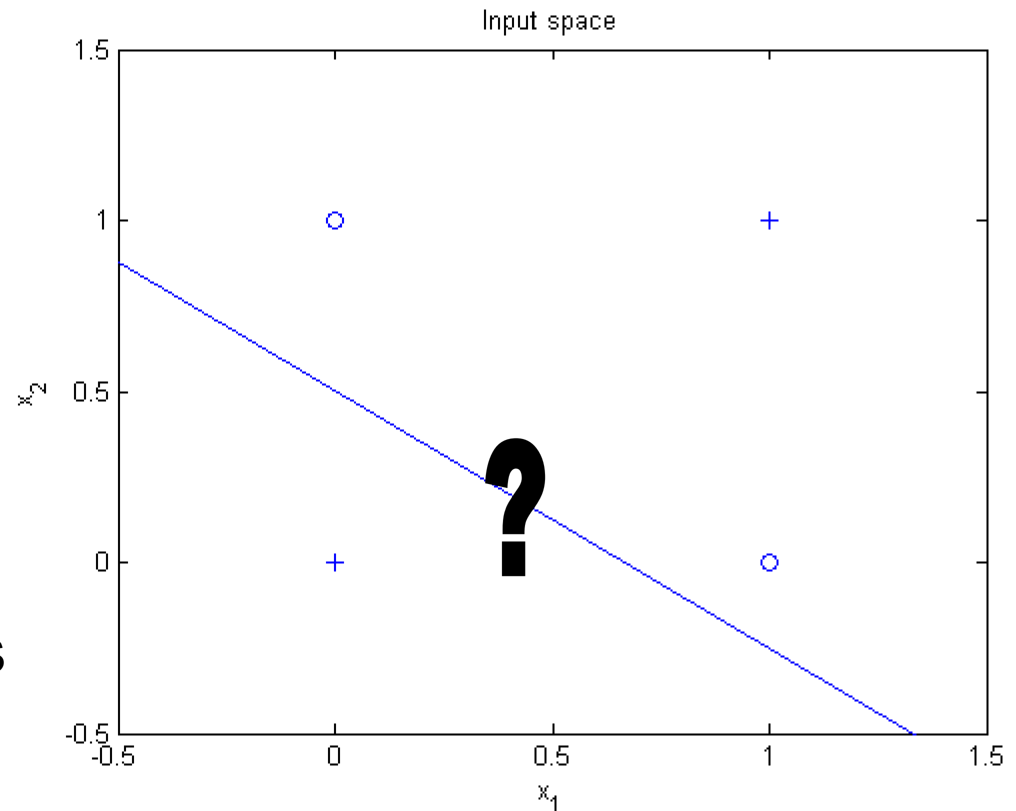


# Decision boundary: XOR

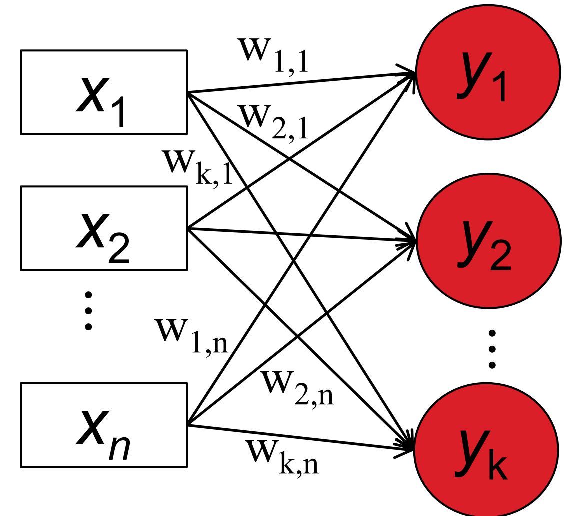
- When inputs are not linearly separable, what can be done?
- Example: XOR problem:
  $y$ = XOR($x_1$, $x_2$).

- Single layer networks can only represent linearly separable functions



# Making a network

Here, really just $k$ separate neurons, each with one output, which each use the same set of inputs.




```
double[][] weights; // all weights
double[] biases;    // all biases

double[] network(double[] x) {
  double[] y=new double[weights.length];
  for (int k=0; k<weights.length; k++) {
    y[k]=outputFunction(sum(x, weights[k],
      biases[k]));
  return y;
}
```

# Learning in neural networks

- Single-layer networks
- Outputs:
  - Binary output: function = threshold
  - Continuous output: threshold = sigmoid

# Notation: counts and indices

- Textbook is not consistent: uses *n* for both number of inputs and number of training patterns.
- Earlier slides not all consistent either.
- Hereon in slides, will use:
  - Observations/patterns: $i=1,..,n$.
  - Training patterns: $i=1,..,n_{tr}$
  - Test patterns: $i=1,..,n_{te}$
    - If using all observations for training, $n_{tr} = n$. If test set is a subset of the observations, $n_{tr} + n_{te} = n$.
  - Inputs/attributes: $j=1,..,p$.
  - Outputs/responses: $k=1,..,m$

