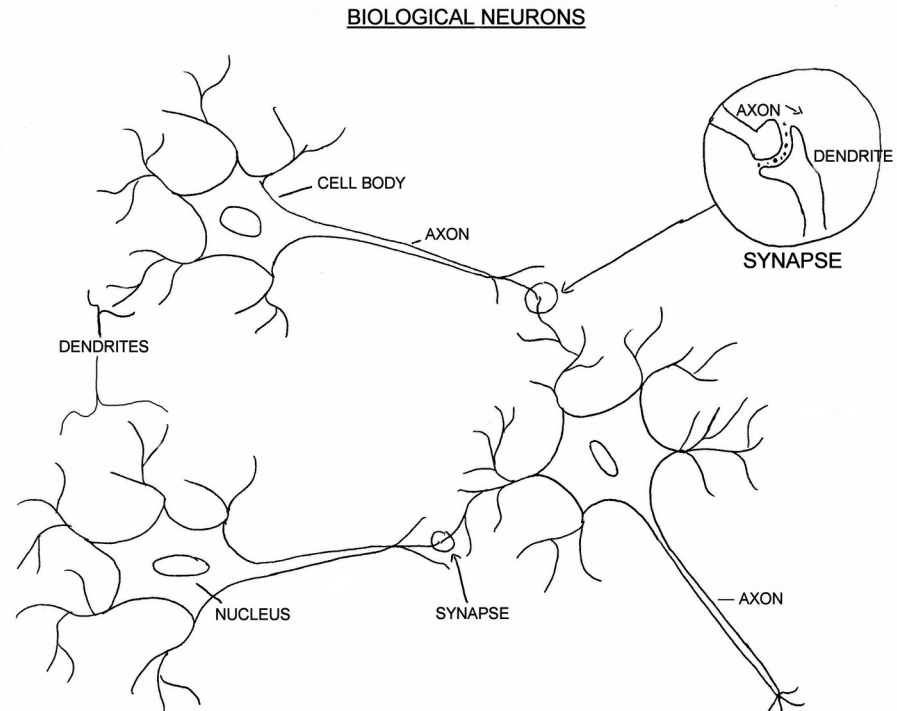


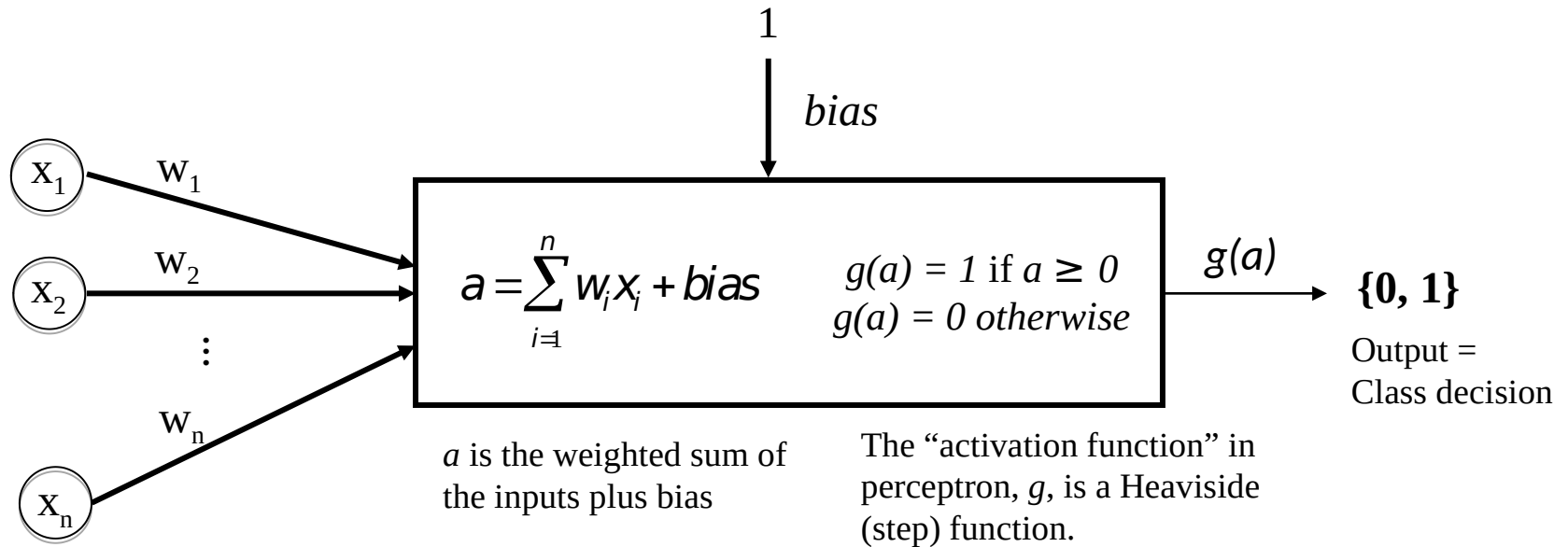
Artificial Neural Networks

Source of inspiration: (Biological) Neural Networks

- Brain as a network of “neurons”
- The interaction between them produces something interesting: intelligence
- The building blocks are neurons.
- **Neuron:**
 - Receives signal from multiple inputs
 - Strength of the signal is affected by “synaptic” weights.
 - If the overall signal is above a threshold, the neuron fires (sends a signal to its outputs.)



Modeling a Single Neuron: Perceptron



Inputs: x_1, x_2, \dots, x_n

Parameters: w_1, w_2, \dots, w_n and $bias$

Output: $g(a)$

Sometimes $bias$ is represented as another weight (w_0) in which case we assume there is a virtual input x_0 which is always 1:

$$a = \sum_{i=0}^n w_i x_i$$

How many parameters are required to specify a perceptron in an n -dimensional input space?

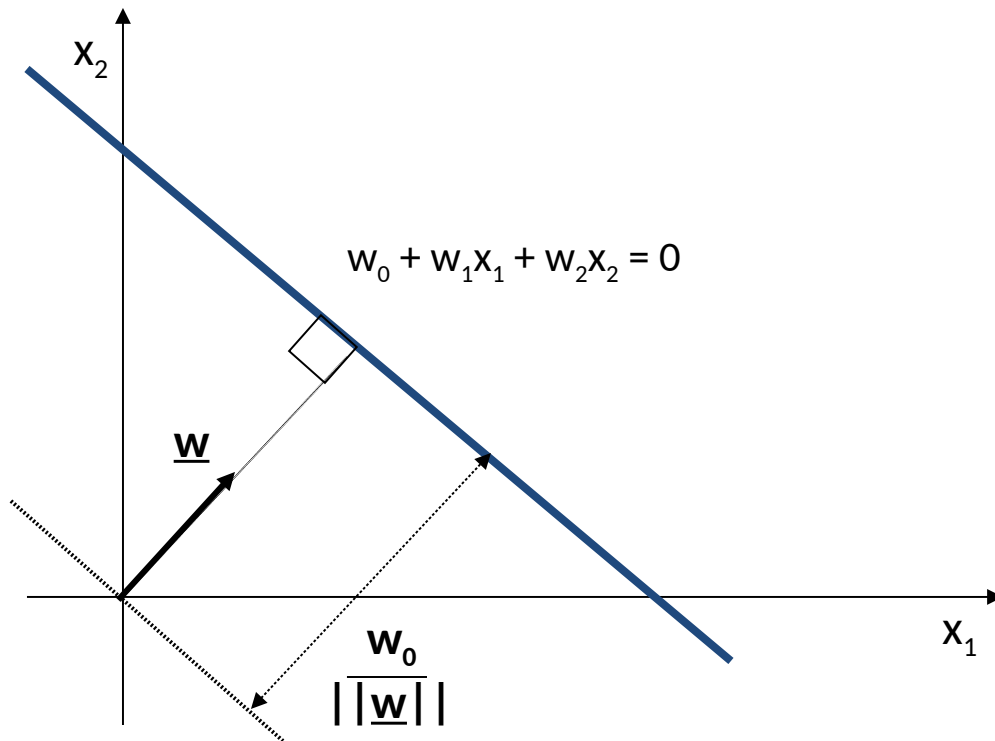
Application: Decision Making and Classification

