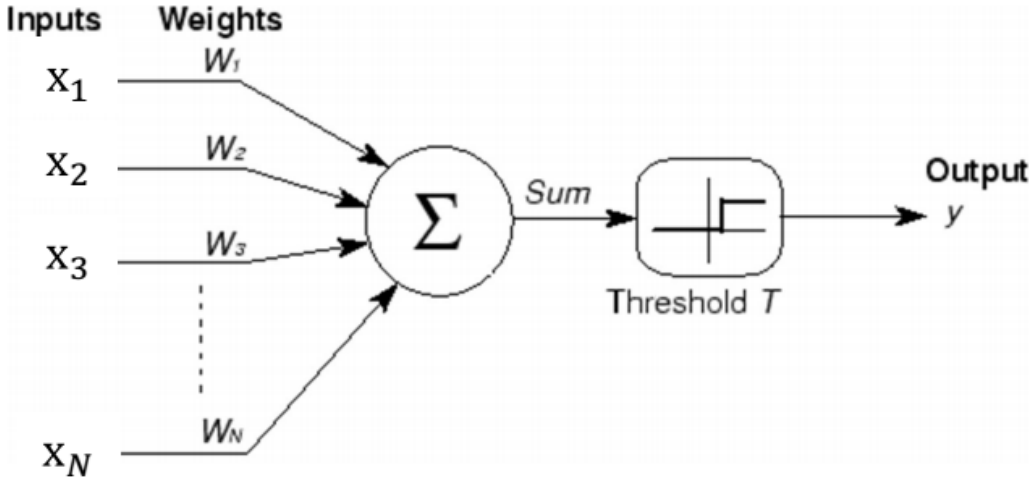
# COMP417 Lecture 4

## Learning the Network

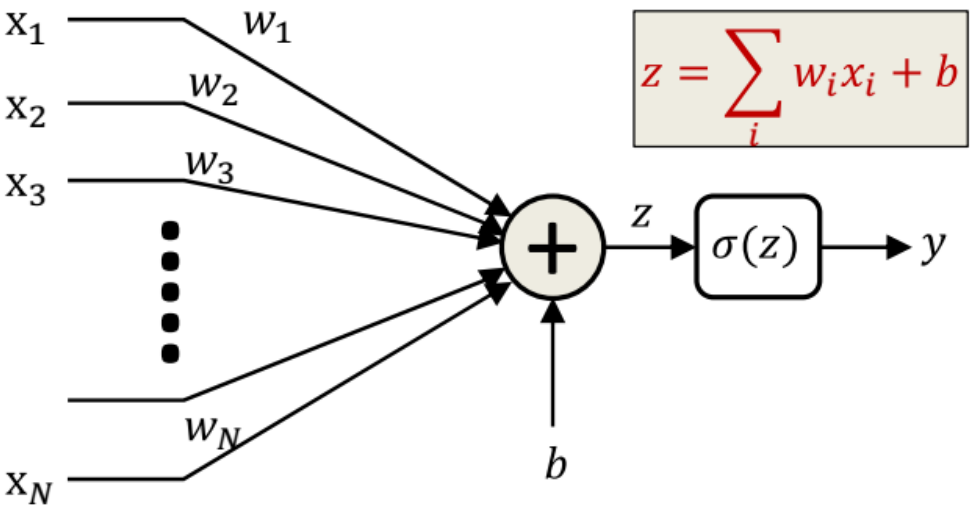
Dr. Hend Dawood

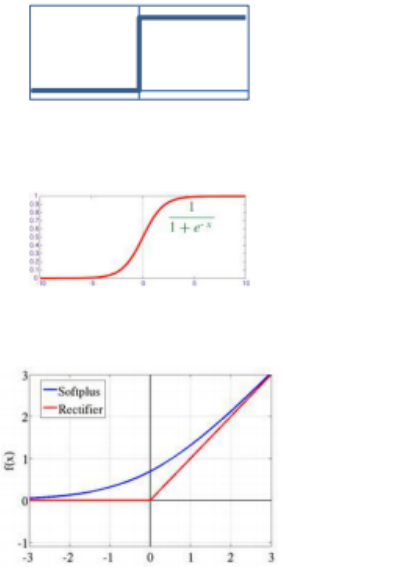
# The original perceptron




- Simple threshold unit
  - Unit comprises a set of weights and a threshold

# The original perceptron



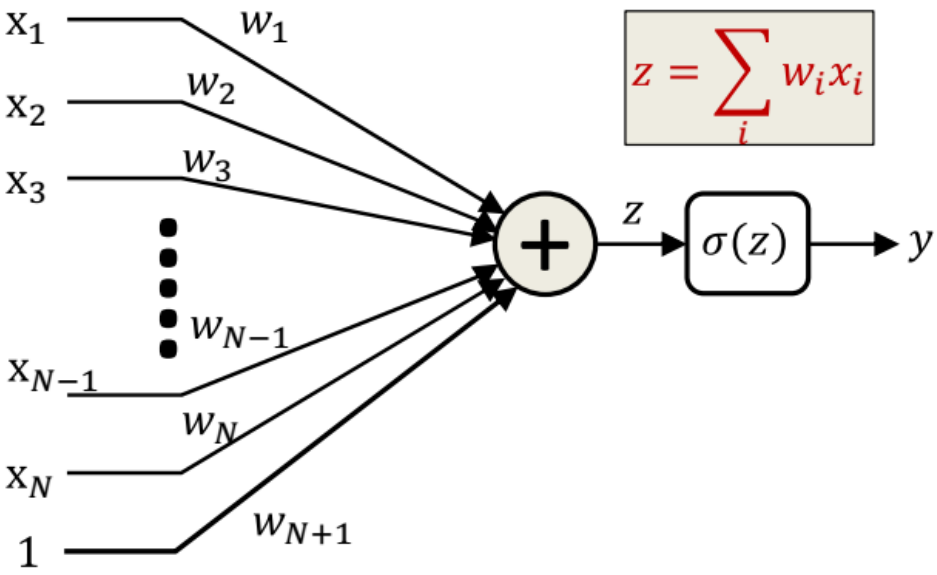



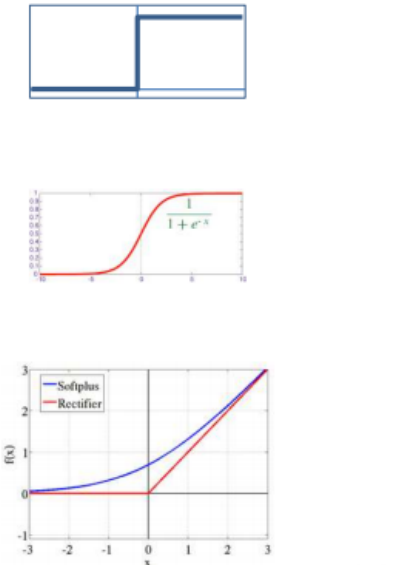

*Activation functions* $\sigma(z)$

- Perceptron
  - General setting, inputs are real-valued
  - A *bias* $b$ representing a threshold to trigger the perceptron
  - Activation functions are not necessarily threshold functions
- The parameters of the perceptron (which determine how it behaves) are its weights and bias

# The original perceptron

## Redrawing the neuron



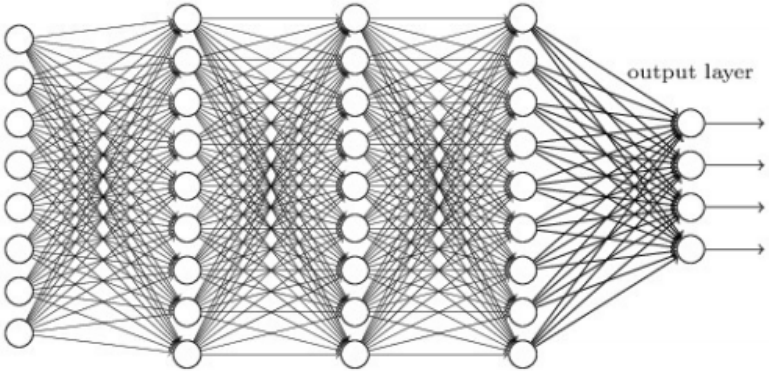



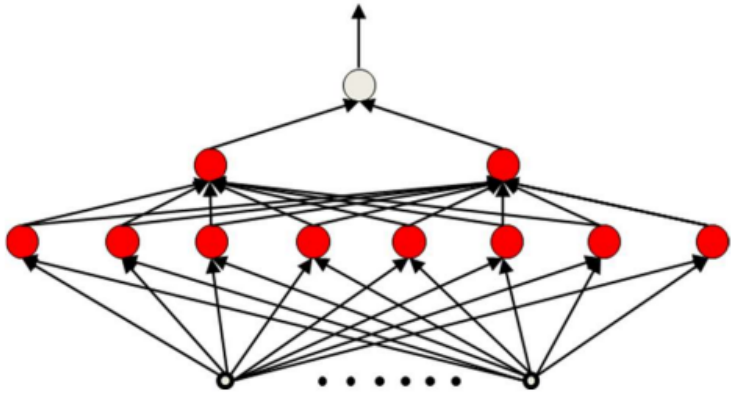

*Activation functions* $\sigma(z)$

- The bias can also be viewed as the weight of another input component that is always set to 1
  - If the bias is not explicitly mentioned, we will implicitly be assuming that every perceptron has an additional input that is always fixed at 1

# The original perceptron

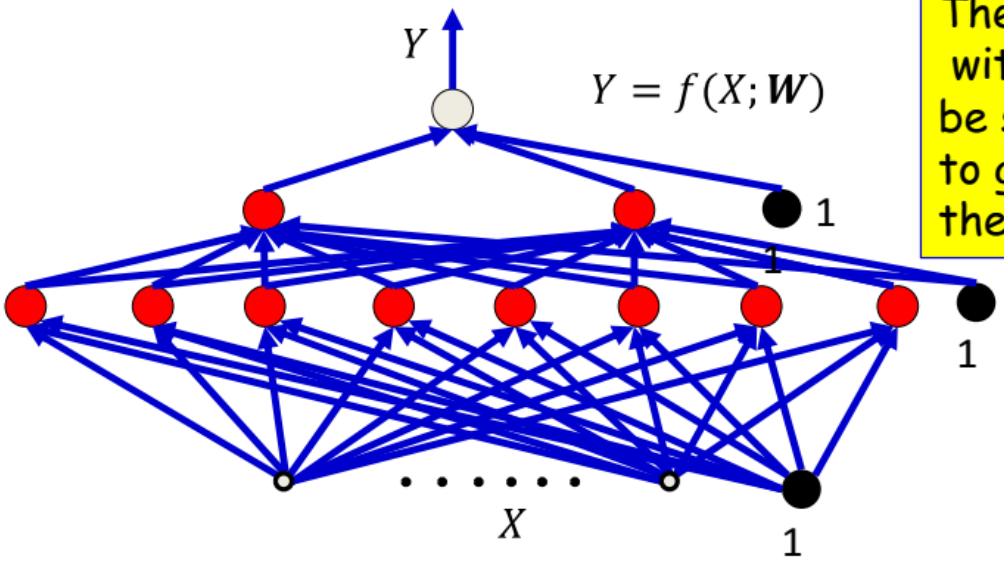
## the structure of the network



- We will assume a *feed-forward* network
  - No loops: Neuron outputs do not feed back to their inputs directly or indirectly
  - Loopy networks are a future topic
- **Part of the design of a network: The architecture**
  - How many layers/neurons, which neuron connects to which and how, etc.
- For now, assume the architecture of the network is capable of representing the needed function