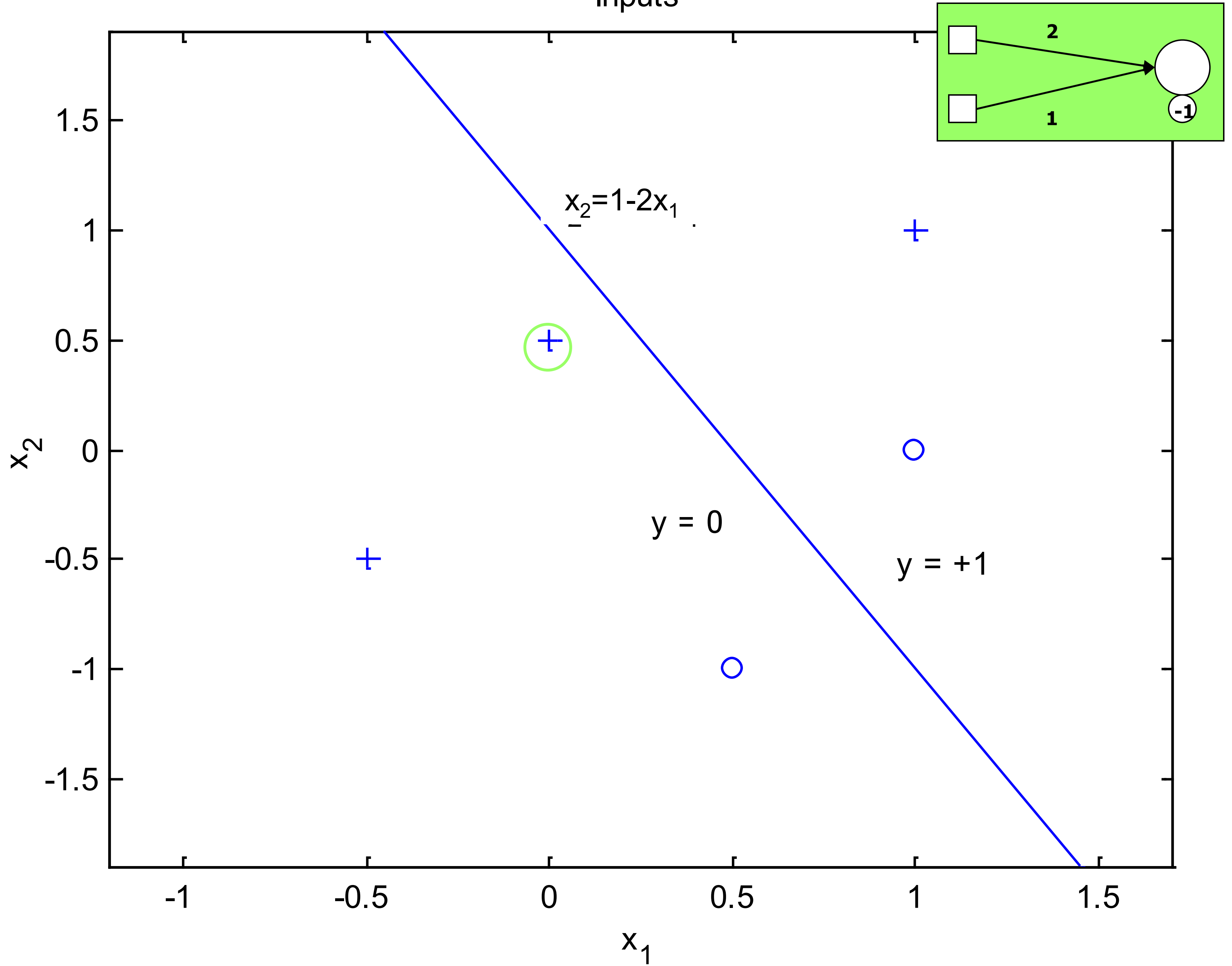
# Notation: data and network output (1)

- Observations/patterns:
  - input: $x_i = \{x_{ij}\}$, $i$ = pattern #, $j$ = input #
  - correct output: $y_i = \{y_{ik}\}$, $i$ = pattern #, $k$ = output #

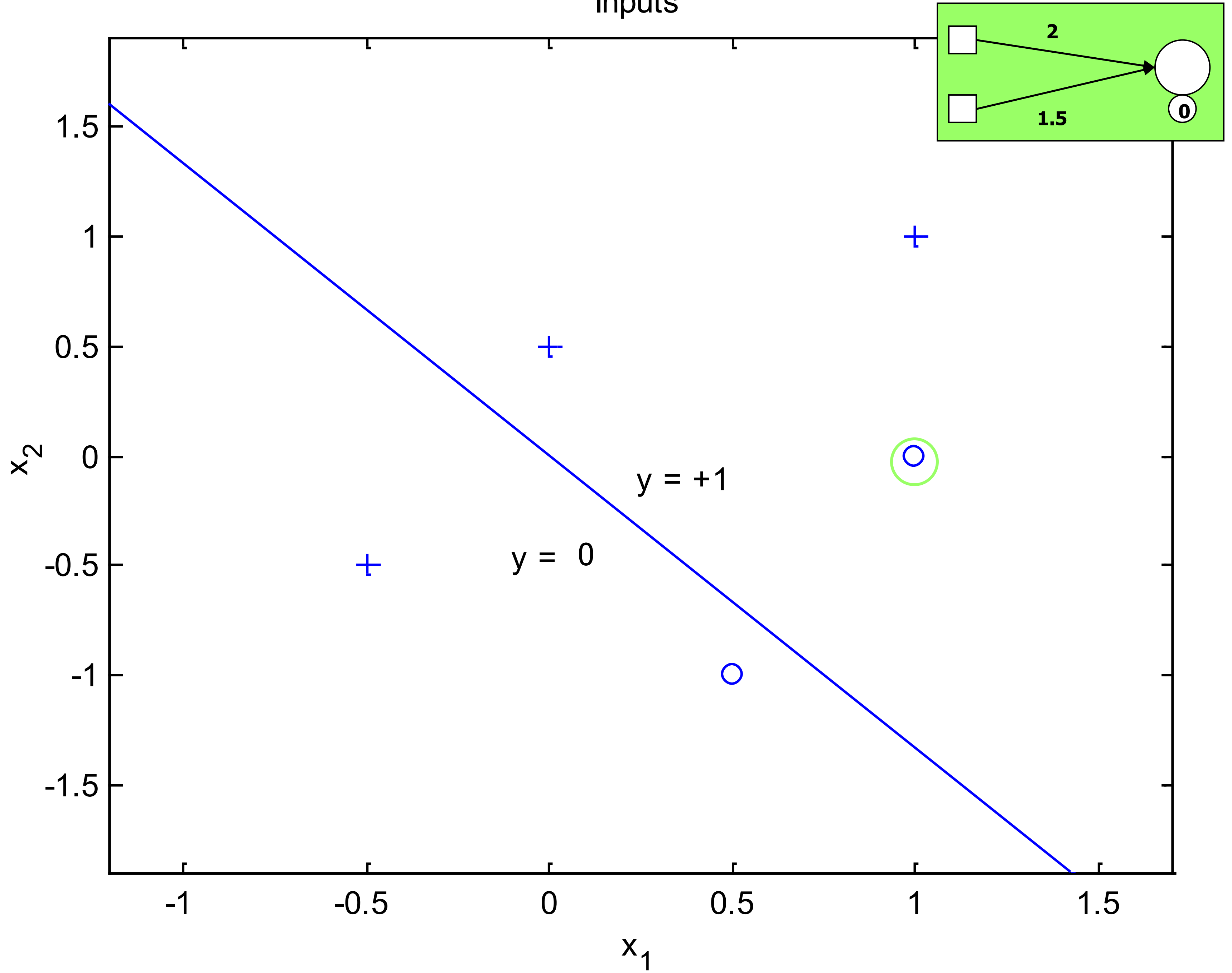
- Neural network:
  - output unit value: $o_i = \{o_{ik}\}$, $i$ = pattern #, $k$ = output #
  - Also sometimes write $o_{ik}$ as $f_{network,k}(x_i)$ to indicate that the neural network output is a function of the input.