- Perceptron can be seen as a predicate
 - given a vector \mathbf{x}
 - $f(\mathbf{x}) = 1$ if that predicate over \mathbf{x} is true
 - $f(\mathbf{x}) = 0$ if the predicate is false
- Therefore it can be used for decision making and binary classification problems:
 - spam vs non-spam
 - diseased vs healthy
- In binary classification:
 - given input vector \mathbf{x}
 - $f(\mathbf{x}) = 1$ if \mathbf{x} is in positive class
 - $f(\mathbf{x}) = 0$ if \mathbf{x} is in the negative class

Geometric interpretation of weights and bias

Since the transition in output happens at $a = 0$, the equation of the decision boundary is:

$$a = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n = 0$$

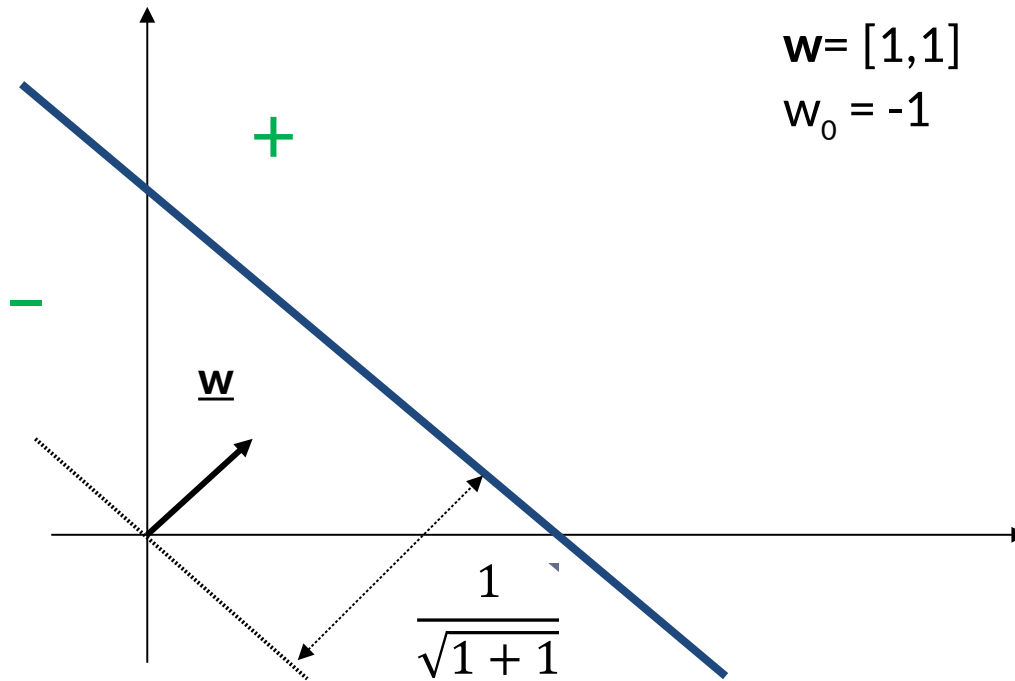


- How do we plot the decision boundary?
- What does a decision boundary look like in a problem with 3 input features (attributes)?

Notes on visualising the decision boundary

- An easy way of finding a linear decision boundary in 2D is to find two points that lie on it (e.g. set x_1 to zero and find x_2 and vice versa)
- The direction of the vector \mathbf{w} (without the bias) shows on which side of the hyper-plane patterns will be classified as positive (output will be 1).
- If w_0 (bias) is positive then the origin is on the positive side of the discriminator (line), otherwise on the negative side.
- **Example:** find the decision boundary of this perceptron:
 - Two inputs x_1 and x_2
 - The weights are $w_1 = 1.0$ and $w_2 = 1.0$
 - The bias $w_0 = -1.0$

Example: answer



Perceptron learning (for one neuron)

- **Given:** a data set - a collection of training vectors in the form $(x_1, x_2, \dots, x_n, t)$
- **We want:** a perceptron that models the data.
- Initialise the weights and bias (randomly).
- Iterate through (training) examples. Classify each example with current values of weights and bias. If the example is classified correctly, then don't do anything. If the example is misclassified then change the weights and bias:

$$w_j \leftarrow w_j + \eta x_j (t - y)$$

$$bias \leftarrow bias + \eta (t - y)$$

Where:

x_j : the value of j -th feature of the input pattern \mathbf{x}

w_j : the j -th weight

η : learning rate

t : target/desired value

y : perceptron output

Example

```
weights = [1.0, 1.0]
```

```
bias = -2.0
```

```
eta = 0.5 # learning rate
```

```
max_epochs = 500
```

```
examples = [  
    ((0, 4), 0),  
    ((-2, 1), 1),  
    ((3, 5), 0),  
    ((1, 1), 1),  
]
```

Example

```
-----  
epoch:  1  
weights: [1.0, 1.0]  
bias:   -2.0  
pattern, output, target: (0, 4), 1, 0  
updating the weights and bias to: [1.0, -1.0] -2.5  
pattern, output, target: (-2, 1), 0, 1  
updating the weights and bias to: [0.0, -0.5] -2.0  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [0.5, 0.0] -1.5  
  
-----  
epoch:  2  
weights: [0.5, 0.0]  
bias:   -1.5  
pattern, output, target: (0, 4), 0, 0  
pattern, output, target: (-2, 1), 0, 1  
updating the weights and bias to: [-0.5, 0.5] -1.0  
pattern, output, target: (3, 5), 1, 0  
updating the weights and bias to: [-2.0, -2.0] -1.5  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [-1.5, -1.5] -1.0
```

Example

```
-----  
epoch: 3  
weights: [-1.5, -1.5]  
bias: -1.0  
pattern, output, target: (0, 4), 0, 0  
pattern, output, target: (-2, 1), 1, 1  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [-1.0, -1.0] -0.5
```

```
-----  
epoch: 4  
weights: [-1.0, -1.0]  
bias: -0.5  
pattern, output, target: (0, 4), 0, 0  
pattern, output, target: (-2, 1), 1, 1  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [-0.5, -0.5] 0.0
```