# What we learn: The parameters of the network

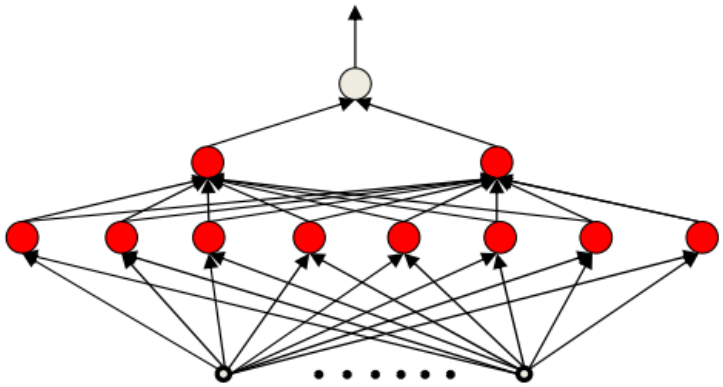


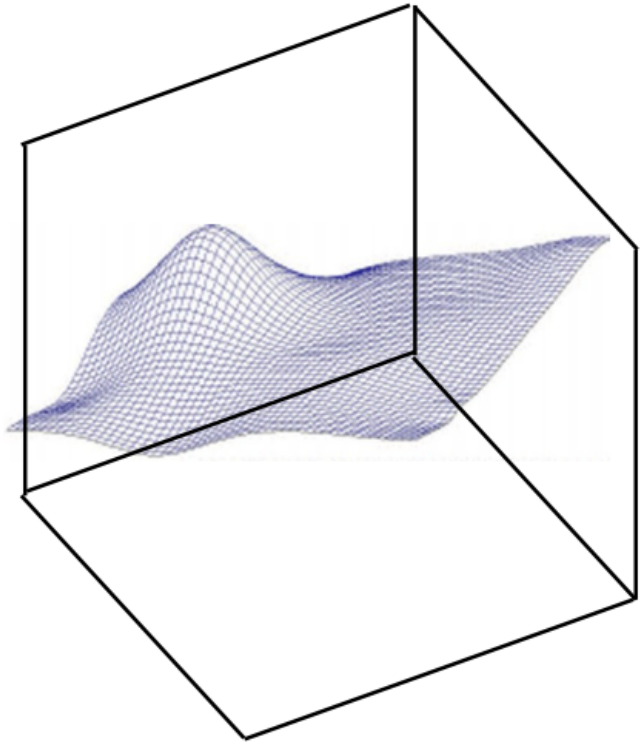

The network is a function f() with parameters W which must be set to the appropriate values to get the desired behavior from the net

- **Given:** the architecture of the network
- The parameters of the network: The weights and biases
  - The weights associated with the blue arrows in the picture
- ***Learning the network :*** **Determining the values of these parameters such that the network computes the desired function**

# What we learn: The parameters of the network

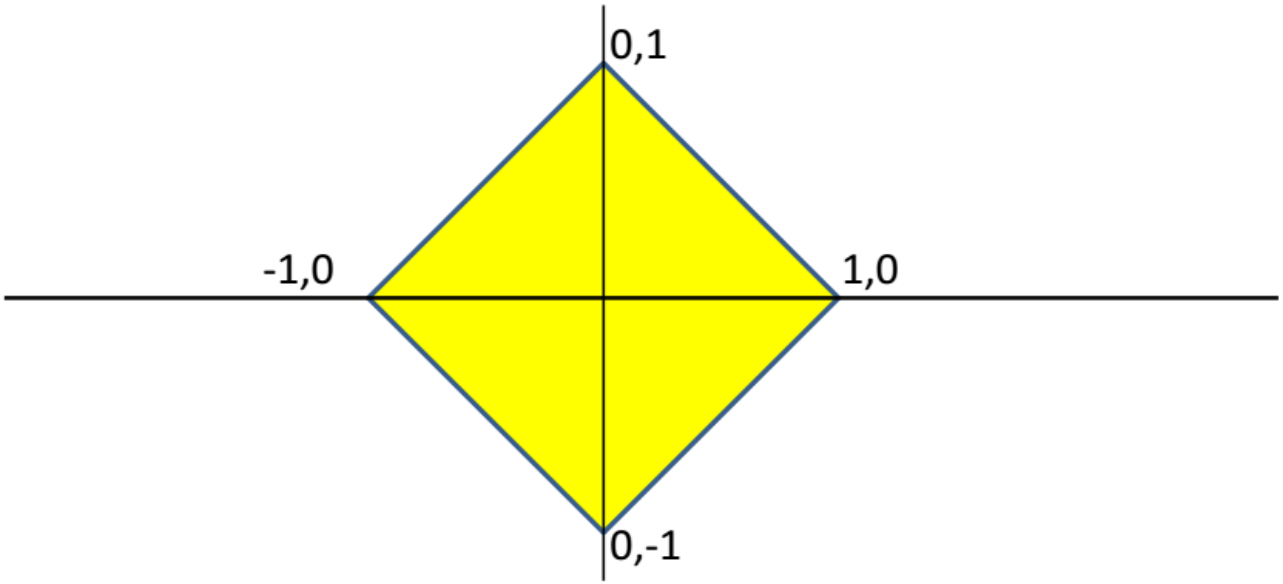
## The MLP *can* represent anything





- The MLP *can be constructed* to represent anything
- But *how* do we construct it?

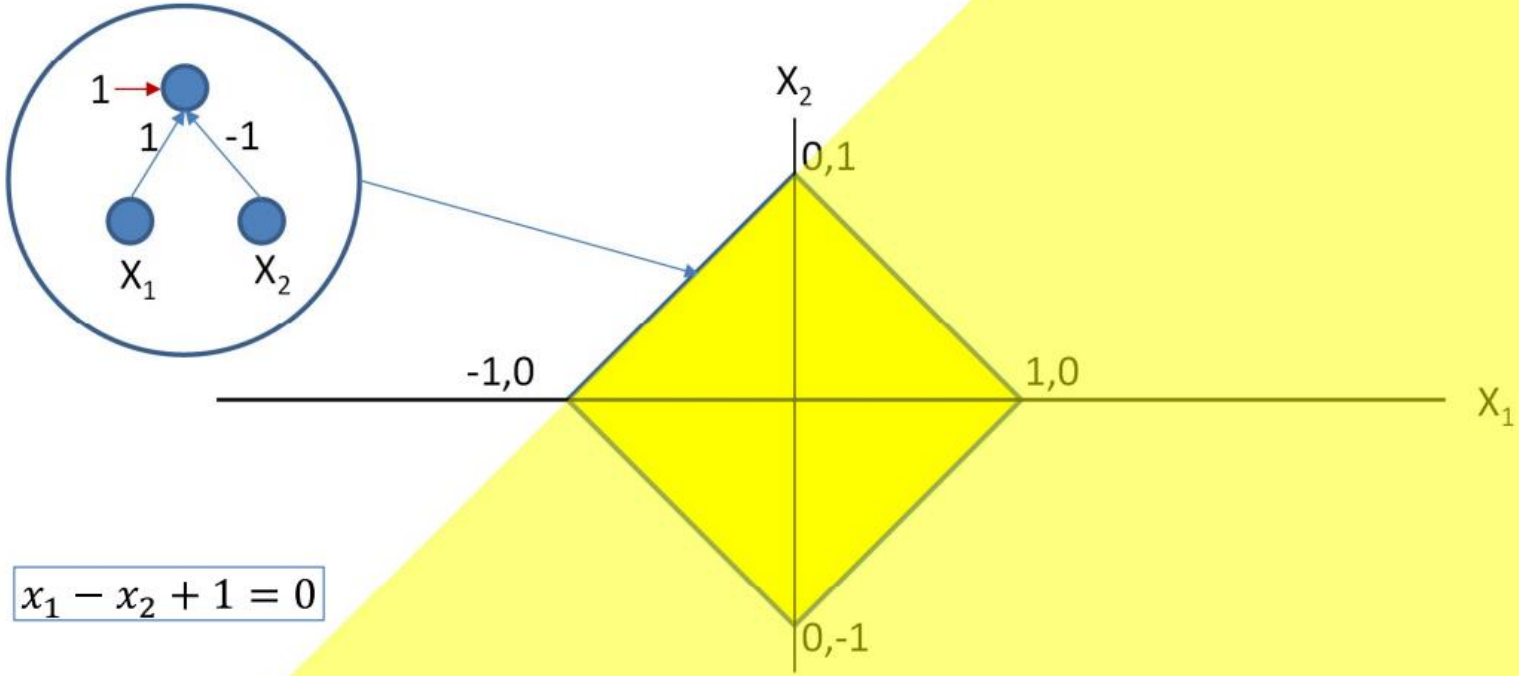
# Option 1: Construct by hand




- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary

# What we learn: The parameters of the network

## Option 1: Construct by hand

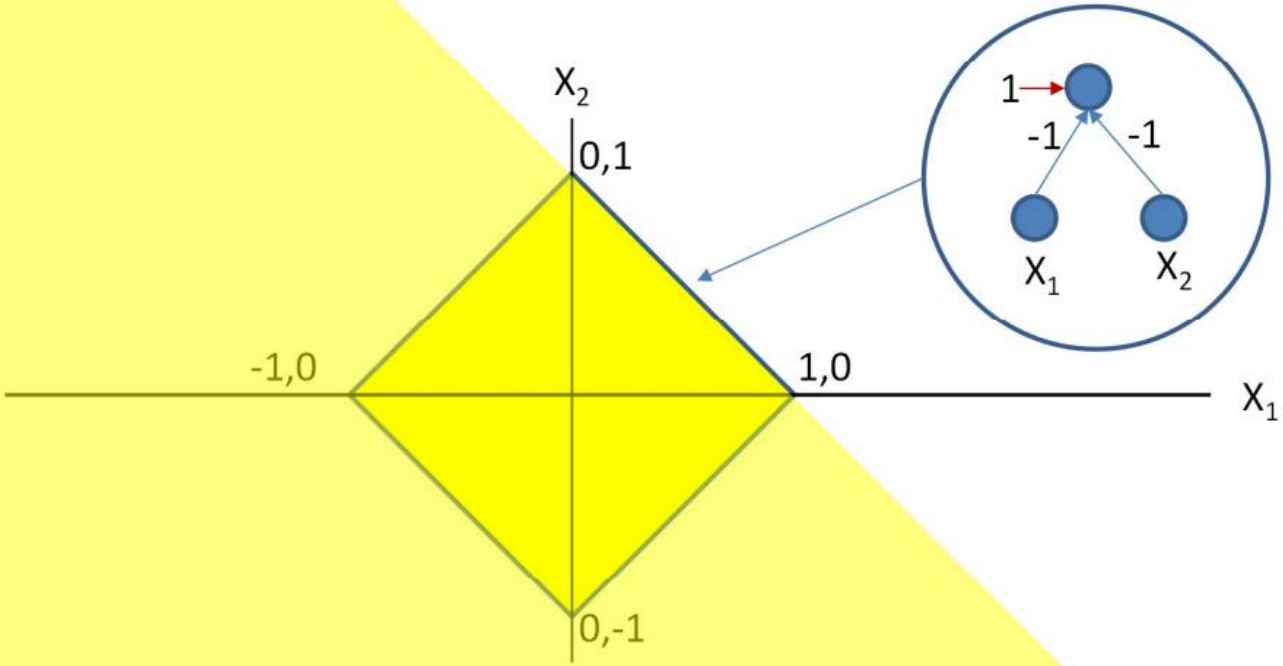



$x_1 - x_2 + 1 = 0$

Assuming simple perceptrons:
output = 1 if $\sum_i w_i x_i + b_i \geq 0, else\ 0$