# Learning in neural networks

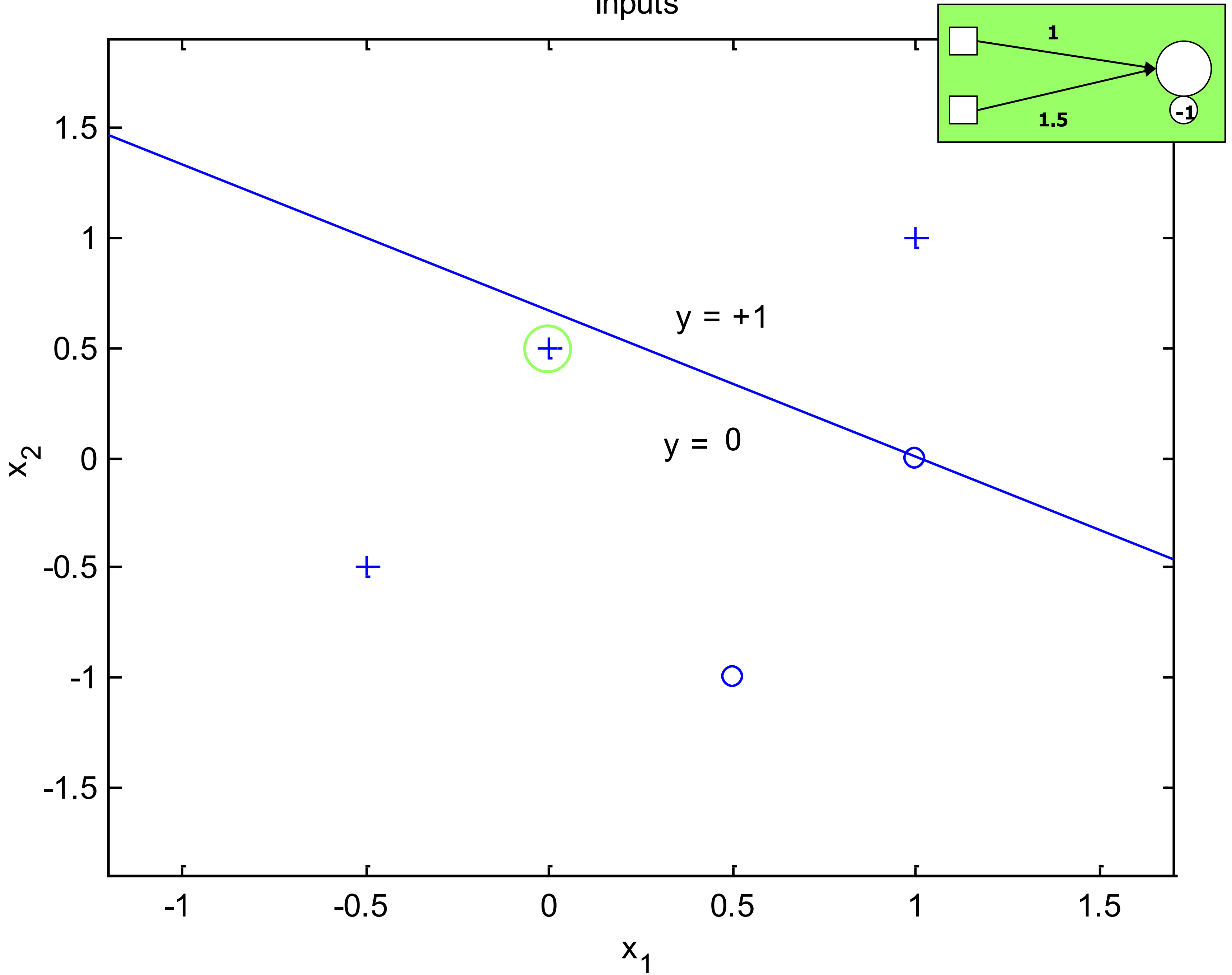
- Following:
  a demonstration of finding suitable weights for a single threshold neuron, from the perspective of the decision boundary in the input space.
- We want to automate this process.