Example

```
-----  
epoch: 5  
weights: [-0.5, -0.5]  
bias: 0.0  
pattern, output, target: (0, 4), 0, 0  
pattern, output, target: (-2, 1), 1, 1  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [0.0, 0.0] 0.5
```

```
-----  
epoch: 6  
weights: [0.0, 0.0]  
bias: 0.5  
pattern, output, target: (0, 4), 1, 0  
updating the weights and bias to: [0.0, -2.0] 0.0  
pattern, output, target: (-2, 1), 0, 1  
updating the weights and bias to: [-1.0, -1.5] 0.5  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [-0.5, -1.0] 1.0
```

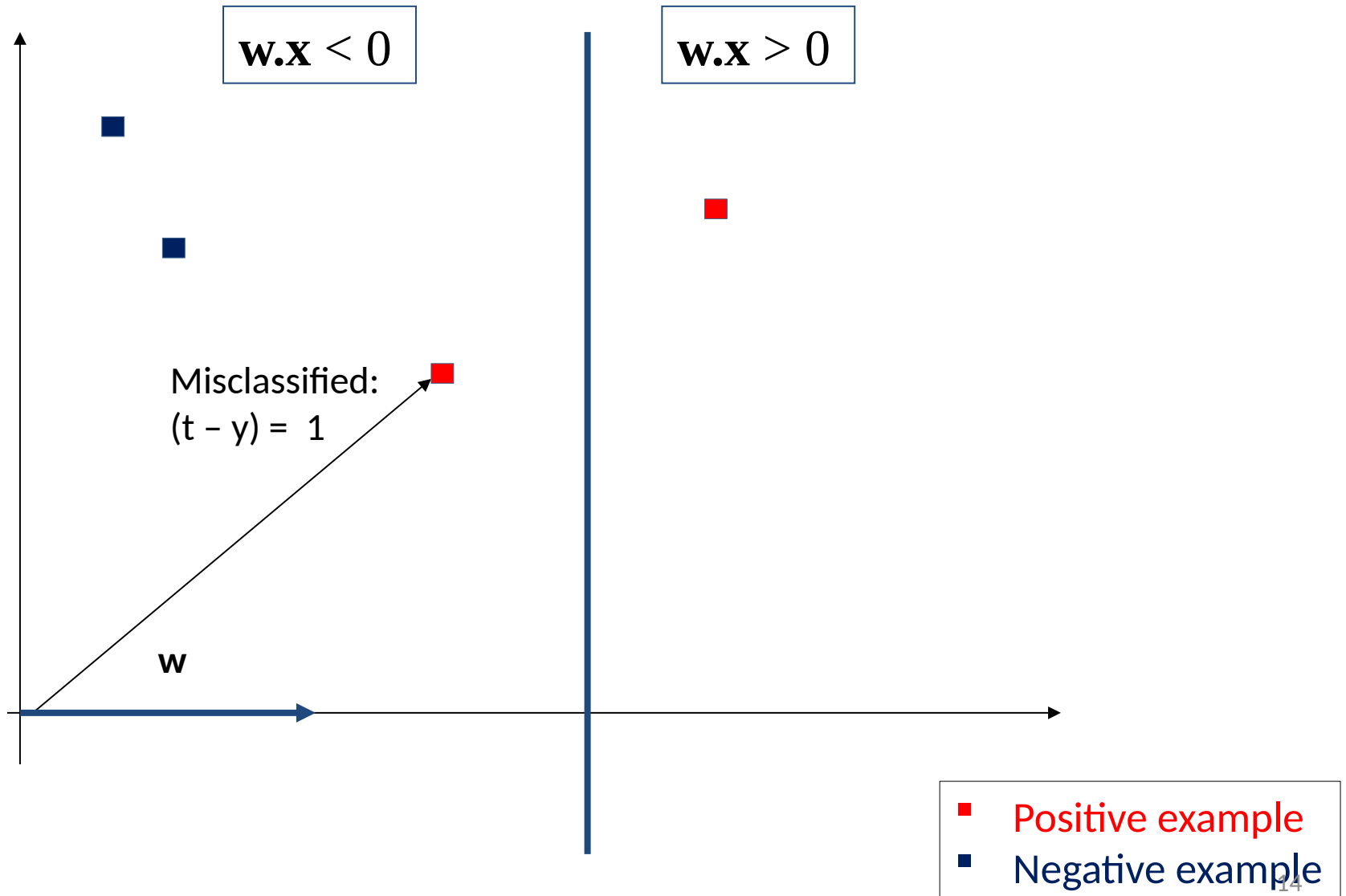
Example

```
-----  
epoch: 7  
weights: [-0.5, -1.0]  
bias: 1.0  
pattern, output, target: (0, 4), 0, 0  
pattern, output, target: (-2, 1), 1, 1  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 0, 1  
updating the weights and bias to: [0.0, -0.5] 1.5
```