# What we learn: The parameters of the network

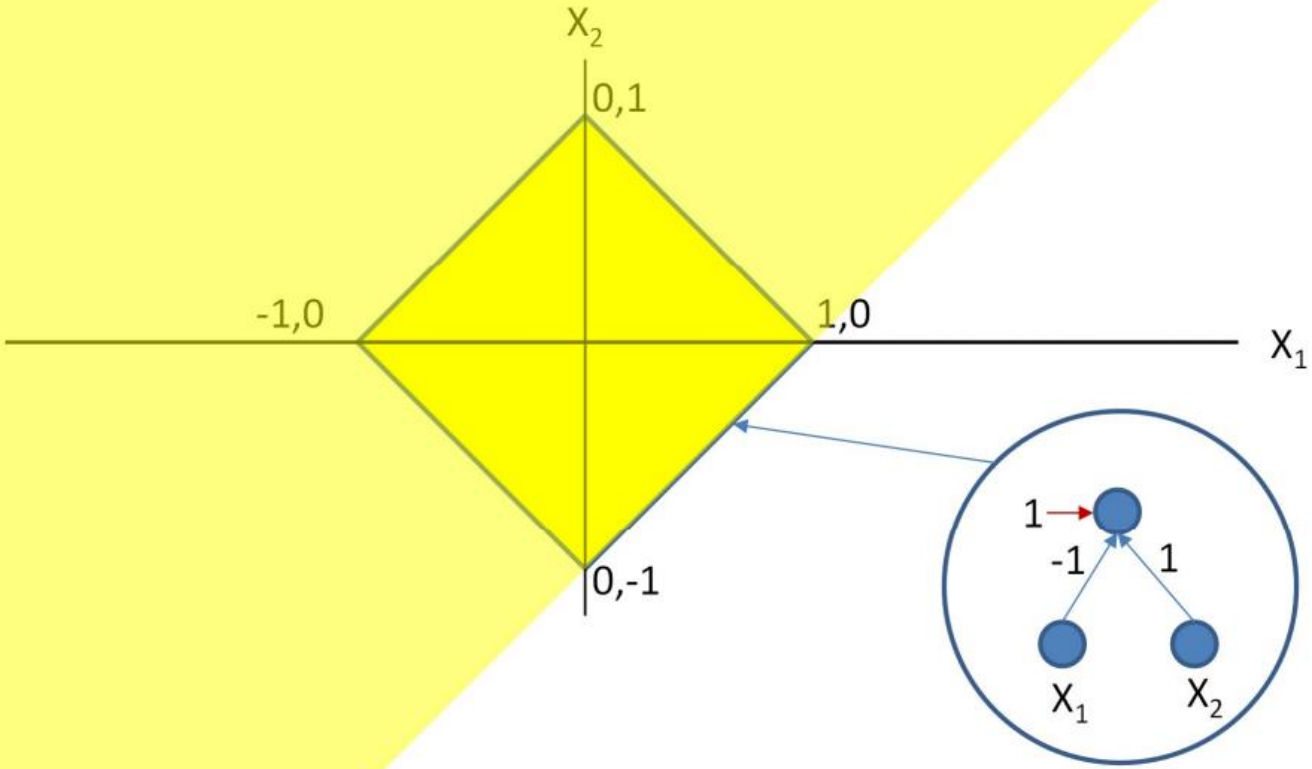
## Option 1: Construct by hand




Assuming simple perceptrons:
output = 1 if $\sum_i w_i x_i + b_i \geq 0, else\ 0$

# What we learn: The parameters of the network

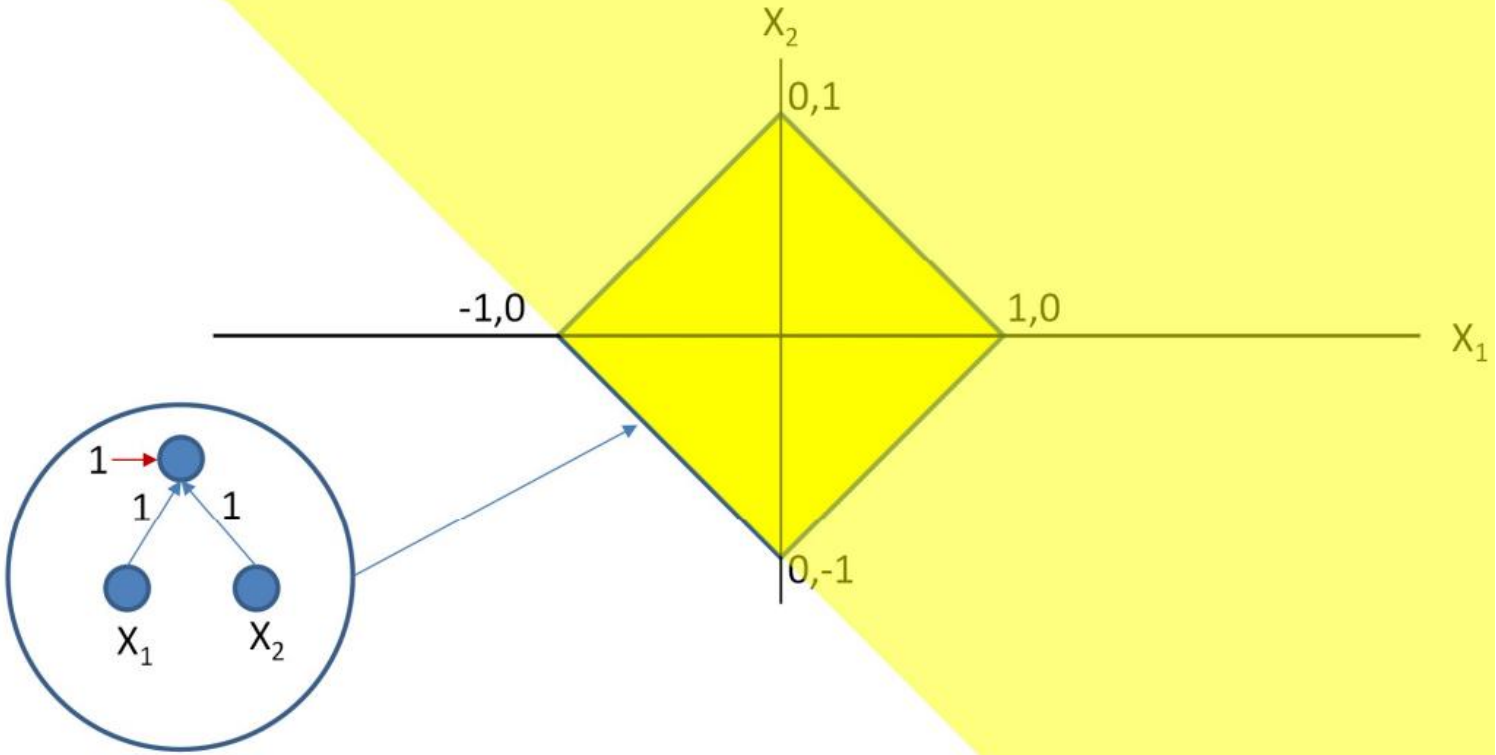
## Option 1: Construct by hand




Assuming simple perceptrons:
output = 1 if $\sum_i w_i x_i + b_i \geq 0, else\ 0$

# What we learn: The parameters of the network

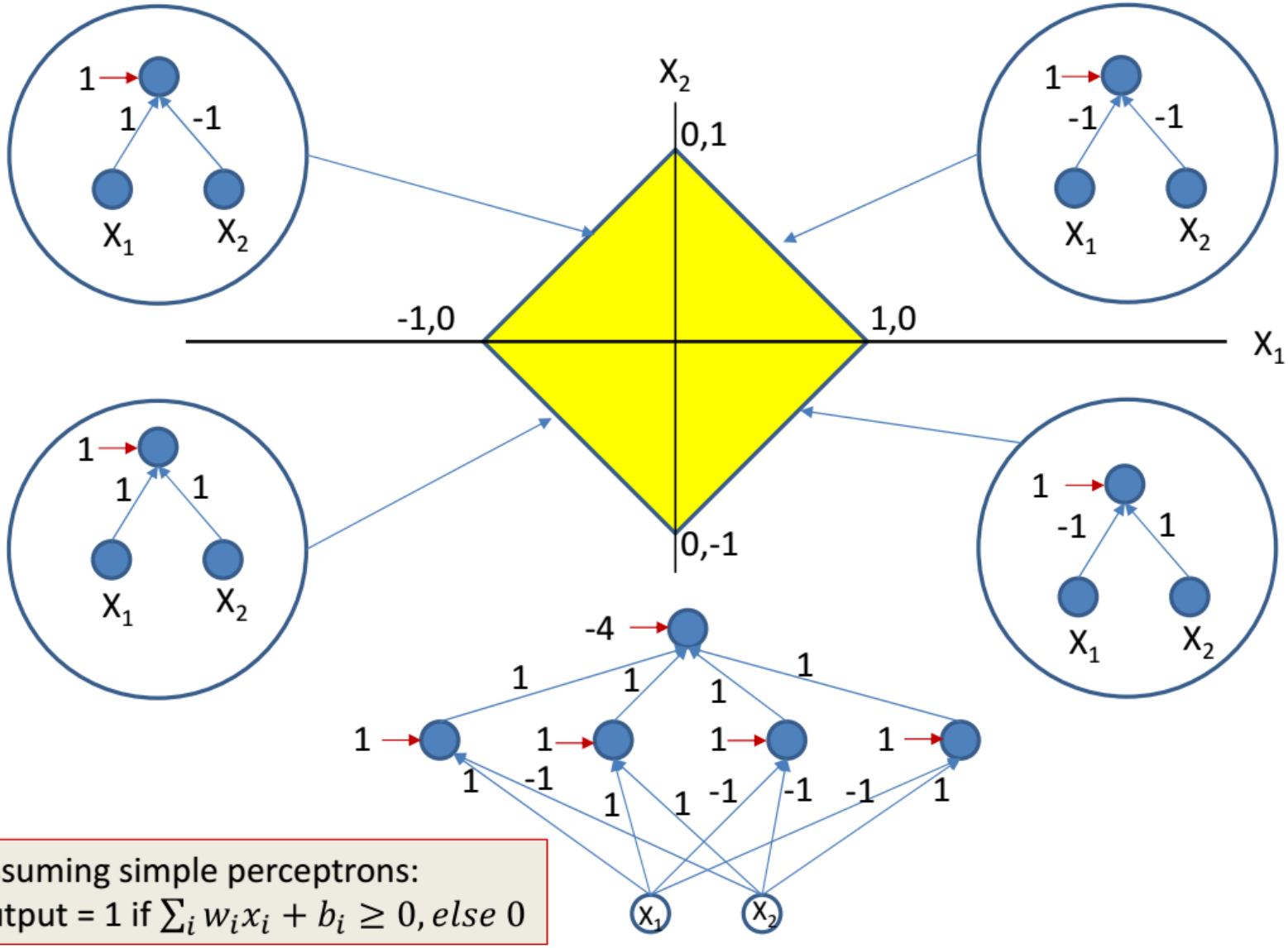
## Option 1: Construct by hand




Assuming simple perceptrons:
output = 1 if $\sum_i w_i x_i + b_i \geq 0, else\ 0$

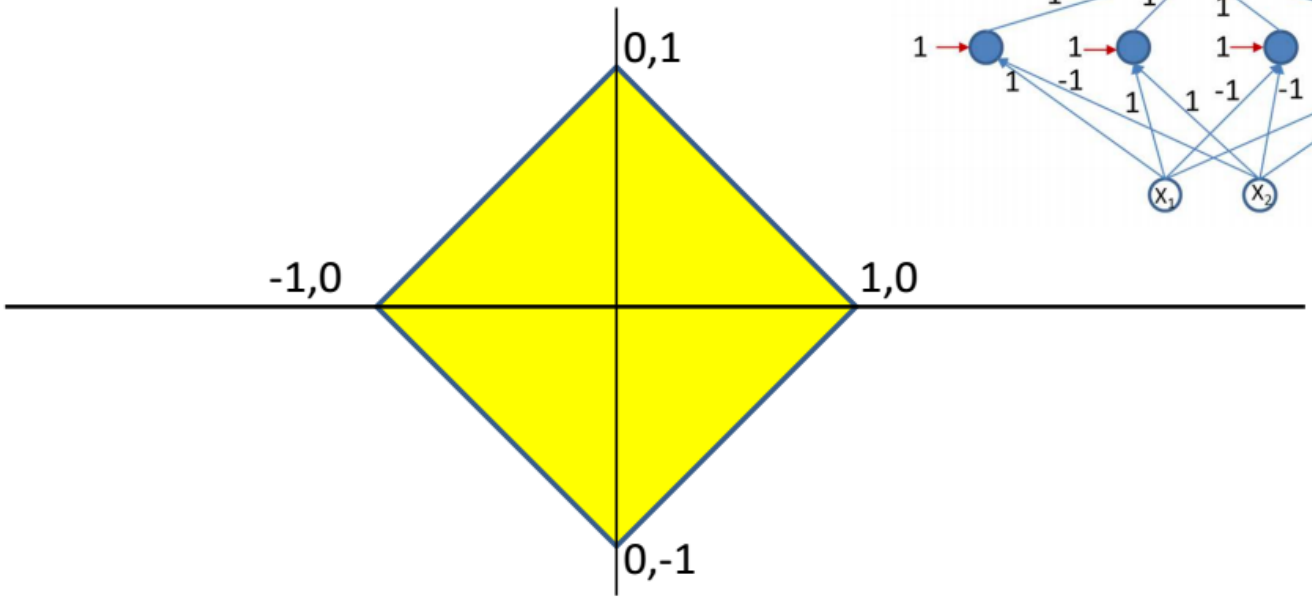
# What we learn: The parameters of the network