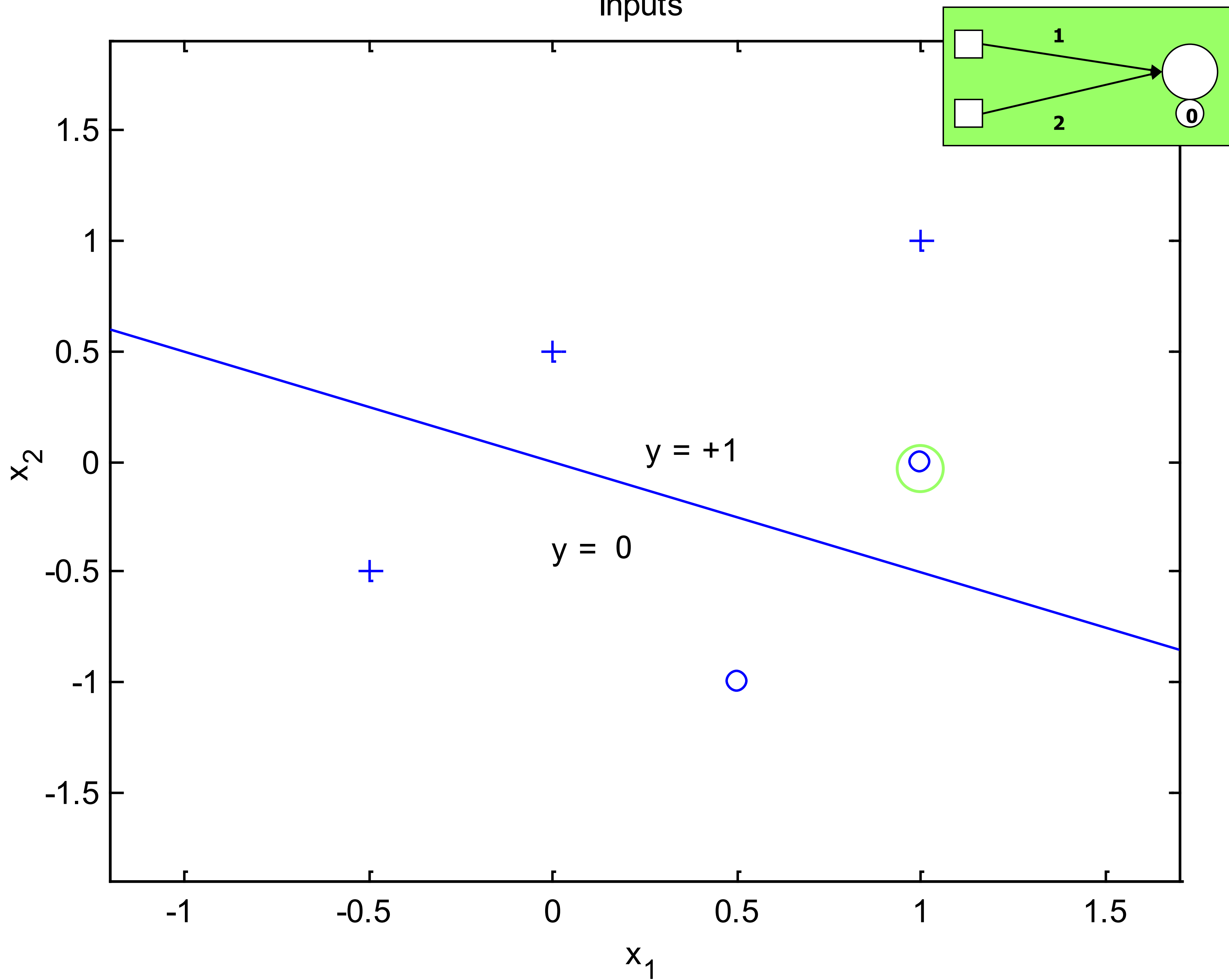


Inputs
2
1
-1
$x_2$=1-2$x_1$
y = 0
y = +1
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.5
0
0.5
1
1.5
$x_1$
$x_2$