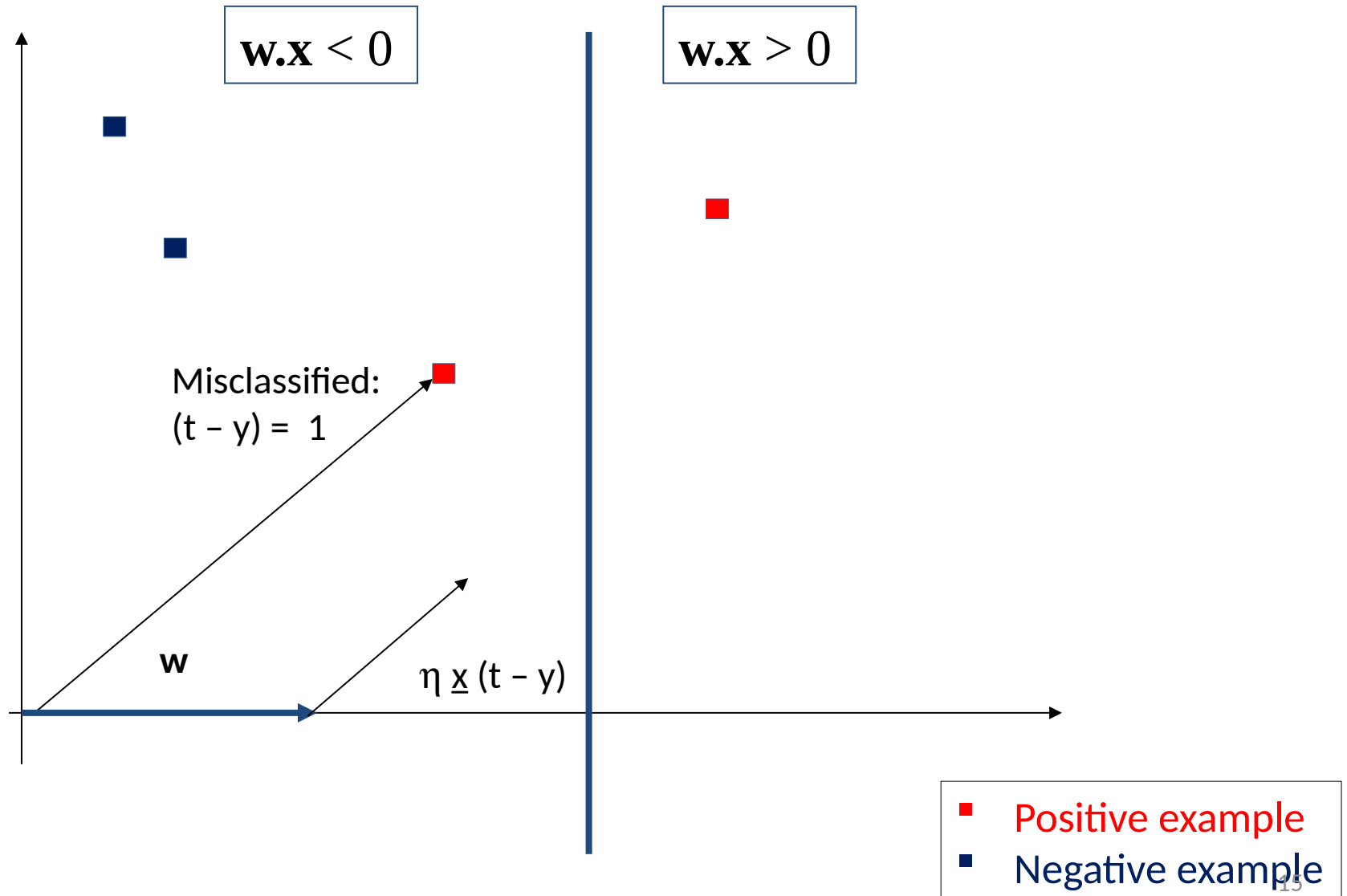
```
-----  
epoch: 8  
weights: [0.0, -0.5]  
bias: 1.5  
pattern, output, target: (0, 4), 0, 0  
pattern, output, target: (-2, 1), 1, 1  
pattern, output, target: (3, 5), 0, 0  
pattern, output, target: (1, 1), 1, 1
```

If examples are linearly separable, then using perceptron learning, the weights and bias will (in finite time) converge to values that produce a perfect separation.

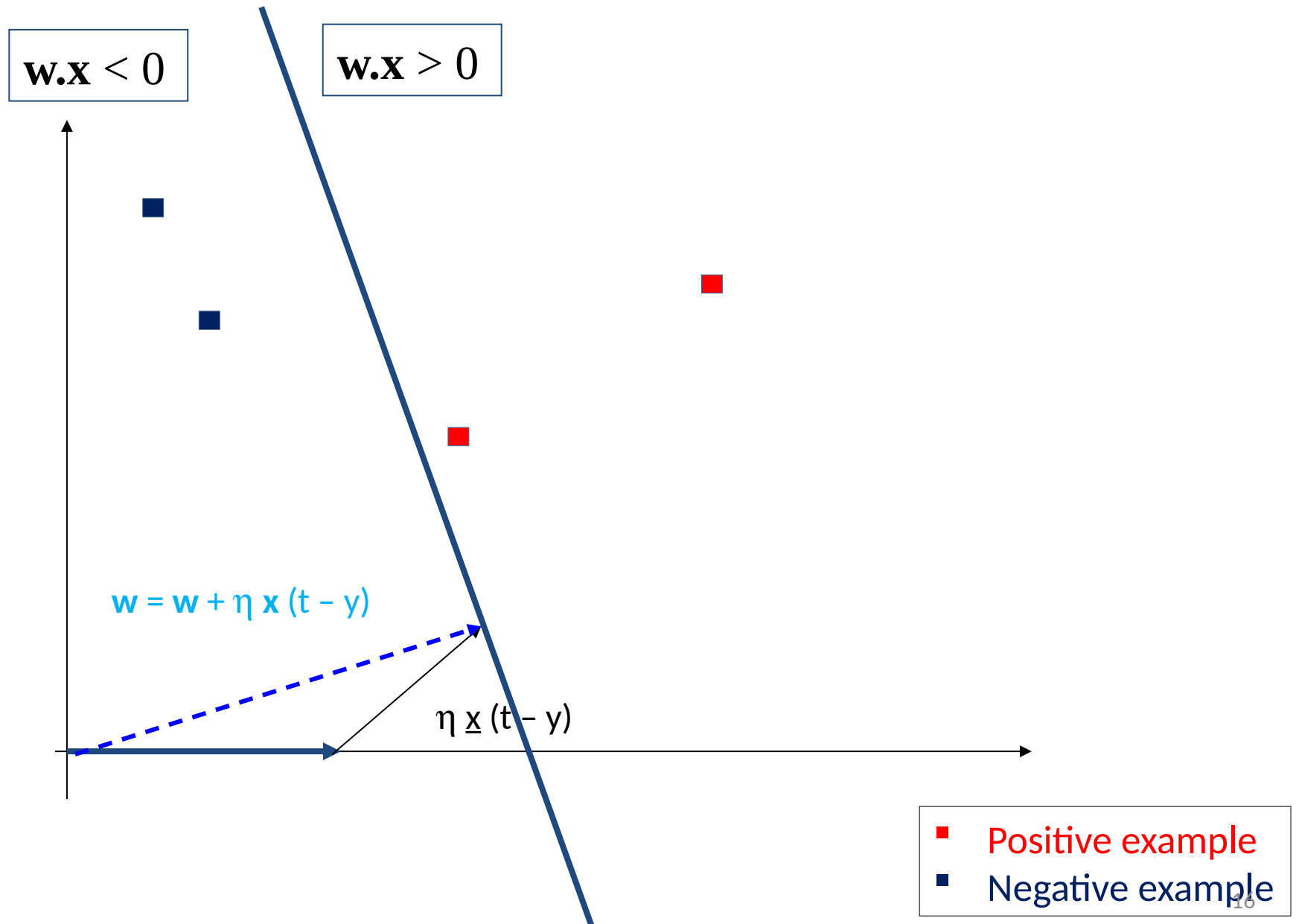
Geometric interpretation of weight update