## Option 1: Construct by hand




Assuming simple perceptrons:
output = 1 if $\sum_i w_i x_i + b_i \geq 0, else\ 0$

# Option 1: Construct by hand



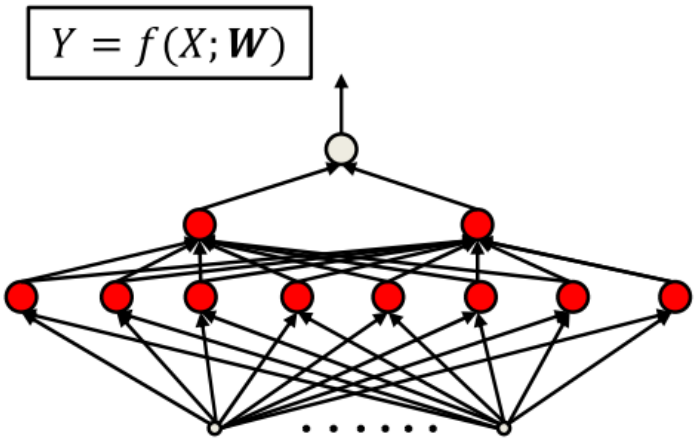


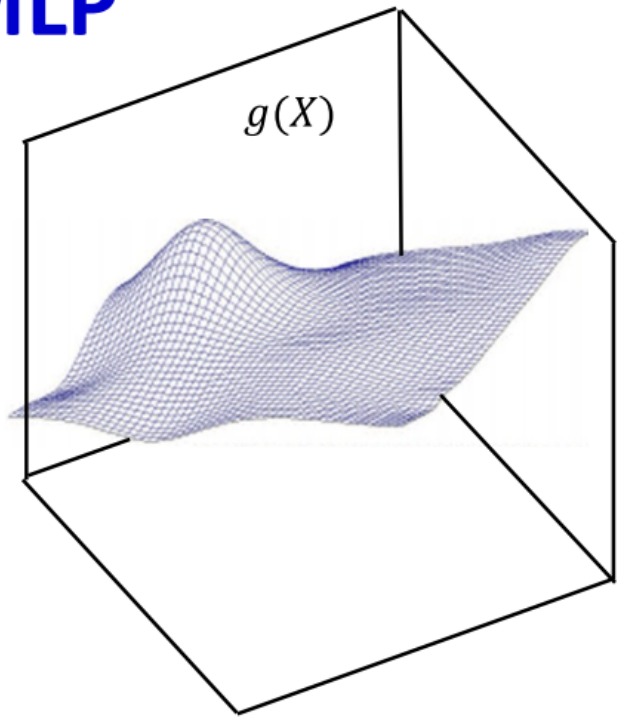


- Given a function, *handcraft* a network to satisfy it
- E.g.: Build an MLP to classify this decision boundary
- Not possible for all but the simplest problems..

# What we learn: The parameters of the network

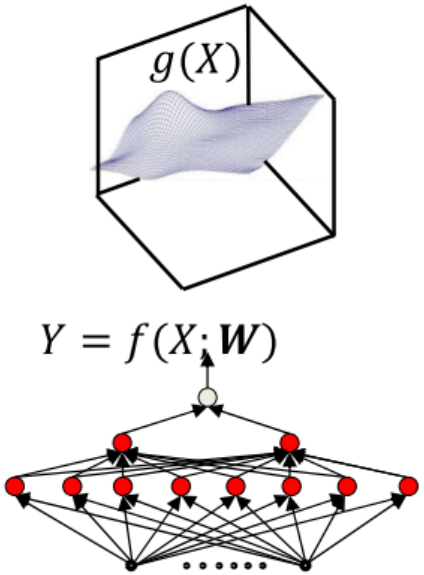
## Option 2: Automatic estimation of an MLP

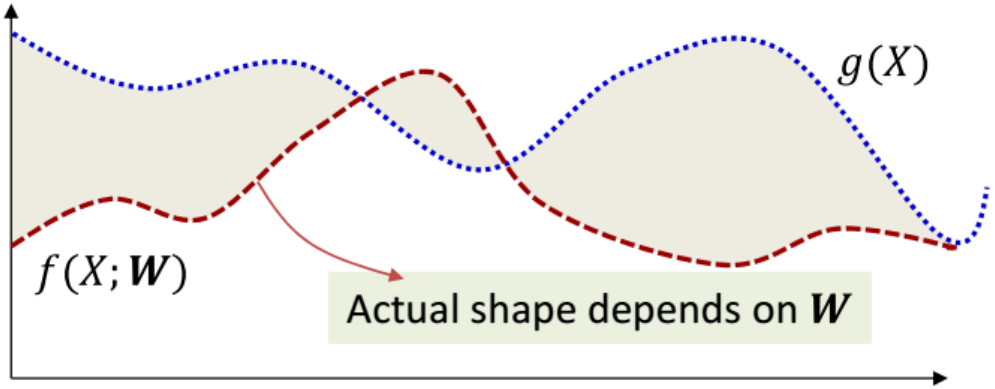



- More generally, *given* the function $g(X)$ to model, we can *derive* the parameters of the network to model it, through computation

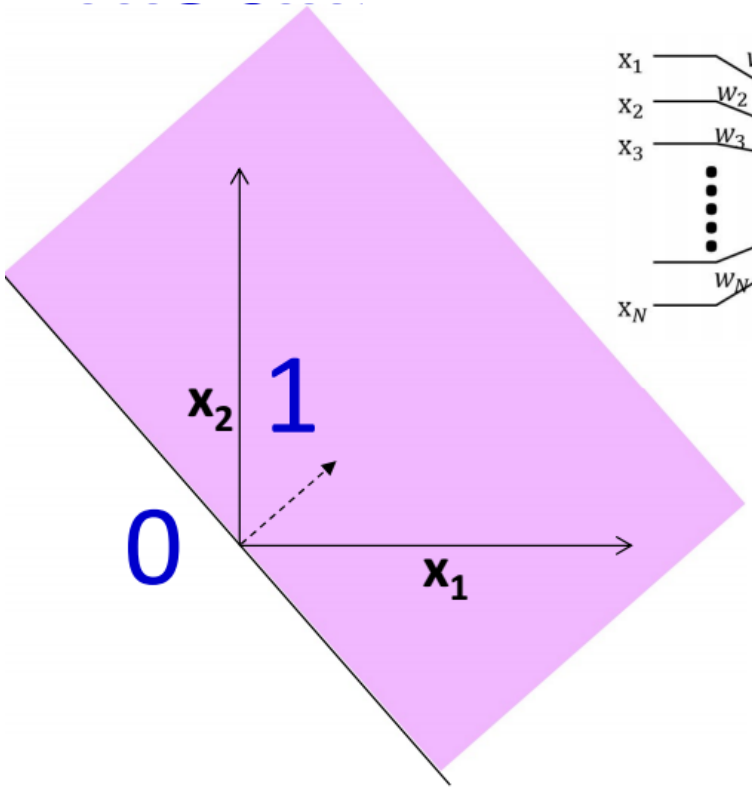
# How to learn a network?

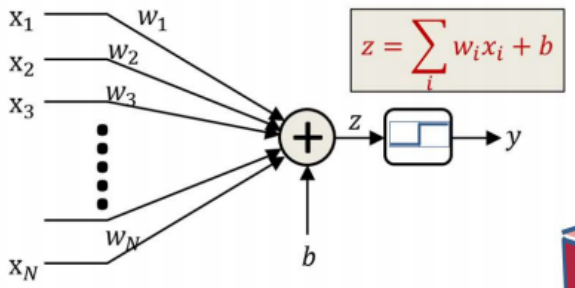


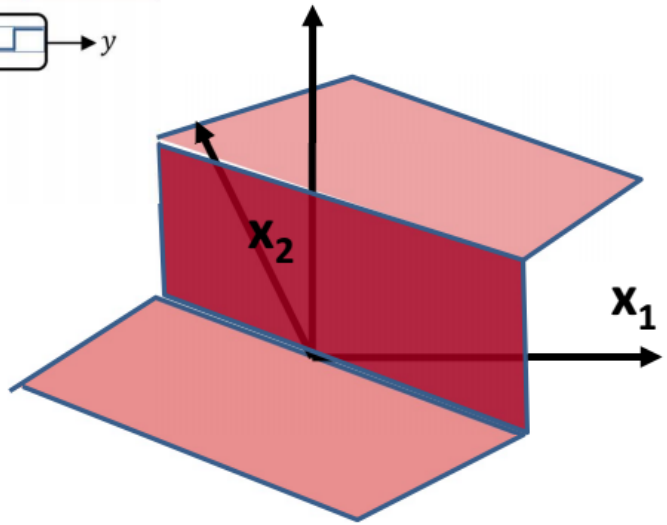




- Solution: Estimate parameters to minimize the error between the target function $g(X)$ and the network function $f(X, \boldsymbol{W})$
  - Find the parameter $\boldsymbol{W}$ that minimizes the shaded area