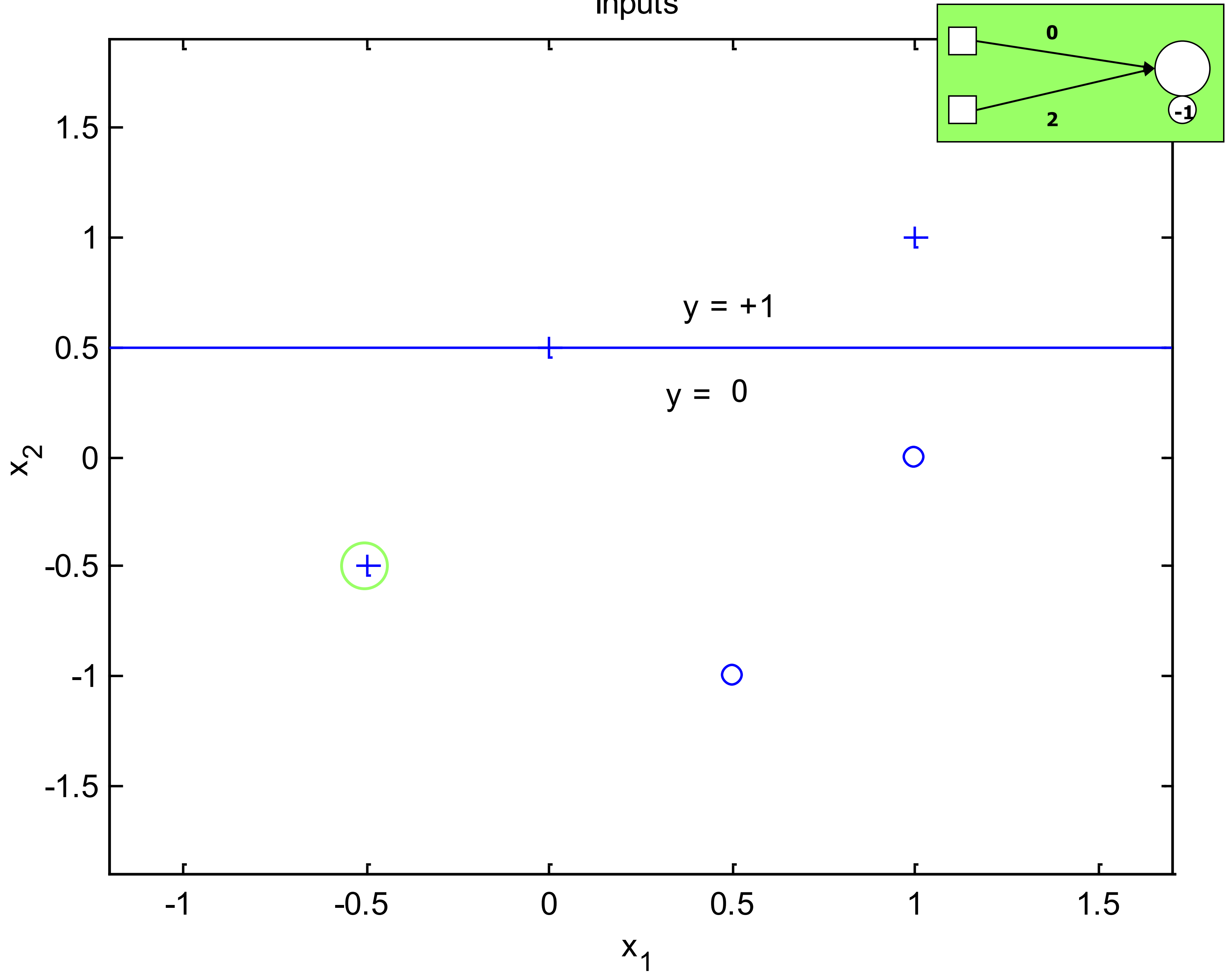


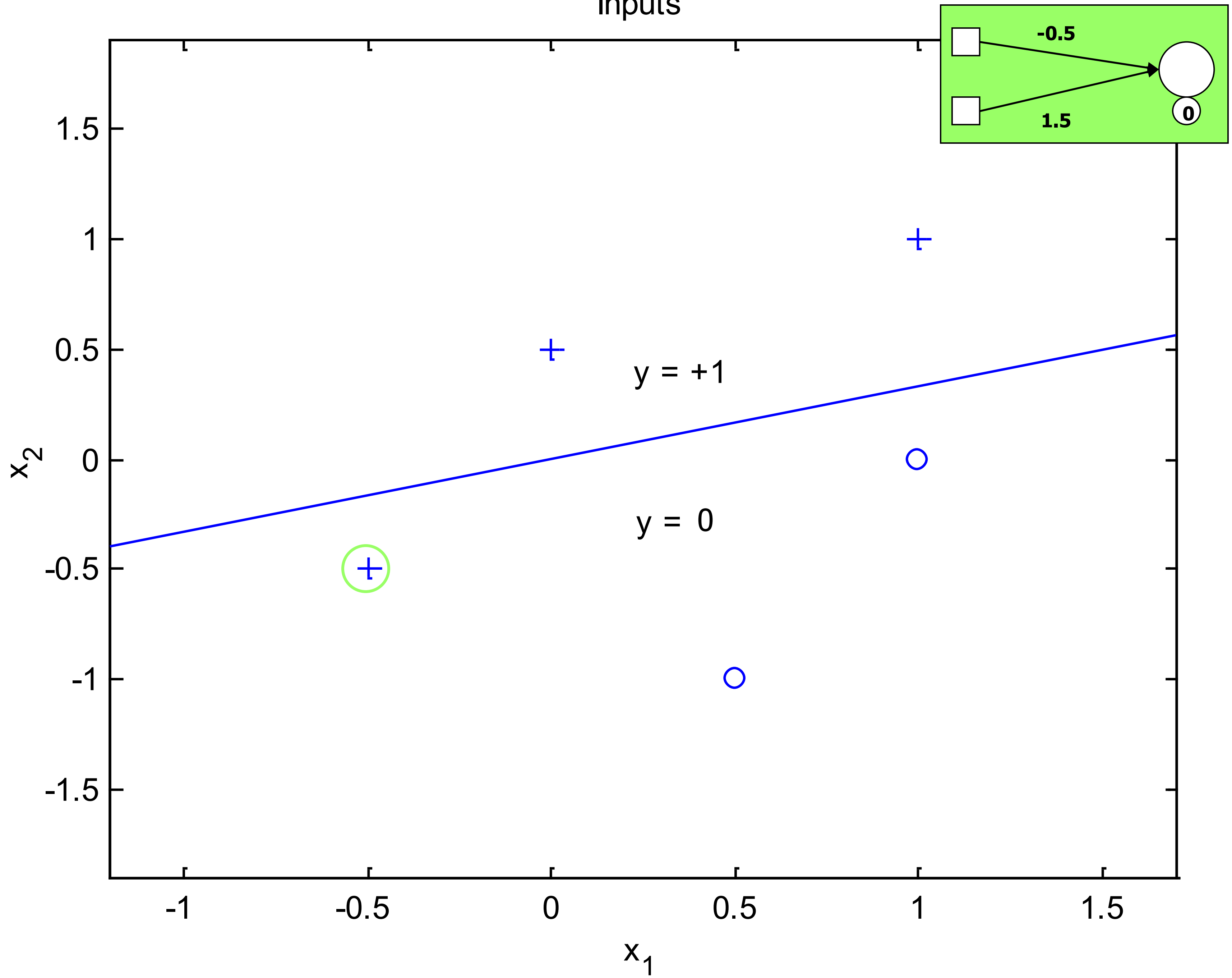
Inputs
2
1.5
0
y = +1
y = 0
x1
x2

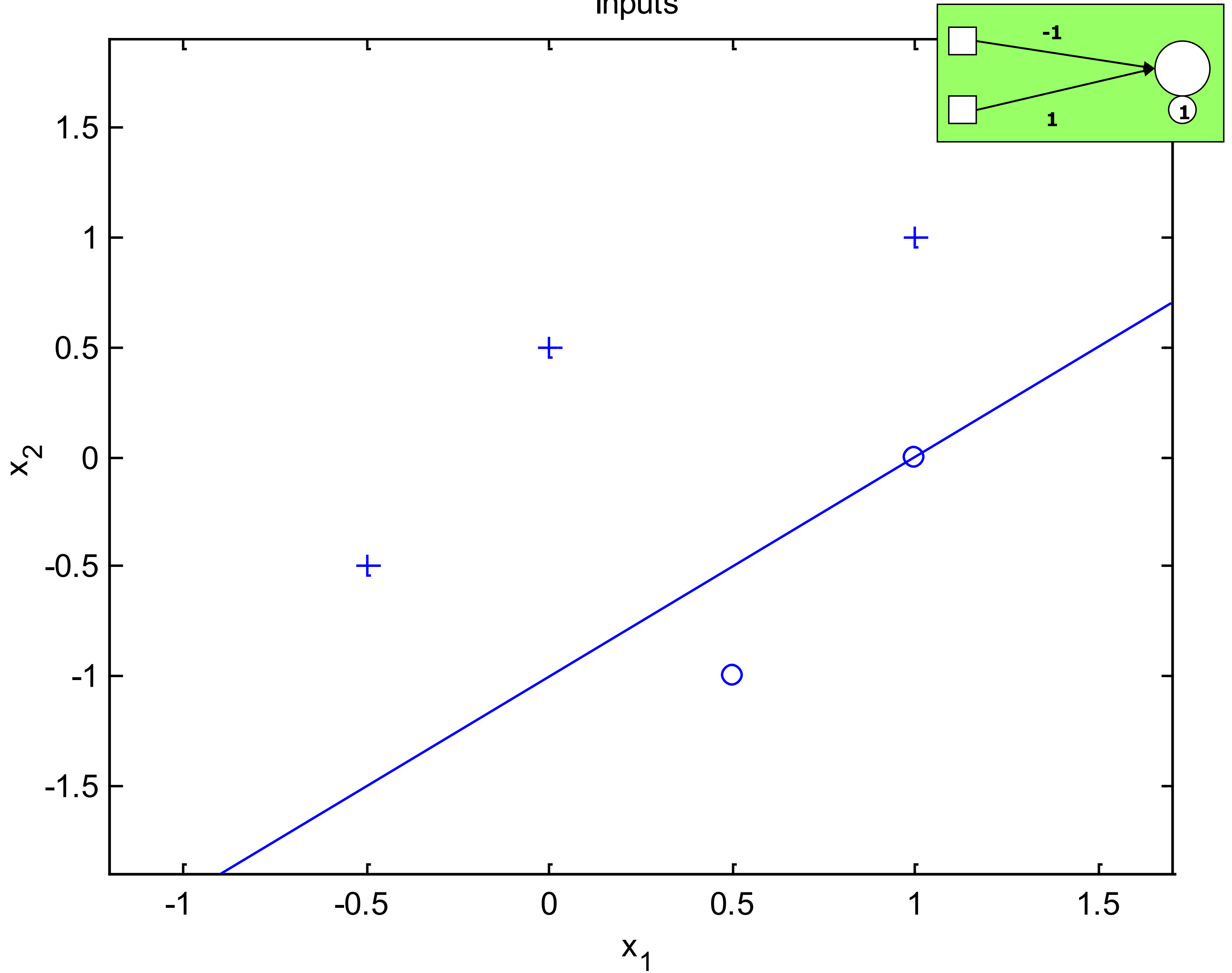

Inputs
1
1.5
-1
y = +1
y = 0
1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.5
0
0.5
1
1.5
$x_1$
$x_2$



Inputs
1
2
0
y = +1
y = 0
x_1
x_2
1.5
1
0.5
0
-0.5
-1
-1.5


Inputs
0
2
-1
y = +1
y = 0
1.5
1
0.5
0
-0.5
-1
-1.5
x2
-1
-0.5
0
0.5
1
1.5
x1


Inputs
-0.5
1.5
0
y = +1
y = 0
x1
x2
-1
-0.5
0
0.5
1
1.5
-1.5


Inputs
-1
1
1
1.5
1
0.5
0
-0.5
-1
-1.5
x2
-1
-0.5
0
0.5
1
1.5
x1

# Learning in Neural Networks

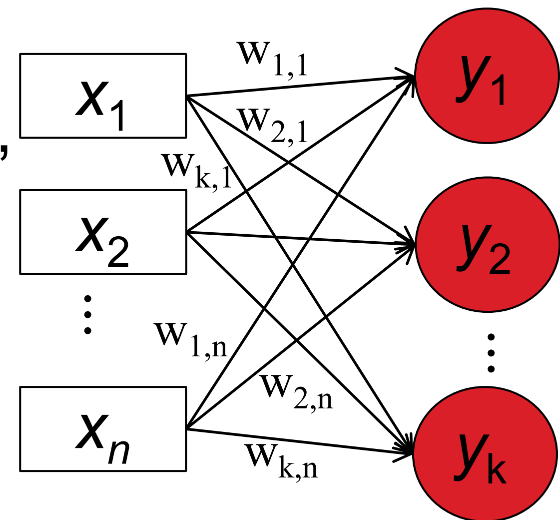
- For a single layer feedforward neural network, learning can be seen as shifting the decision boundary until the training set examples are classified correctly