Geometric interpretation of weight update

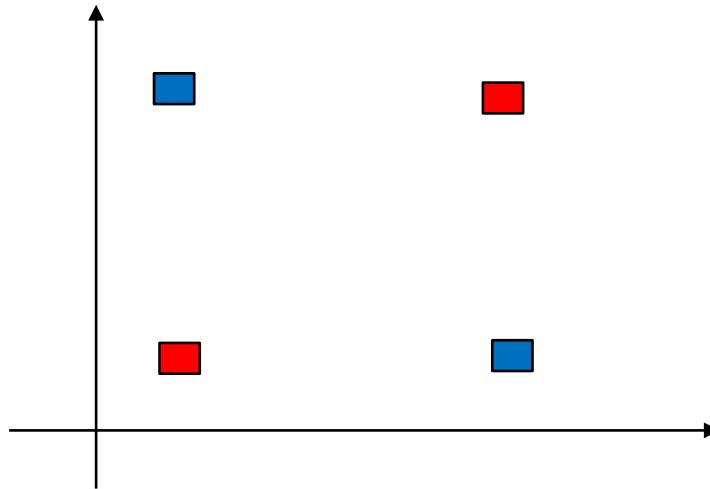


Geometric interpretation of weight update



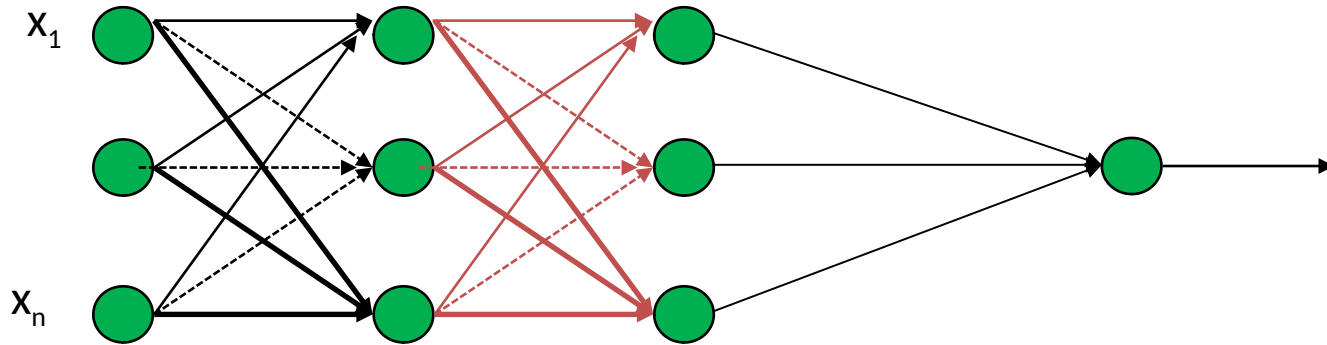
Limitations of single perceptron

*The “XOR”
problem*



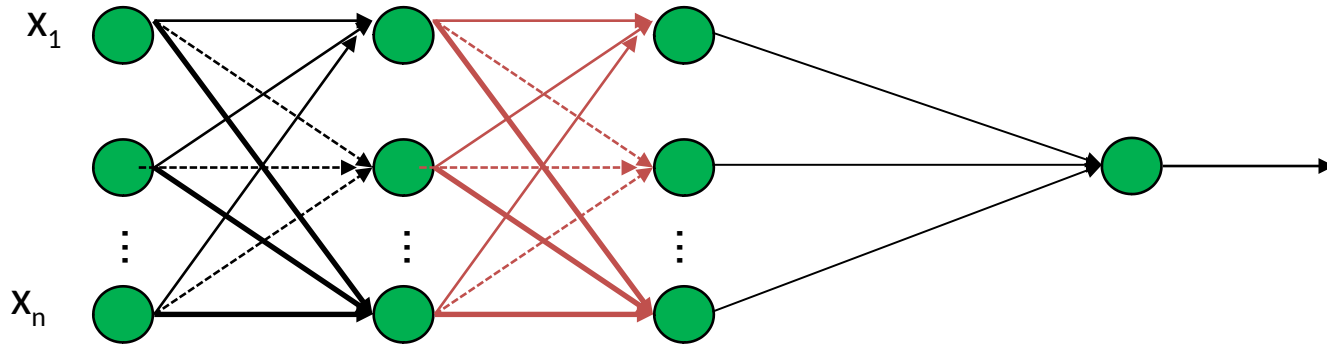
Can a single perceptron learn this problem?

Solution: Multi Layer Perceptrons (MLPs)



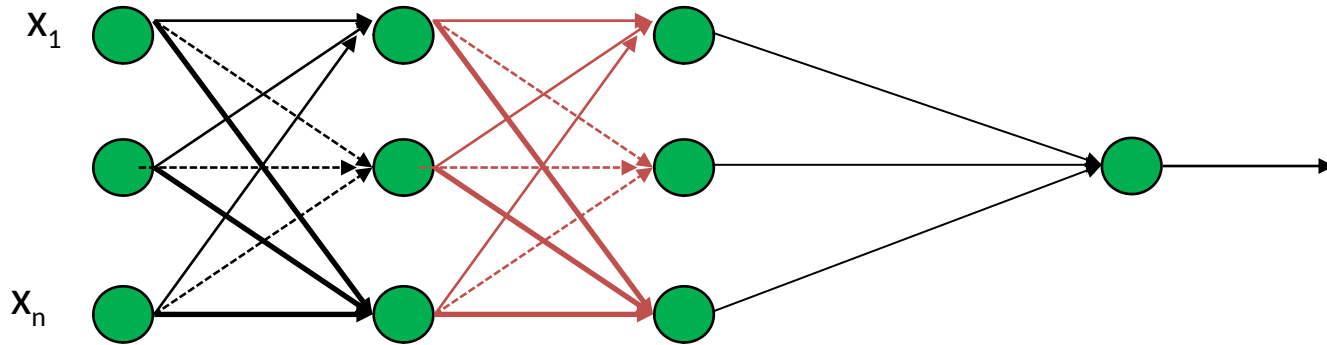
- Instead of a single perceptron, use a whole network of them!
- The output of a node becomes an input for another node.
- The network is usually arranged in layers. The data flows through layers, one layer at a time.
- As a whole, the network acts like a function taking a vector of input data and outputting a scalar (or vector).
- Can be used for regression or classification.

MLPs (2)



- The network function is computed layer by layer.
- The first layer of the network (drawn on the left) is called the **input layer**.
- The number of units in the input layer is equal to the number of features in the problem domain. For each input vector, the i -th input unit returns the value of the i -th feature (no weight or function is applied).
- The last layer of the network (the right most layer) is called the **output layer**.
- The number of output nodes depends on how we have encoded the output.
- Layers between the input and output (for networks with three layers or more) are called **hidden layers**.
- All nodes (units) in all layers except the input layer are Perceptrons.
- A network with only one layer (acting as input and output) is an identity function.