# The simplest MLP: a single perceptron

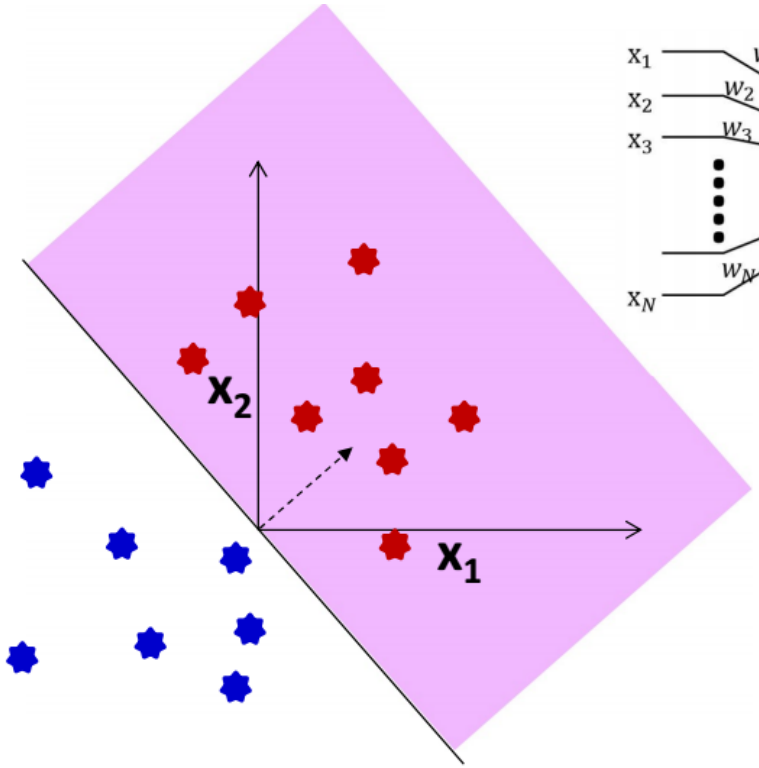


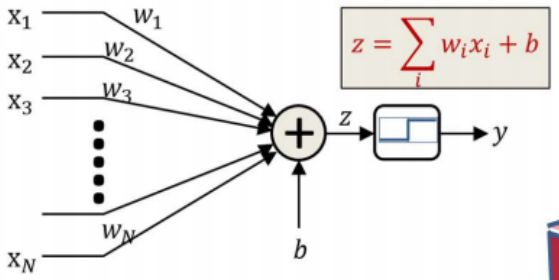



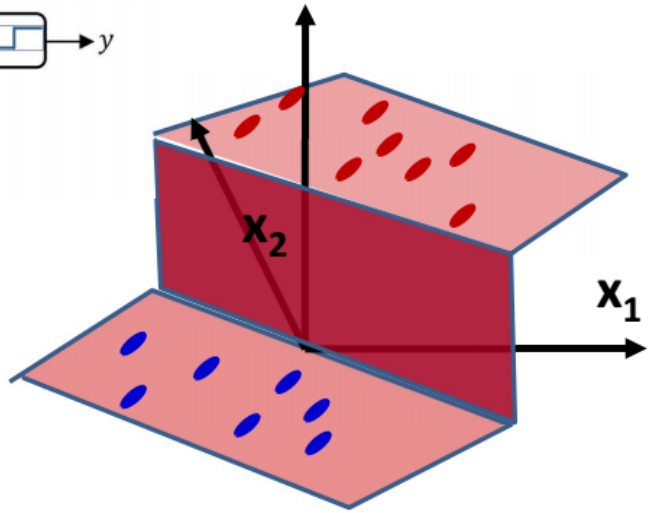




- Learn this function
  - **A step function across a hyperplane**

# The simplest MLP: a single perceptron

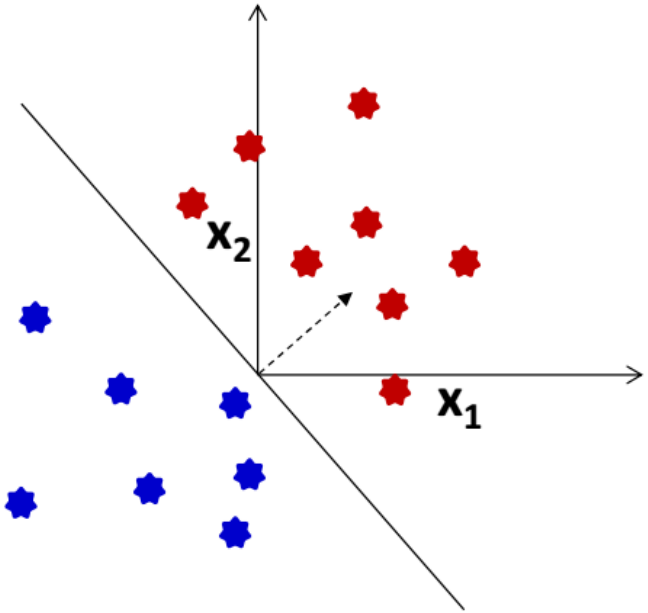


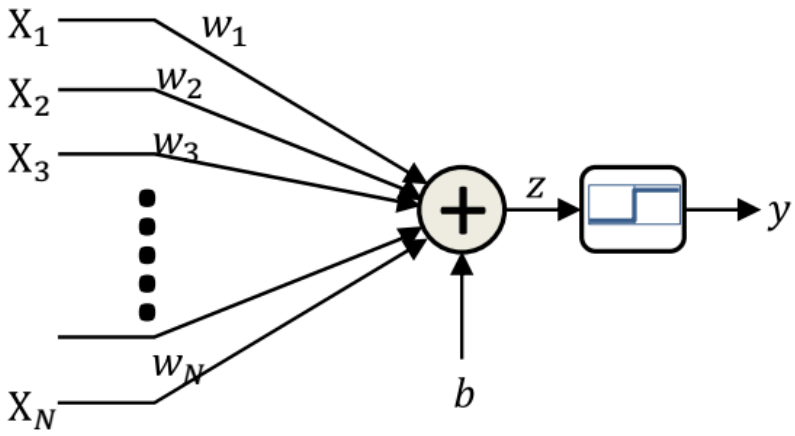




- Learn this function
  - **A step function across a hyperplane**
  - **Given only samples from it**

# Learning the perceptron

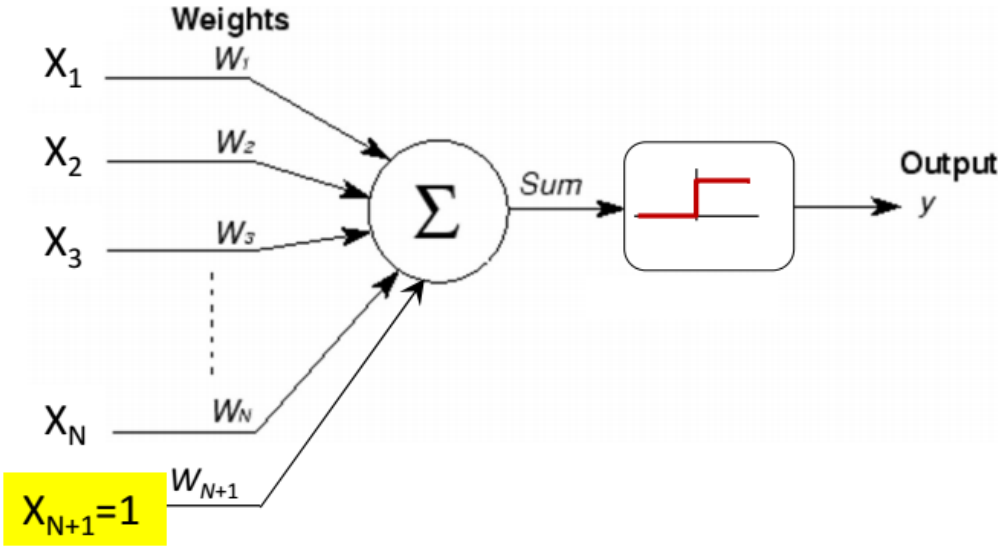



- Given a number of input output pairs, learn the weights and bias
  - $y = \begin{cases} 1 & if \ \sum_{i=1}^{N} w_i X_i + b \geq 0 \\ 0 & otherwise \end{cases}$
  - Learn $W = [w_1 .. w_N]^T$ and $b$, given several $(X, y)$ pairs
    - $X = [X_1 .. X_N]^T$

Boundary: $\sum_{i=1}^{N} w_i X_i + b = 0$

## Restating the perceptron




- Restating the perceptron equation by adding another dimension to $X$

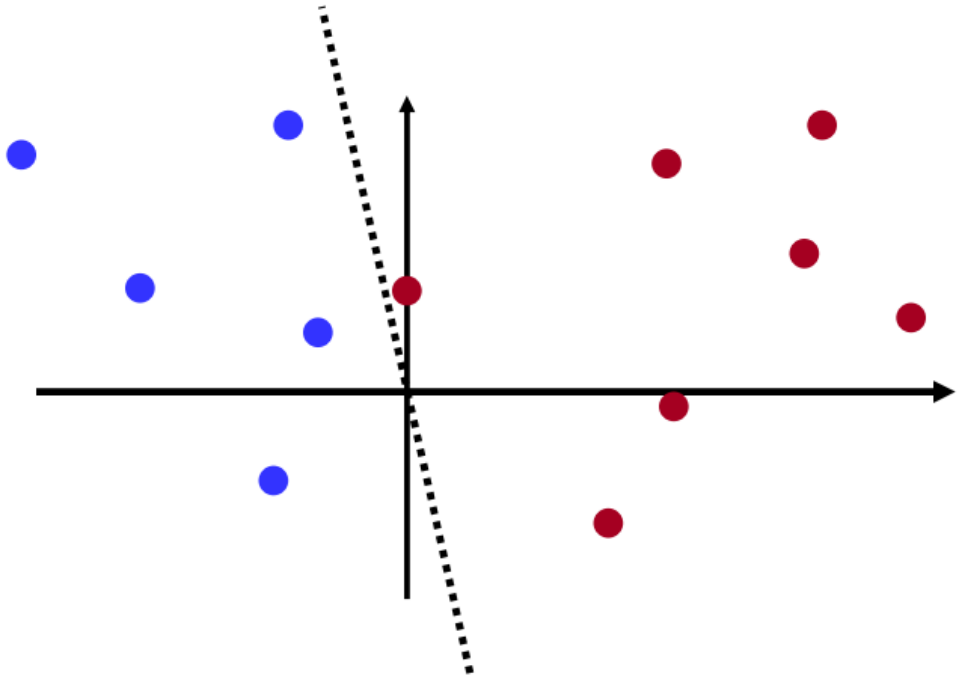
$$y = \begin{cases} 1 \;\; if \;\; \sum_{i=1}^{N+1} w_i X_i \geq 0 \\ 0 \;\; otherwise \end{cases}$$

where $X_{N+1} = 1$

- Note that the boundary $\sum_{i=1}^{N+1} w_i X_i = 0$ is now a hyperplane through origin

# Learning the perceptron

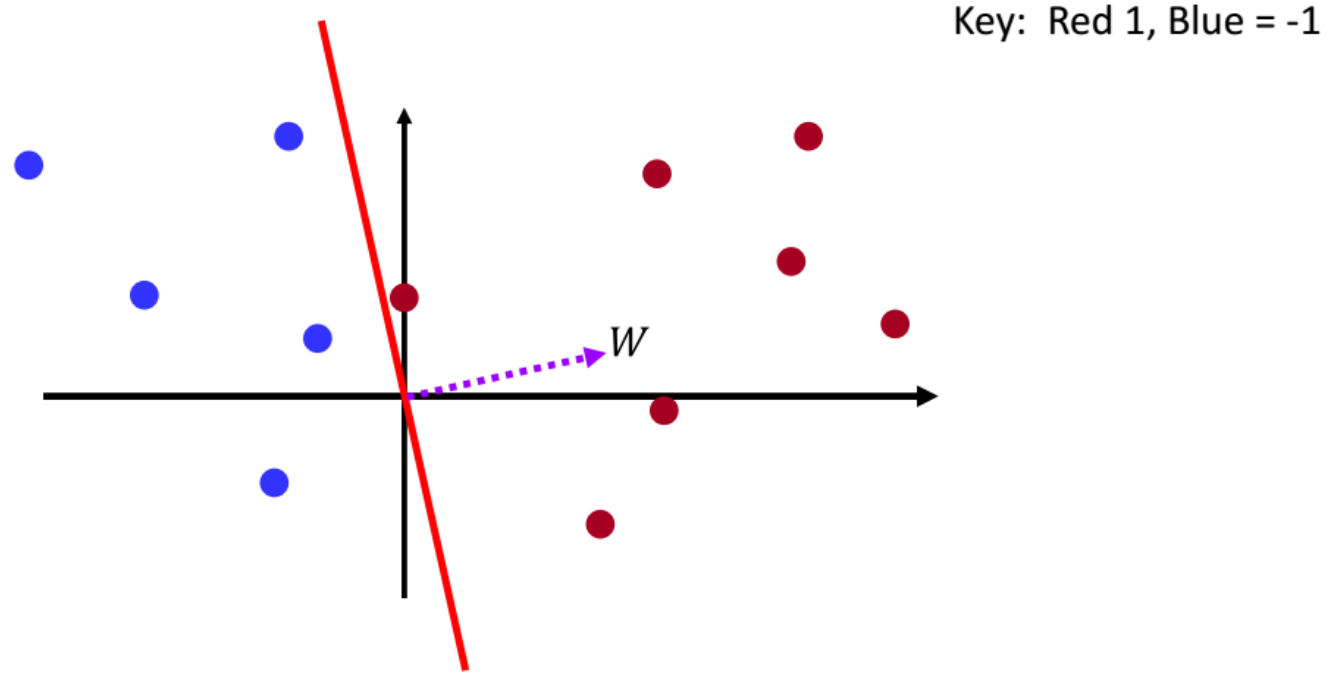
## The Perceptron Problem



- Find the hyperplane $\sum_{i=1}^{N+1} w_i X_i = 0$ that perfectly separates the two groups of points