- Decision boundary is shifted by changing the weights (including the bias)

$$w_{jk,new} = w_{jk} + \Delta w_{jk}$$

# How to find useful weights

- Labelled training data. For a given set of weights, the network will make predictions in response to inputs.
- Classification: count the number of correct responses
- Regression: work out the squared error, ie:

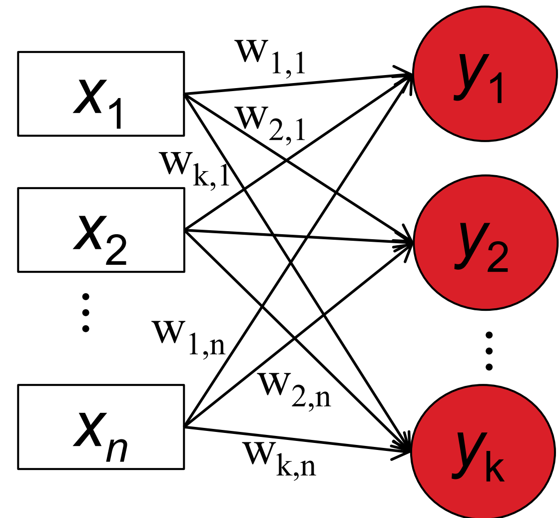
$$E = \sum_{i=1}^{n} \sum_{k=1}^{m} E_{ik} = \sum_{i=1}^{n} \sum_{k=1}^{m} [y_{ik} - f_{network,k}(x_i)]^2$$




- where *n* is the number of training patterns (observations), *m* is the number of outputs, $y_{ik}$ is the *k*th correct output label for pattern *i* and $f_{network,k}(x)$ is the *k*th element of the overall neural network function