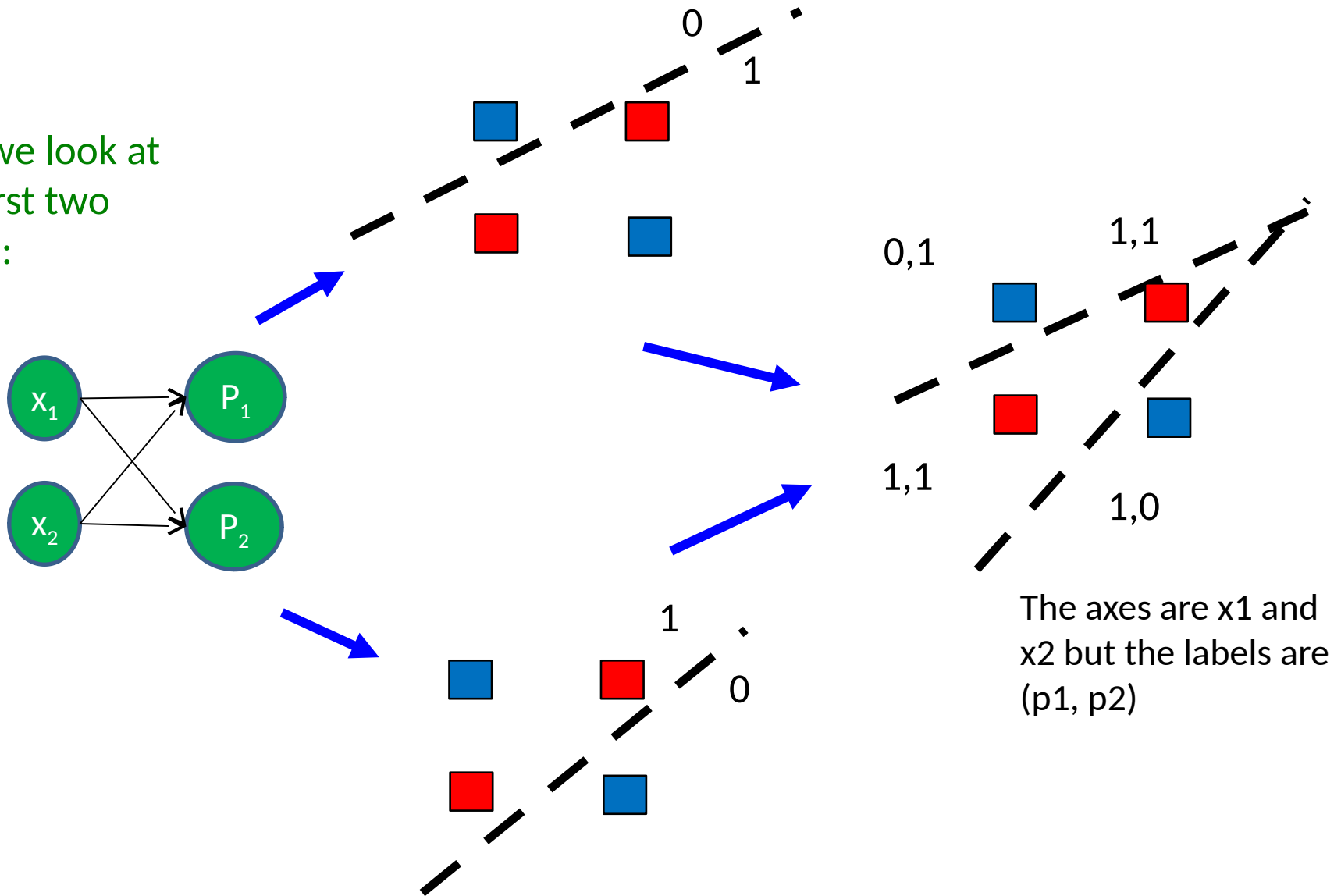
MLPs (3)



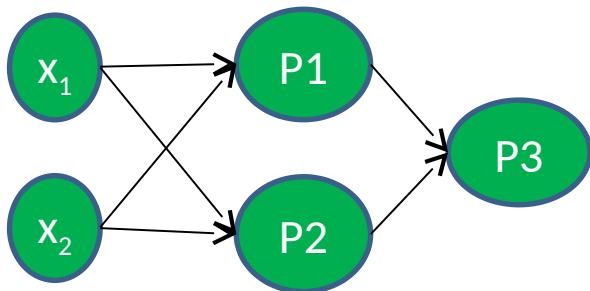
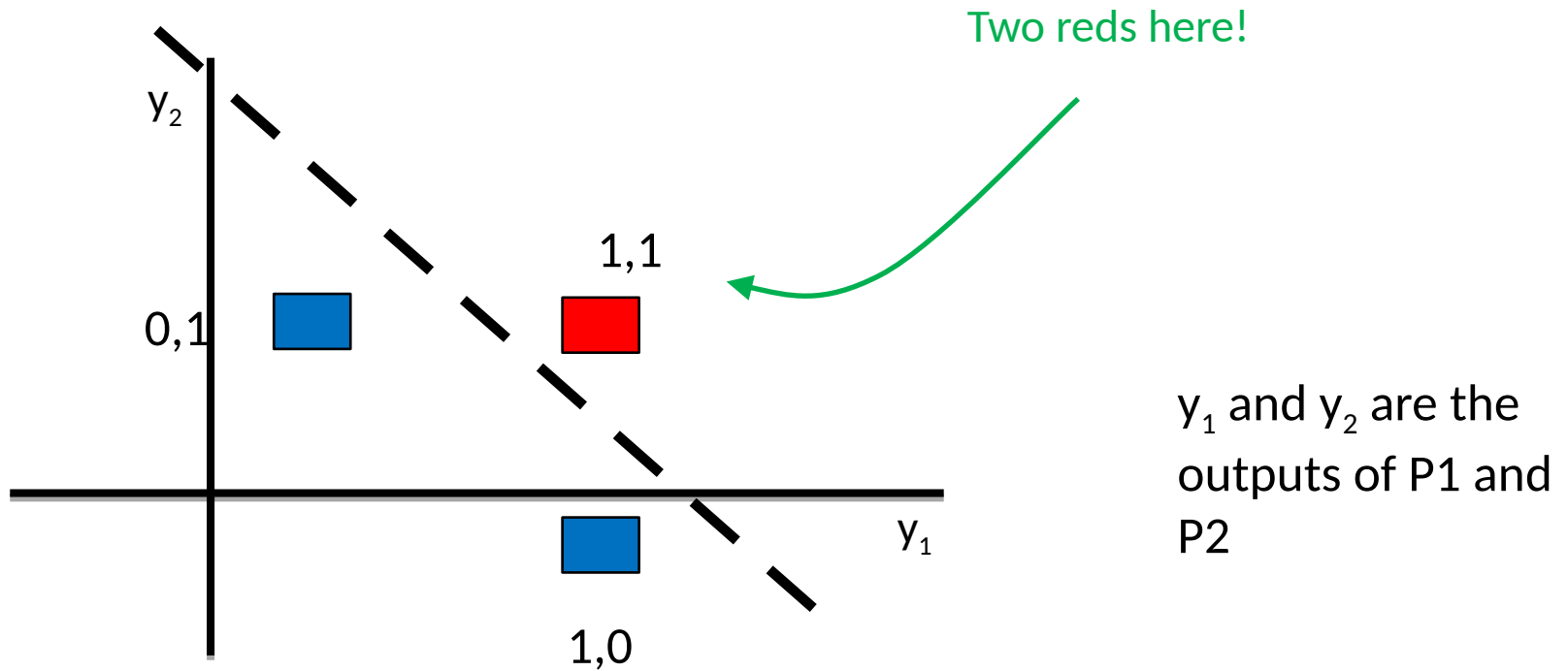
- Adjacent layers are fully connected.
- There is no backward connection.
- There is not shortcut to the right layers (layers cannot be skipped)
- Every unit (node) that is a Perceptron has a bias parameter.
- Each Perceptron has as many weight parameters as the number of its inputs (the number of nodes in the previous layer).
- In designing neural networks, the number of hidden layers and the number of nodes in the hidden layers depends on the complexity of the problem.