# Learning the perceptron

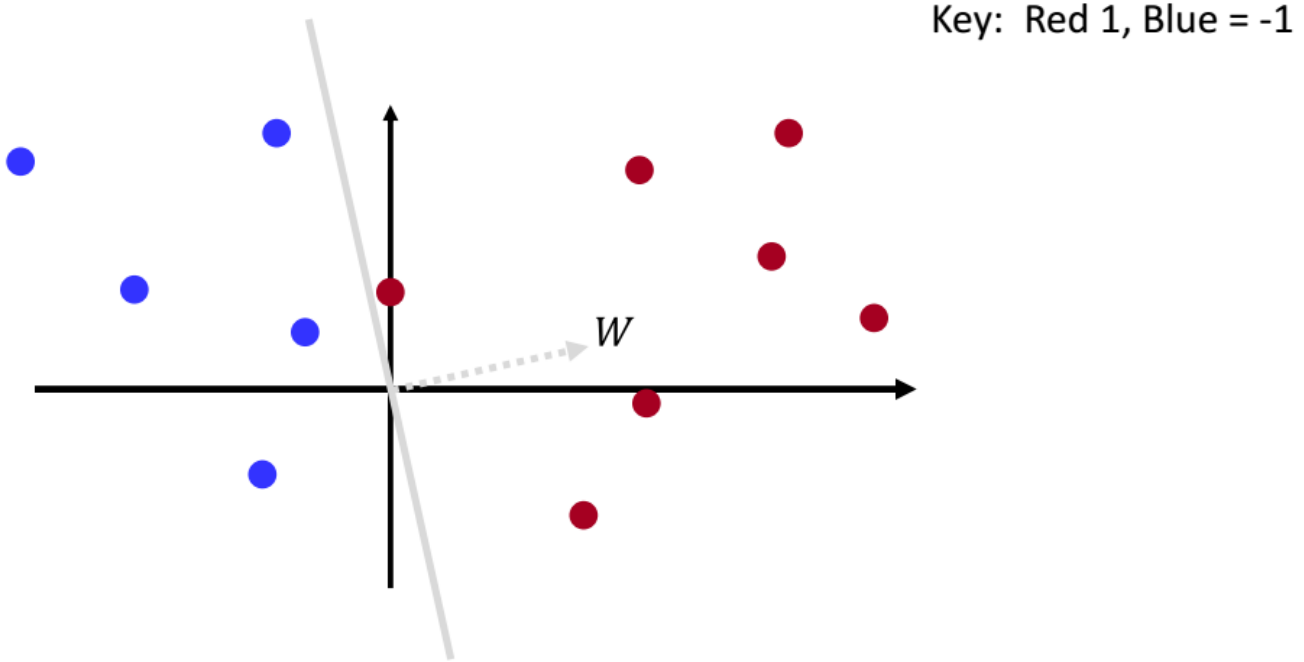
## The Perceptron Problem




- Find the hyperplane $\sum_{i=1}^{N+1} w_i X_i = 0$ that perfectly separates the two groups of points
  - Let vector $W = [w_1, w_2, \ldots, w_{N+1}]^T$ and vector $X = [x_1, x_2, \ldots, x_N, 1]^T$
  - $\sum_{i=1}^{N+1} w_i X_i = W^T X$ is an inner product
  - $W^T X = 0$ is the hyperplane comprising all $X$s orthogonal to vector $W$
    - Learning the perceptron = finding the weight vector $W$ for the separating hyperplane
    - $W$ points in the direction of the positive class

# Learning the perceptron

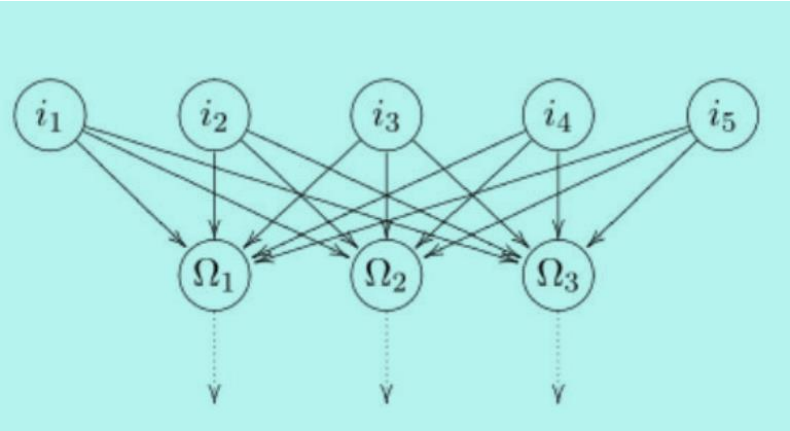
## The Perceptron Problem

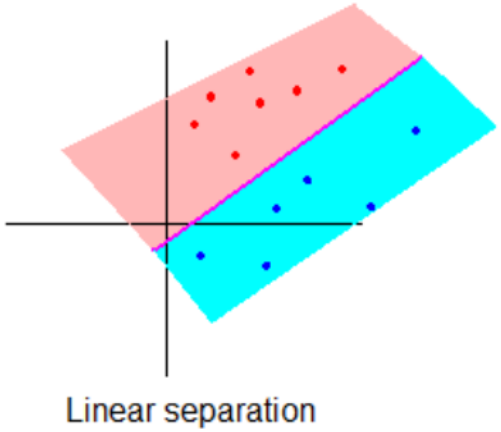



- Learning the perceptron: Find the weights vector $W$ such that the plane described by $W^T X = 0$ perfectly separates the classes
  - $W^T X$ is positive ($W^T X > 0$) for all red dots
    - The angle between $W$ and positive-class vectors is less than 90
  - $W^T X$ is negative ($W^T X < 0$) for all blue dots
    - The angle between $W$ and negative-class vectors is greater than 90

# Learning the perceptron

- Two methods:
  - The perceptron learning algorithm
  - The delta rule



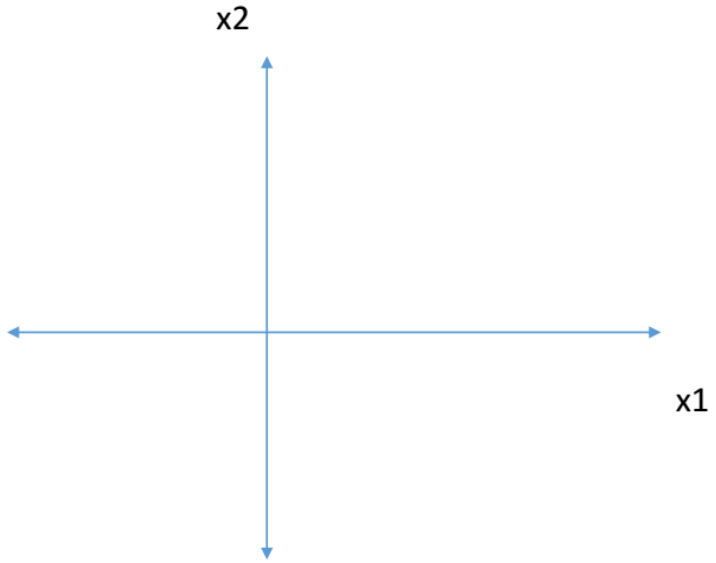



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

- Visualization example (2 features)

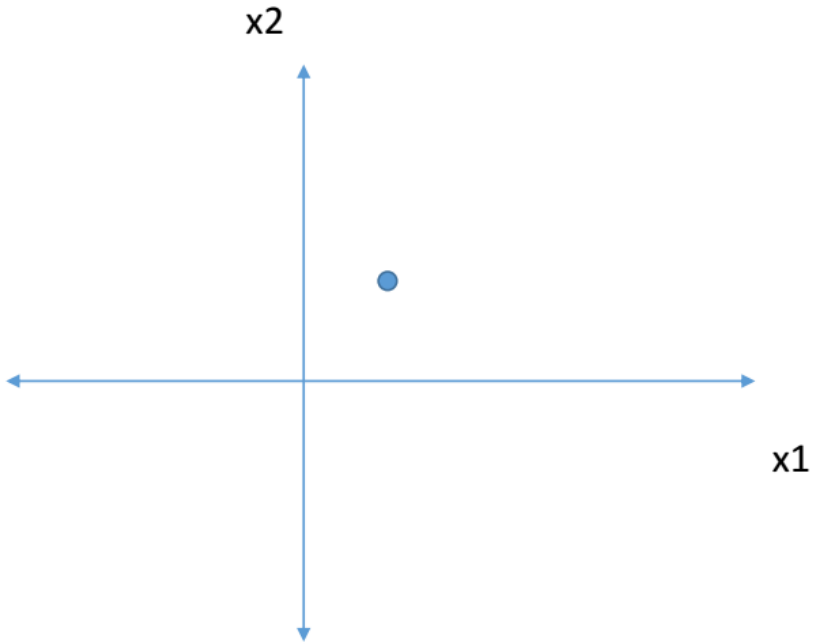
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

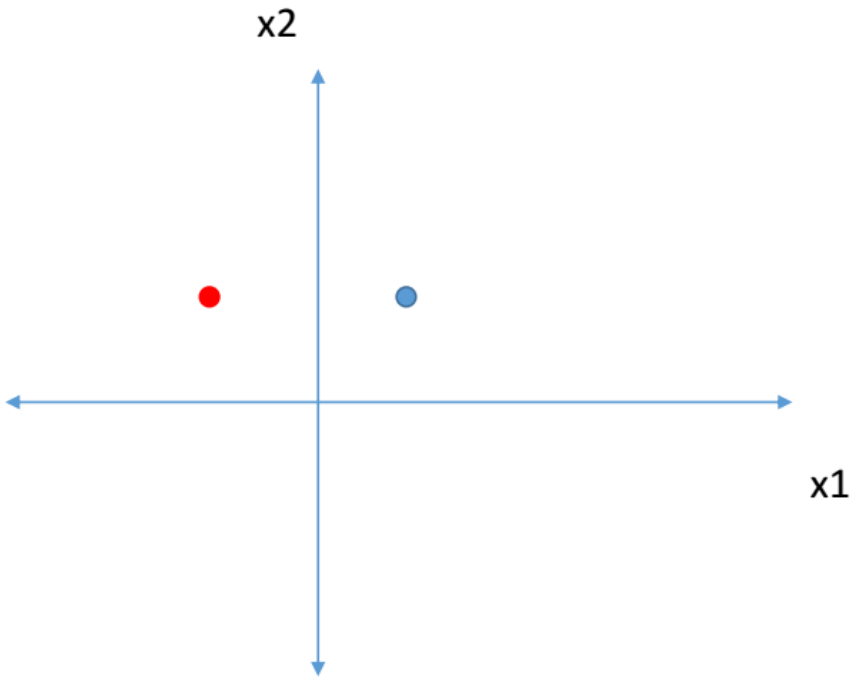
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

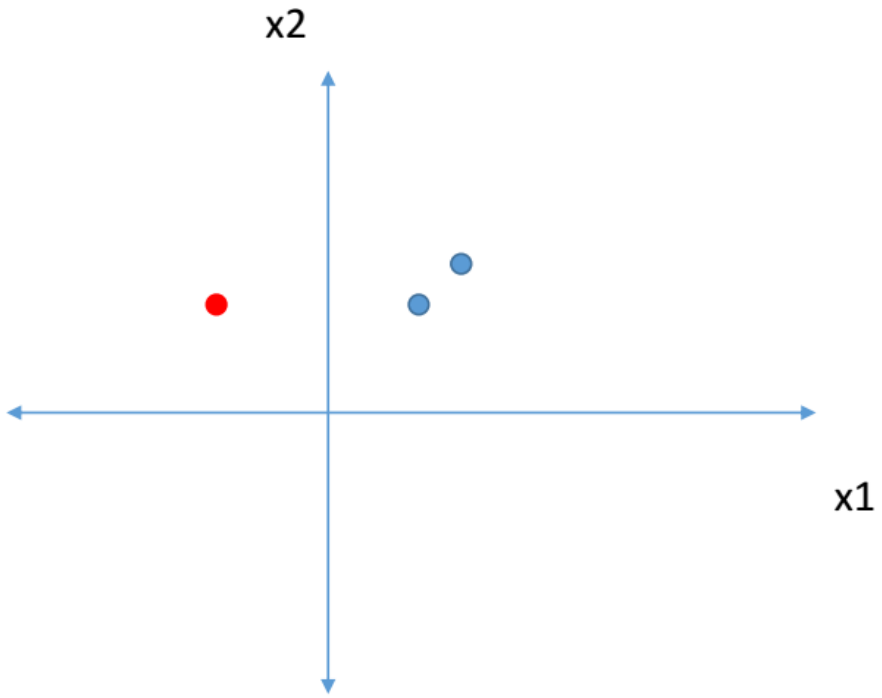
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