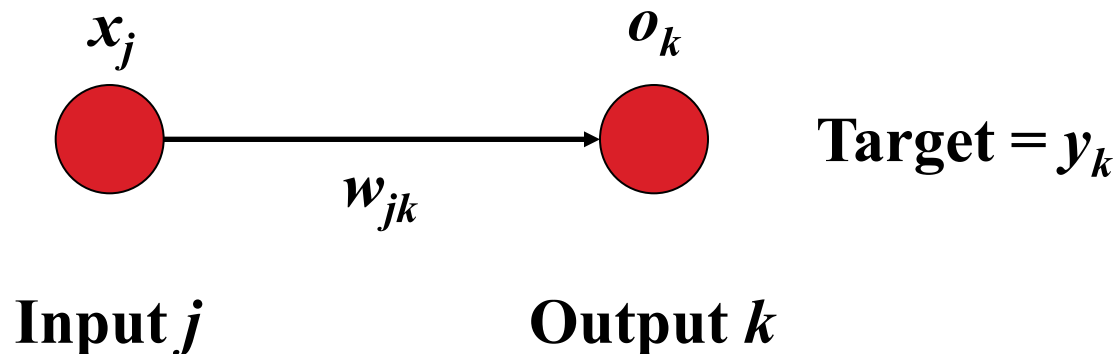
# Finding useful weights

- Use optimisation to find the set of weights which minimises *E*
- Global optimum set of weights would have for every weight: $\frac{\partial E}{\partial w} = 0$
- Unfortunately, we can't solve that equation
- Let's now ignore the state space, we need to search in the weight space (which is typically a lot bigger)
- The partial derivatives with respect to each weight give us a direction of steepest descent in our quest to minimise *E*



# Perceptron Learning Rule (1)

- Adjusting weights to minimise the error at the output of the network
- $y_k$ = target of output $k$
- $o_k$ = actual output $k$
- $x_j$ = input $j$
- $w_{jk}$ = weight between input $j$ and output $k$



# Perceptron Learning Rule (2)

- If $o_k = y_k$
  - No weight change needed
- If $o_k = 1$, $y_k = 0$, $y_k - o_k = -1$
  - Weighted input to output k is too large

$$\Delta w_{jk} = -\eta x_j$$

- If $o_k = 0$, $y_k = 1$, $y_k - o_k = +1$
  - Weighted input to output k is too small

$$\Delta w_{jk} = +\eta x_j$$

- Gives us:

$$\Delta w_{jk} = (y_k - o_k)\eta x_j$$

# Learning: guided by performance

- Measure of performance: sum squared error

$$E = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} E_{ik} = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} [y_{ik} - f_{network,k}(x_i)]^2$$

- $y_{ik}$ is the *k*th correct output label (value) for training pattern *i*
- $f_{network,k}(x)$ is the *k*th element of the overall neural network function (i.e. *k*th output $o_{ik}$)

# Learning: error rates

$$E = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} E_{ik} = \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} [y_{ik} - f_{network,k}(x_i)]^2$$

- Can divide *E* by the number of training patterns $n_{tr}$ to get average squared error per observation
- Textbook has

$$E = \frac{1}{2} \sum_{i=1}^{n_{tr}} \sum_{k=1}^{m} E_{ik}$$

- Constant multipliers don't really matter – absorbed in learning rate $\eta$ during the weight update:

$$w = w - \eta \frac{\partial E}{\partial w} = w + \Delta w$$

# Learning: optimisation

- The neural network models the training set best when the error is minimised
- Parameters:
  - weights
  - biases
  - (number of hidden units)
- Assume that each output is equally important
  - Could weight the errors from different outputs

# Optimisation Algorithms

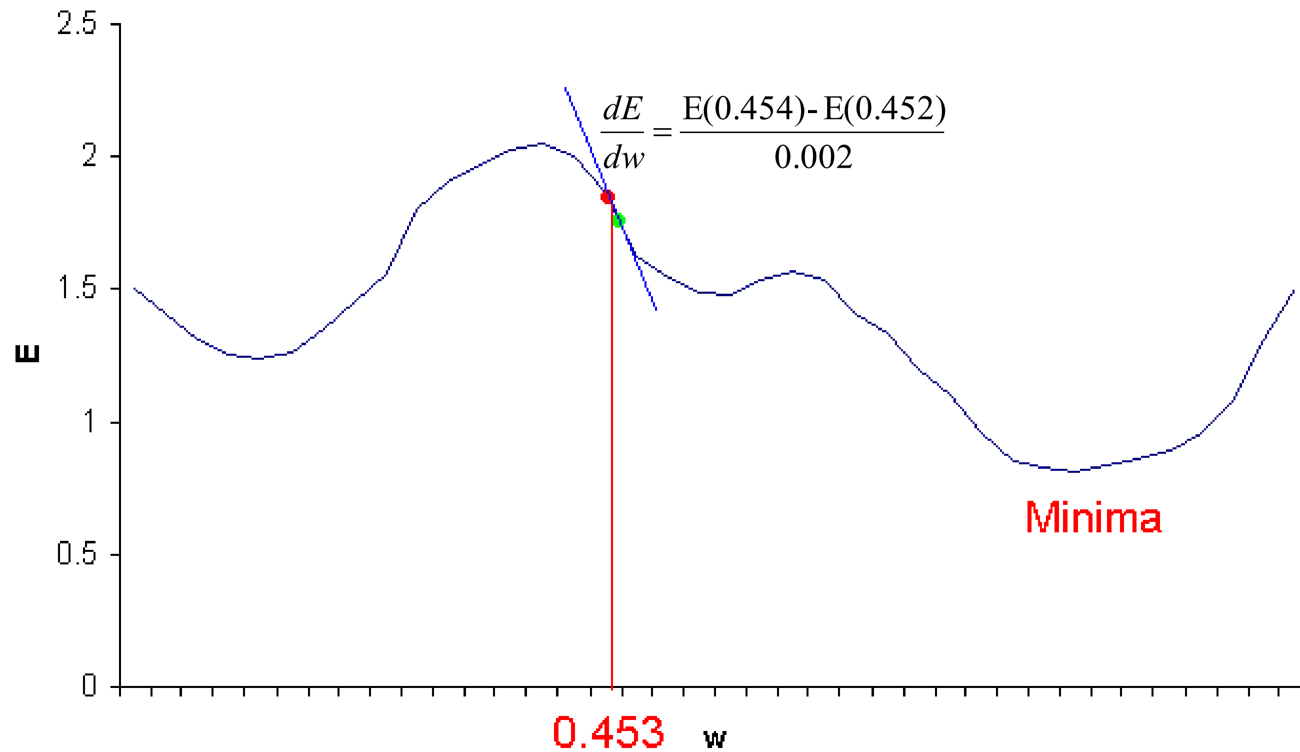
- Gradient descent
- $2^{nd}$ order methods
- Evolutionary algorithms
- Expectation-maximisation algorithm
- Simulated annealing