Example: using a 3-layer MLP for the “XOR” problem

First we look at
the first two
layers:

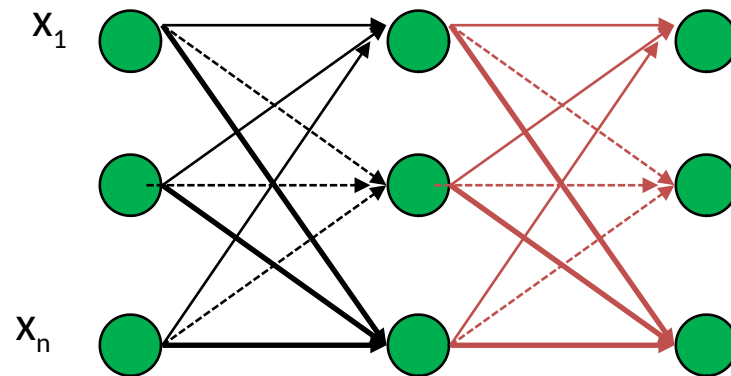


Example cont'd

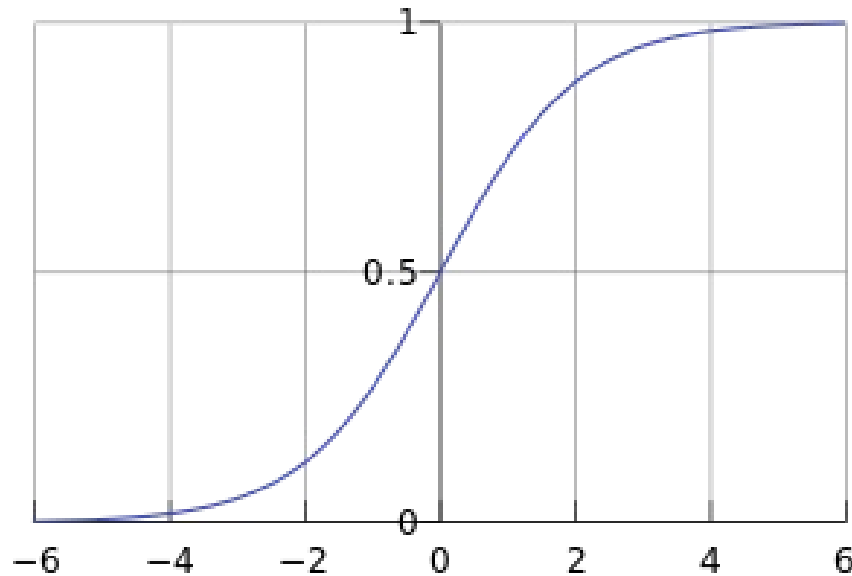


Now in the last layer (from perceptron 3 point of view) the examples are linearly separable!

How about multi-class classification?



Using a Sigmoid Function



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

- It is differentiable at all points.
- It handles uncertainty (e.g. an input example could be positive with different degrees of certainty from 0.5 to 1).

Learning in neural networks: The Error function

Mean squared error is typically used to measure error:

$$E = \sum_{i=1}^n (t_i - y_i)^2$$

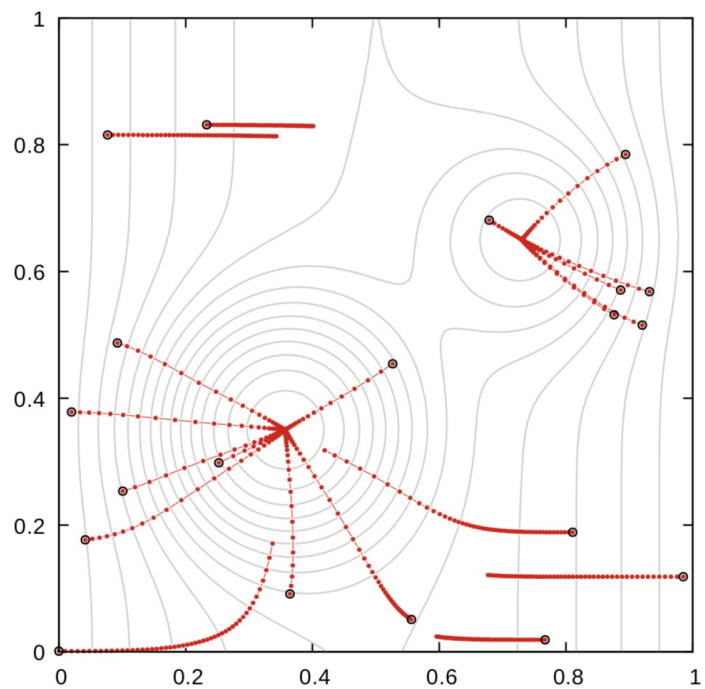
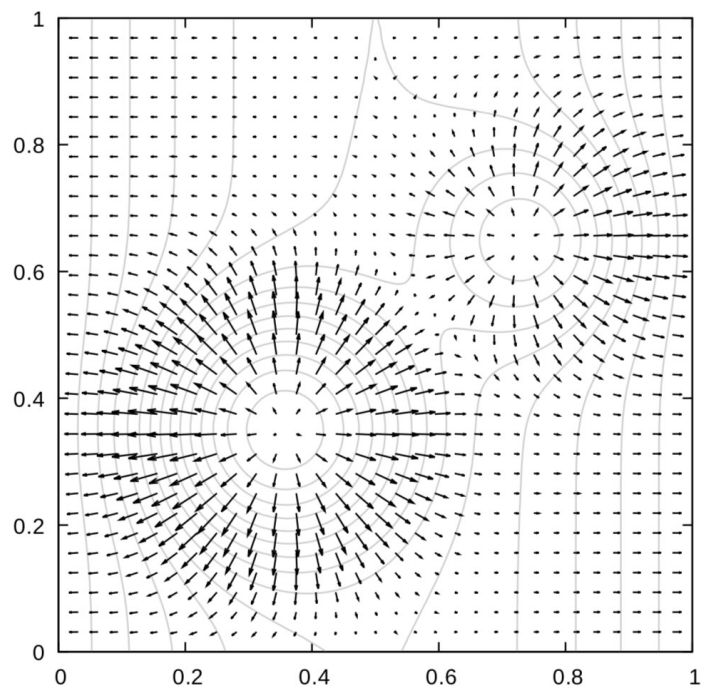
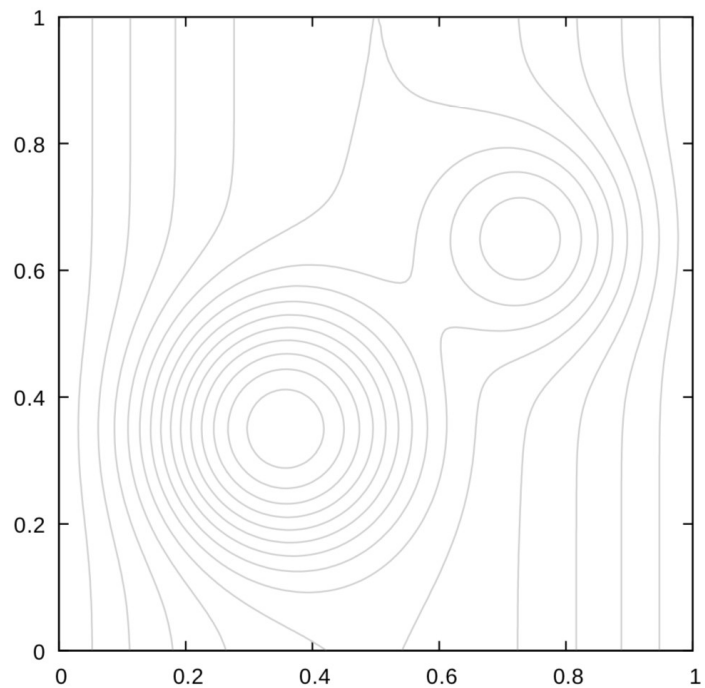
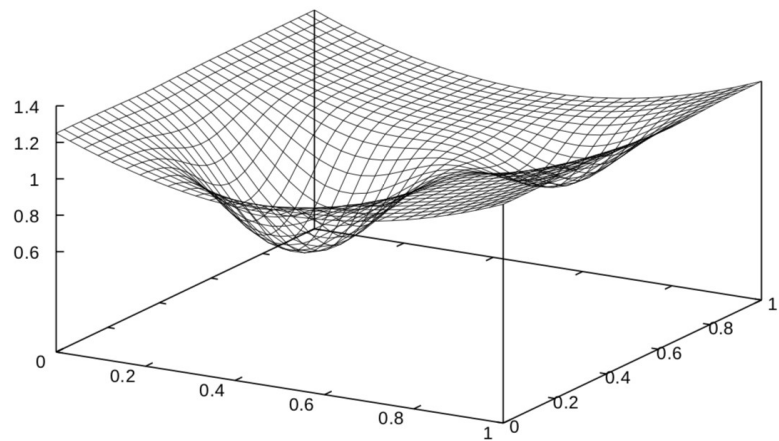
Where t_i is the desired output (according to the training) data, y_i is the output of the network, and n is the number of patterns (examples) in the training set.