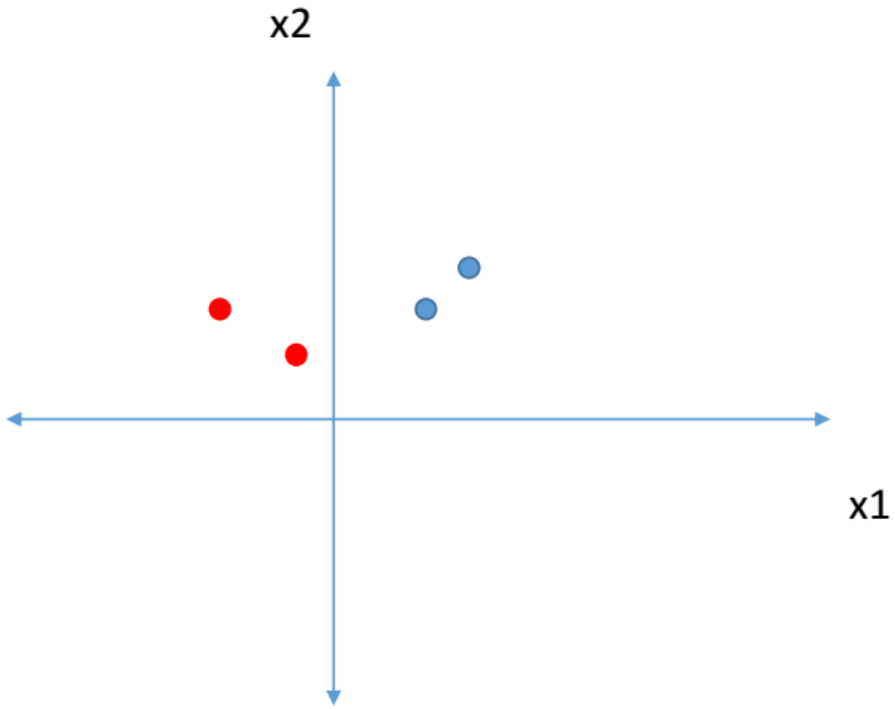
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

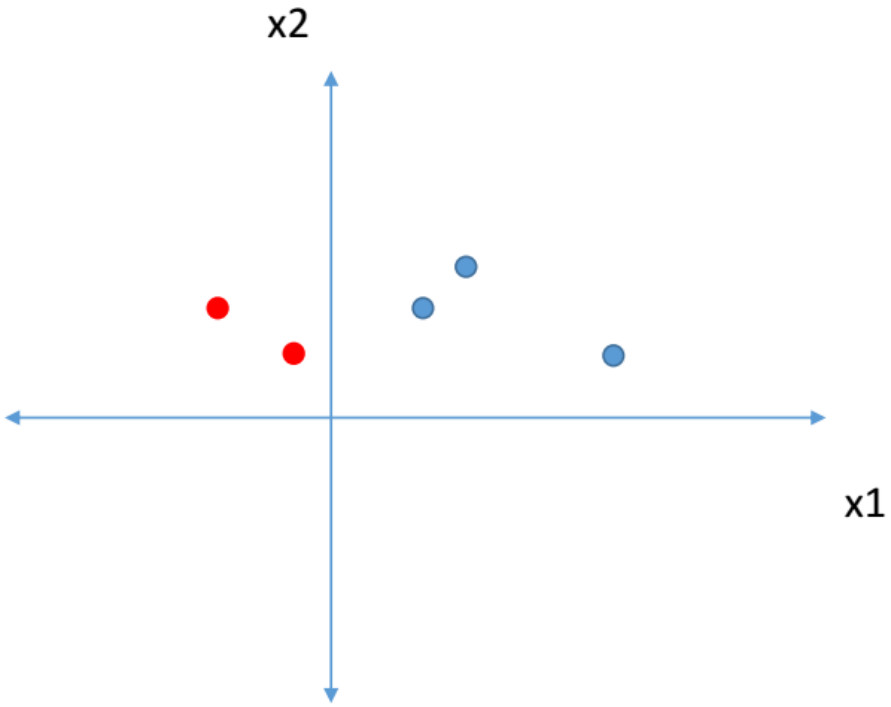
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

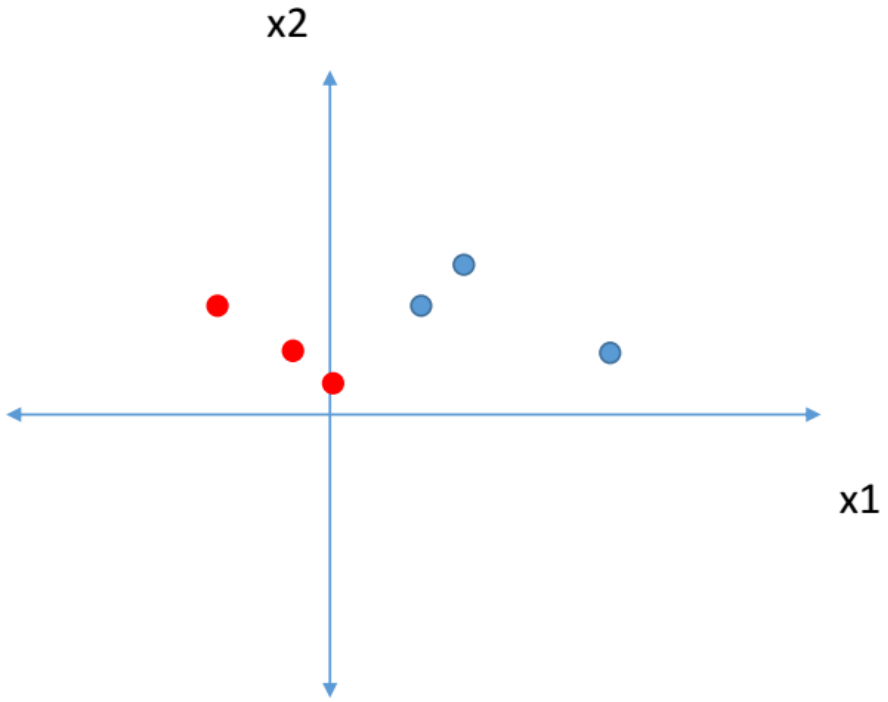
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

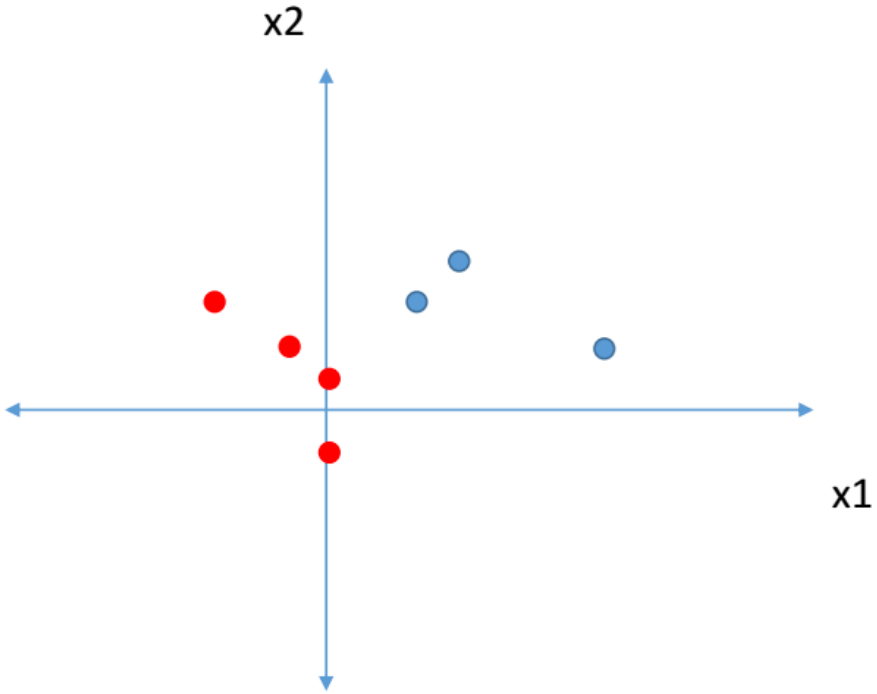
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

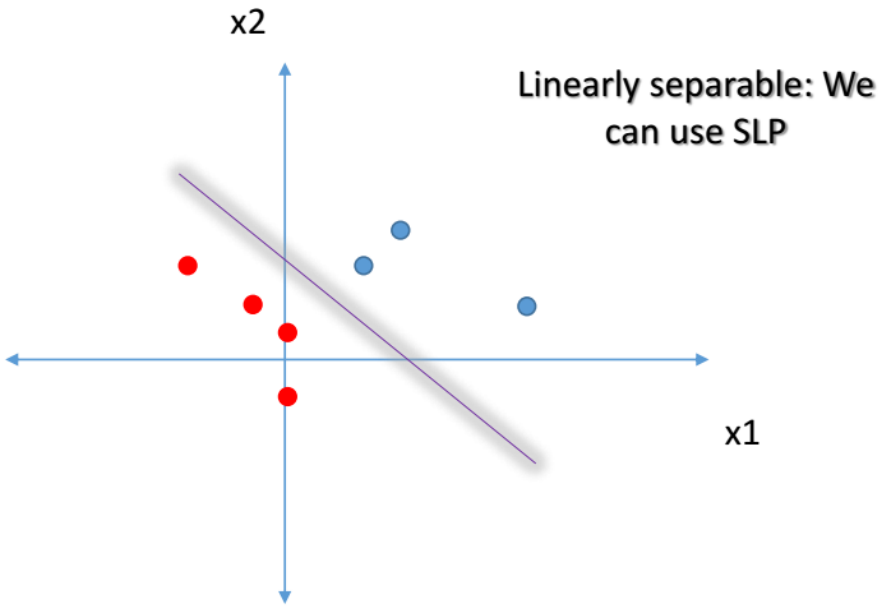
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

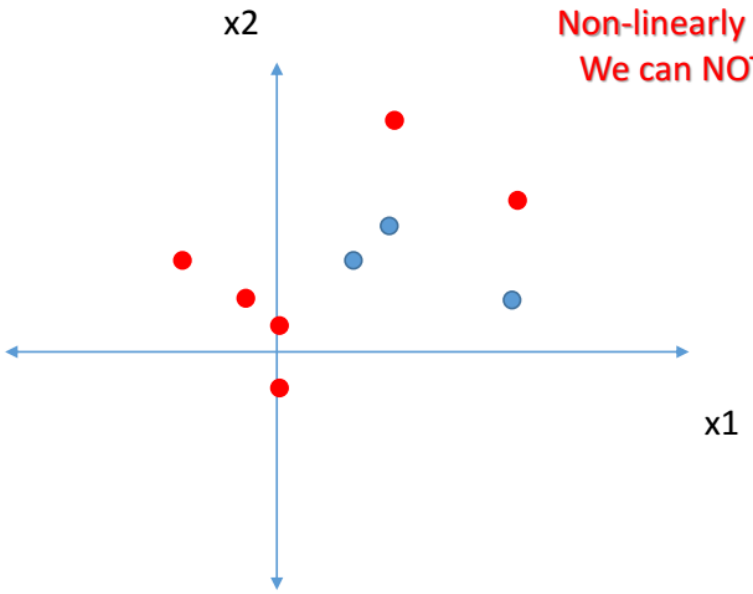
| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |



# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

| x1 | x2 | t |
|---|---|---|
| 2 | 3 | 0 |
| -3 | 3 | 1 |
| 3 | 4 | 0 |
| -1 | 2 | 1 |
| 7 | 2 | 0 |
| 0 | 1 | 1 |
| 0 | -2 | 1 |
| 3 | 8 | 1 |
| 7 | 5 | 1 |




Non-linearly separable:
We can NOT use SLP

# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

- Visualization example (1 feature)

| x | t |
|---|---|
| 1.1 | 0 |
| 2.8 | 1 |
| -0.5 | 0 |
| 1.2 | 0 |
| 4 | 1 |
| 2.6 | 0 |
| 3.4 | 1 |

Note: in 1d, SLP is a point separator


We can separate the two classes with one point: We can use SLP

# Learning the perceptron

- Before using SLP, make sure the data is linearly separable
  - Visualize the data (not possible for more than 2 features)

- Visualization example (1 feature)

Need at least 2 points:
Can't use SLP

Need at least 3 points:
Can't use SLP

# The perceptron learning algorithm

```
1: while ∃p ∈ P and error too large do
2:   Input p into the network, calculate output y {P set of training patterns}
3:   for all output neurons Ω do
4:     if y_Ω = t_Ω then
5:       Output is okay, no correction of weights
6:     else
7:       if y_Ω = 0 then
8:         for all input neurons i do
9:           w_{i,Ω} := w_{i,Ω} + o_i {...increase weight towards Ω by o_i}
10:        end for
11:      end if
12:      if y_Ω = 1 then
13:        for all input neurons i do
14:          w_{i,Ω} := w_{i,Ω} − o_i {...decrease weight towards Ω by o_i}
15:        end for
16:      end if
17:    end if
18:  end for
19: end while
```

# The perceptron learning algorithm

It assumes the output is either 0 or 1

```
1: while ∃p ∈ P and error too large do
2:    Input p into the network, calculate output y {P set of training patterns}
3:    for all output neurons Ω do
4:       if y_Ω = t_Ω then
5:          Output is okay, no correction of weights
6:       else
7:          if y_Ω = 0 then
8:             for all input neurons i do
9:                w_{i,Ω} := w_{i,Ω} + o_i {...increase weight towards Ω by o_i}
10:            end for
11:         end if
12:         if y_Ω = 1 then
13:            for all input neurons i do
14:               w_{i,Ω} := w_{i,Ω} − o_i {...decrease weight towards Ω by o_i}
15:            end for
16:         end if
17:      end if
18:   end for
19: end while
```

# The perceptron learning algorithm

- Example:

| x1 | x2 | t |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | | | | | | | 0 |
| 0 | 1 | | | | | | | 1 |
| 1 | 0 | | | | | | | 1 |
| 1 | 1 | | | | | | | 1 |

Add new columns

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | | | | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Bias node always produces 1

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Put initial weights (given)

If not given: assume random weights
(but not 0)

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net = x1*w1 + x2*w2 + bias*w_bias

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate y =

1 if net >= threshold,

0 if net < threshold

Threshold should be given. If not, assume random threshold

Here we assume threshold = 0.1 → net < threshold

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

y = t ? yes

Weights will not be changed

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Use the same weights for next pattern

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net = x1*w1 + x2*w2 + bias*w_bias

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

net < 0.1 → y = 0

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

y != t

We need to change weights

y = 0 we want y = 1 increase weights

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

w1 := w1 + x1
w2 := w1 + x1
w_bias := w_bias + bias

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net = x1*w1 + x2*w2 + bias*w_bias

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | | | | | | 1 |

net >= 0.1 → y = 1

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 0.1 | 1.2 | 0.8 | | | 1 |

y = t

Don't change weights

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 0.1 | 1.2 | 0.8 | 2.1 | | 1 |

Calculate net = x1*w1 + x2*w2 + bias*w_bias

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 0.1 | 1.2 | 0.8 | 2.1 | 1 | 1 |

net >= 0.1 → y = 1

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 0.1 | 1.2 | 0.8 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 0.1 | 1.2 | 0.8 | | | |

y = t

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 0.1 | 1.2 | 0.8 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 0.1 | 1.2 | 0.8 | | | |

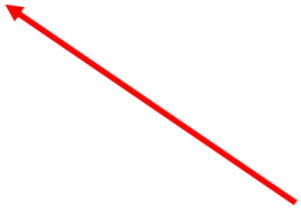
1 Epoch complete:

But we still have 1 error

We need to run another epoch

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 0.1 | 1.2 | 0.8 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 0.1 | 1.2 | 0.8 | | | |



Use these as initial weights for next epoch

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

New epoch with initial weights from previous slide

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

y != t  y = 1 and we want y = 0

Decrease weights:

w1  := w1 - x1
w2  := w1 - x1
w_bias := w_bias - bias

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | -0.2 | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

y = t

Don't change weights

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | -0.2 | -0.1 | 0 | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | -0.2 | -0.1 | 0 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | 0.8 | | | 1 |

y = 0 and we want y = 1

Increase weights:

w1 := w1 + x1
w2 := w1 +x1
w_bias := w_bias + bias

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | -0.2 | -0.1 | 0 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | 0.8 | 3.1 | 1 | 1 |

Calculate net and y

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | -0.2 | -0.1 | 0 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | 0.8 | 3.1 | 1 | 1 |
| Weights for next epoch | | | 1.1 | 1.2 | 0.8 | | | |

y = t

Don't change weights

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 0.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0.1 | 1.2 | -0.2 | -0.1 | 0 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | 0.8 | 3.1 | 1 | 1 |
| Weights for next epoch | | | 1.1 | 1.2 | 0.8 | | | |

Second epoch done

We still have 2 errors

We need to run another epoch

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Starting third epoch

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Decrease weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | | | 1 |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | 1 | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 1.1 | 1.2 | -0.2 | | | |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | 0.8 | 0.8 | 1 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 1.1 | 1.2 | -0.2 | | | |

We still have one error

We need to run another epoch

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Fourth epoch

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | | | 1 |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | 1 | 1 |

Calculate net and y

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 1.1 | 1.2 | -0.2 | | | |

Don't change weights

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | **0** | **0** |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | **1** | **1** |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | **1** | **1** |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | **1** | **1** |
| Weights for next epoch | | | 1.1 | 1.2 | -0.2 | | | |

Fourth epoch done

No errors → Stop training

# The perceptron learning algorithm

| x1 | x2 | bias | w1 | w2 | w_bias | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1.1 | 1.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 1.1 | 1.2 | -0.2 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1.1 | 1.2 | -0.2 | 0.9 | 1 | 1 |
| 1 | 1 | 1 | 1.1 | 1.2 | -0.2 | 2.1 | 1 | 1 |
| Weights for next epoch | | | 1.1 | 1.2 | -0.2 | | | |

# The delta rule

- Same as the previous example. Just updating weights is different

$$w_{i,\Omega} := w_{i,\Omega} + \eta\, o_i\, (t_\Omega - y_\Omega)$$

- For previous example:
  - w1 := w1 + η * x1 * (t – y)
  - w2 := w2 + η * x2 * (t – y)
  - w_bias := w_bias + η * bias * (t – y)

# The delta rule

- Same as the previous example. Just updating weights is different

$$w_{i,\Omega} := w_{i,\Omega} + \eta\, o_i\, (t_\Omega - y_\Omega)$$

- For previous example:
  - w1 := w1 + η * x1 * (t – y)
  - w2 := w2 + η * x2 * (t – y)
  - w_bias := w_bias + η * bias * (t – y)

The term “bias”
always equals 1
(can be omitted)

# The delta rule

- Same as the previous example. Just updating weights is different

$$w_{i,\Omega} := w_{i,\Omega} + \eta\, o_i\, (t_\Omega - y_\Omega)$$

- For previous example:
  - w1 := w1 + η * x1 * (t – y)
  - w2 := w2 + η * x2 * (t – y)
  - w_bias := w_bias + η * bias * (t – y)

This is the learning rate (a given constant). If not given, assume a value between 0.01 and 0.9

# The delta rule

- Same as the previous example. Just updating weights is different

$$w_{i,\Omega} := w_{i,\Omega} + \eta\, o_i\, (t_\Omega - y_\Omega)$$

- For previous example:
  - w1 := w1 + η * x1 * (t – y)
  - w2 := w2 + η * x2 * (t – y)
  - w_bias := w_bias + η * bias * (t – y)

We always add
(even if y > t)

But how do we
decrease weights?

# The delta rule

- Same as the previous example. Just updating weights is different

$$w_{i,\Omega} := w_{i,\Omega} + \eta\, o_i\, (t_\Omega - y_\Omega)$$

- For previous example:
  - w1 := w1 + η * x1 * (t – y)
  - w2 := w2 + η * x2 * (t – y)
  - w_bias := w_bias + η * bias * (t – y)

If y > t, this term will be negative, causing weights to be decreased

# The delta rule

- The same example using delta rule:

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Assume learning rate = 0.1

# The delta rule

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculating net and y is not different

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

If we try to update weights: (even though y = t)

w1 := w1 + 0.1 * x1 * (t - y)
w2 := w2 + 0.1 * x2 * (t - y)
wb := wb + 0.1 * bias * (t - y)

(t - y) = 0 so the weights will not be changed

# The delta rule

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate y and net

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 0.3 | -0.1 | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

update weights:

w1 := w1 + 0.1 * x1 * (t - y) → w1 := 0.1 + 0.1 * 0 * 1 → 0.1
w2 := w2 + 0.1 * x2 * (t - y) → w2 := 0.2 + 0.1 * 1 * 1 → 0.3
wb := wb + 0.1 * bias * (t - y) → wb := -0.2 + 0.1 * 1 * 1 → -0.1

# The delta rule

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 0.3 | -0.1 | 0 | 0 | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Calculate net and y

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 0.3 | -0.1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0.2 | 0.3 | 0 | | | 1 |

update weights:

w1 := w1 + 0.1 * x1 * (t - y) → 0.1 + 0.1 * 1 * 1 → 0.2
w2 := w2 + 0.1 * x2 * (t - y) → 0.3 + 0.1 * 0 * 1 → 0.3
wb := wb + 0.1 * bias * (t - y) → -0.1 + 0.1 * 1 * 1 → 0

# The delta rule

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 0.3 | -0.1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0.2 | 0.3 | 0 | 0.5 | 1 | 1 |

Calculate net and y

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 0.3 | -0.1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0.2 | 0.3 | 0 | 0.5 | 1 | 1 |
| Weights for next epoch: | | | 0.2 | 0.3 | 0 | | | |

Weights will not be changed

# The delta rule

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.1 | 0.2 | -0.2 | -0.2 | 0 | 0 |
| 0 | 1 | 1 | 0.1 | 0.2 | -0.2 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0.1 | 0.3 | -0.1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0.2 | 0.3 | 0 | 0.5 | 1 | 1 |
| Weights for next epoch: | | | 0.2 | 0.3 | 0 | | | |

First epoch done

We have 2 errors

We need to run another epoch

# The delta rule

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.2 | 0.3 | 0 | | | 0 |
| 0 | 1 | 1 | | | | | | 1 |
| 1 | 0 | 1 | | | | | | 1 |
| 1 | 1 | 1 | | | | | | 1 |

Second epoch

| x1 | x2 | bias | w1 | w2 | wb | net | y | t |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0.2 | 0.3 | 0 | 0 | **0** | **0** |
| 0 | 1 | 1 | 0.2 | 0.3 | 0 | 0.3 | **1** | **1** |
| 1 | 0 | 1 | 0.2 | 0.3 | 0 | 0.2 | **1** | **1** |
| 1 | 1 | 1 | 0.2 | 0.3 | 0 | 0.5 | **1** | **1** |
| Weights for next epoch: | | | 0.2 | 0.3 | 0 | | | |

Second epoch has no errors → stop training

# Learning the perceptron

- Next:
  - SLP with multiple outputs