# Gradient Descent

- Gradient is often available, so it makes sense to use it (follow the gradient downhill)

# Numerical derivation

$$\Delta w \propto -\frac{\partial E}{\partial W}$$



# Basic gradient descent learning

- Minimise the error using the following:
  - Choose a random set of initial weights, e.g. from U[-0.5,0.5] (uniform distribution)
  - Repeat:
    - Run an input training pattern through the network and record the activations of hidden units and output units
    - Calculate *E* for the current weights.
    - Update every weight via $w = w - \eta \frac{\partial E}{\partial W}$
    -
  - Until stop condition, e.g. happy with performance or nothing much changing
- $\eta > 0$ is the learning rate, typically 0.1

# Gradient Descent in Single Layer Neural Networks

$$w_{jk,new} = w_{jk} + \eta \times E_k \times f'(z_k) \times x_j$$

- $w_{jk,new}$ is the updated weight between input *j* and output *k*
- $w_{jk}$ is the current weight between input *j* and output *k*
- $\eta$ is the learning rate
- $E_k$ is the error for output *k*

  $E_k = (y_k - f(z_k))$

- $z_k$ is the weighted input to output *k*

$$z_k = \sum_{j=0}^{n} x_j w_{jk}$$

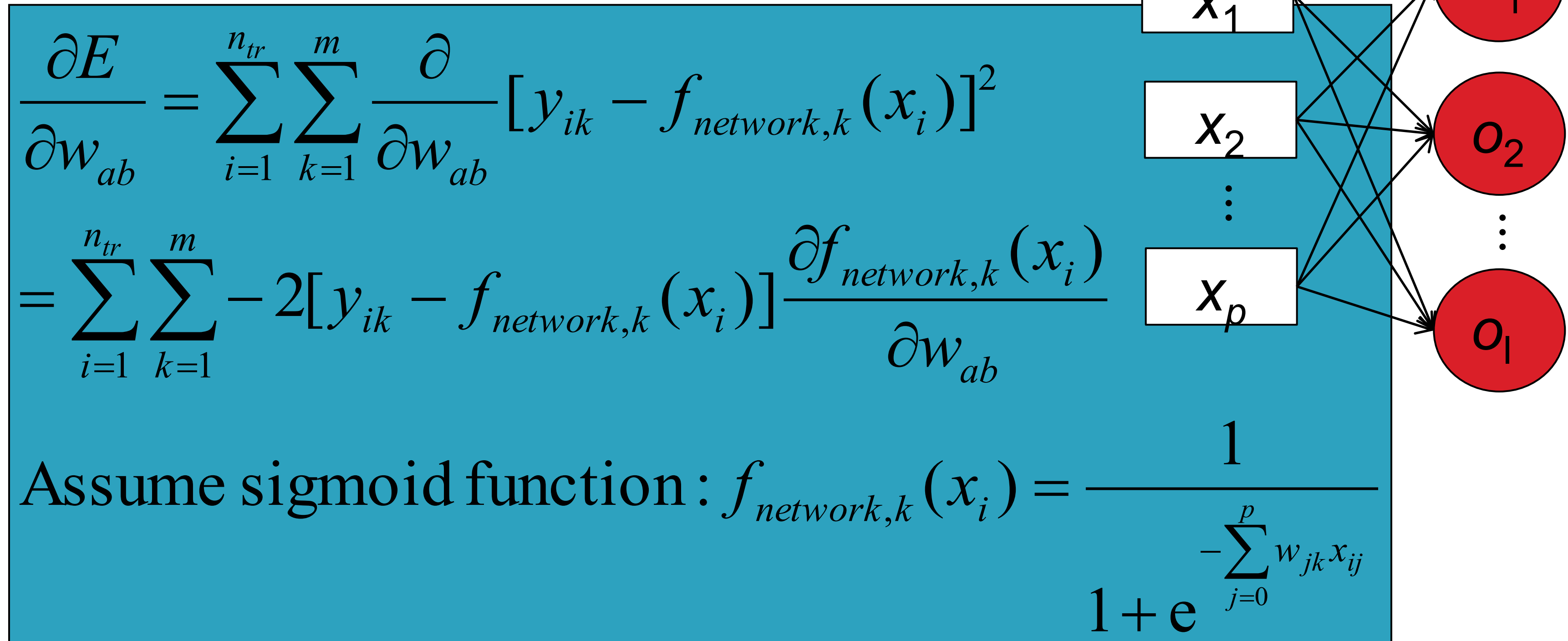
- $f'(z_k)$ is the derivative of the activation function $f(z_k)$

  *For the sigmoid function* $f'(z_k) = f(z_k)(1-f(z_k))$

- $x_j$ is the activation of the input *j*

# Single layer: continuous activation function

- The neural network has just one layer of weights – no hidden units




- Note: including bias weights as $w_{0k}$ ($j$=0): $x_{i0}$=1 for all training patterns $i$

# Single layer: continuous activation function

- So:

$$\frac{\partial f_{network,k}(x_i)}{\partial w_{ab}} = \begin{cases} \dfrac{-1.-1x_{ia}}{\left(1+e^{-\sum_{j=0}^{p} w_{jk}x_{ij}}\right)^2}, b = k \\ 0, b \neq k \end{cases}$$

$$\text{So } \frac{\partial E}{\partial w_{ab}} = -2\sum_{i=1}^{n_{tr}} \frac{[y_{ib} - f_{network,b}(x_i)]x_{ia}}{\left(1+e^{-\sum_{j=0}^{p} w_{jk}x_{ij}}\right)^2}$$

$$= -2\sum_{i=1}^{n_{tr}} [y_{ib} - o_{ib})]o_{ib}{}^2 x_{ia}$$

$$\text{Thus } \Delta w_{ab} = 2\eta\sum_{i=1}^{n_{tr}} [y_{ib} - o_{ib}]o_{ib}{}^2 x_{ia}$$

# Summary

- Applications
- History
- Units
- Decision Boundaries
- Learning