Learning: search for optimal weights

- The aim is to minimise an error function over all training patterns by adapting weights.
- The weight can be updated incrementally:

$$W \leftarrow W - \eta \nabla E(W)$$

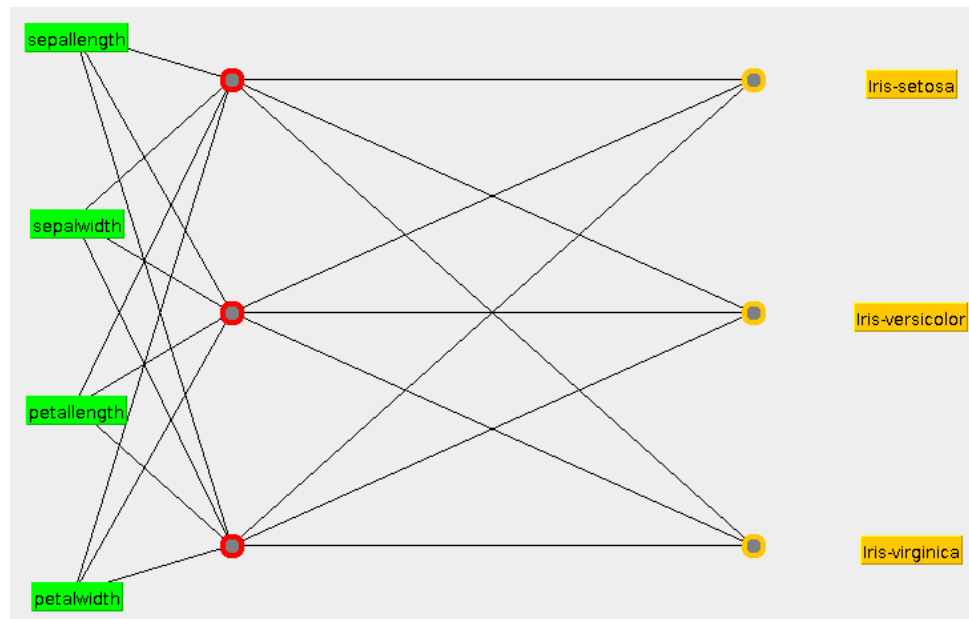
- The gradient is a vector of partial derivatives.
- The learning rate (step size) determines how much is made in each iteration.



Network architecture for a typical classification task

- The numbers of inputs and outputs is determined by the problem.
- One hidden layer is enough for many classification tasks. Therefore, there will be a layer of input nodes (equal to the number of attributes in the problem), a layer of hidden neurons, and a layer of output neurons. This means that there are two matrices of weights: one for the hidden layer and the other for the output layer.

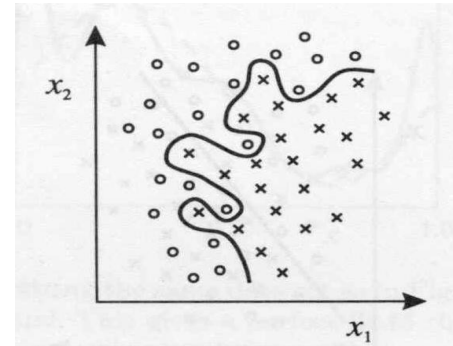
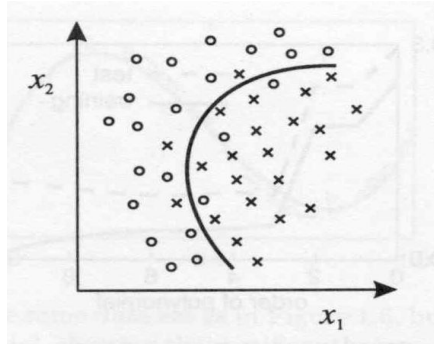
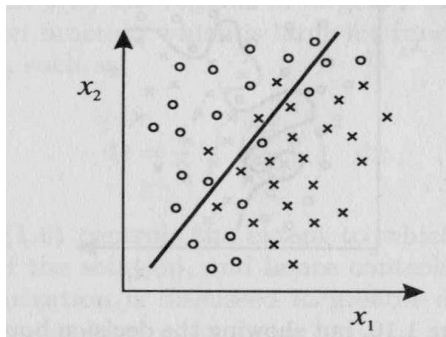
4
attributes



3 Classes

A Rough Guideline for network size

- Best is to have as few hidden layers/nodes as possible
 - Forces better generalization
 - Fewer weights to be found
 - Note: too many nodes results in too much flexibility and possibly overfitting.



- Number of nodes in the hidden layer:
 - Make the best guess you can (e.g. a number between the number of input and output neurons)
 - If training is unsuccessful try more hidden nodes
 - If training is successful try fewer hidden nodes