UNIVERSITY of
HOUSTON
DEPARTMENT OF COMPUTER SCIENCE

Perceptron

Rakesh Verma

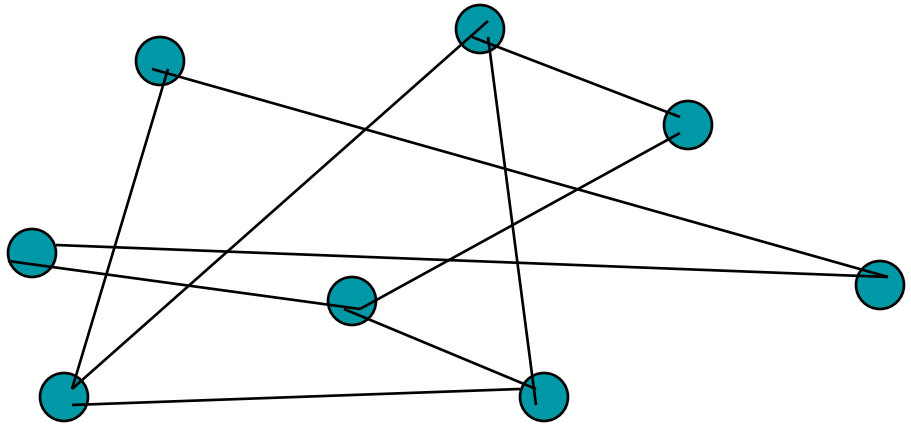
Perceptron as one Type of Linear Discriminants

- Introduction
- Design of Primitive Units
  - ✓ Perceptrons
- Security applications of perceptron

Introduction

- Artificial Neural Networks are crude attempts to model the highly massive parallel and distributed processing we believe takes place in the brain.

- Consider:

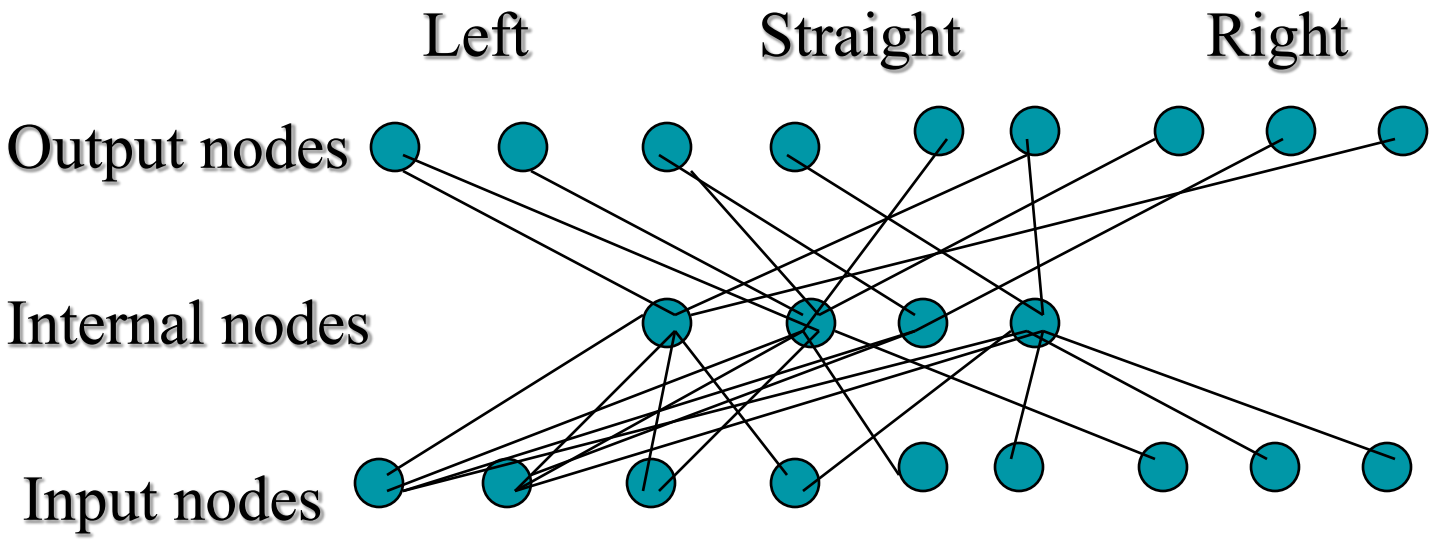


✓ the speed at which the brain recognizes images;

✓ the many neurons populating a brain;

✓ the speed at which a single neuron transmits signals.

Introduction Neural Network Representation



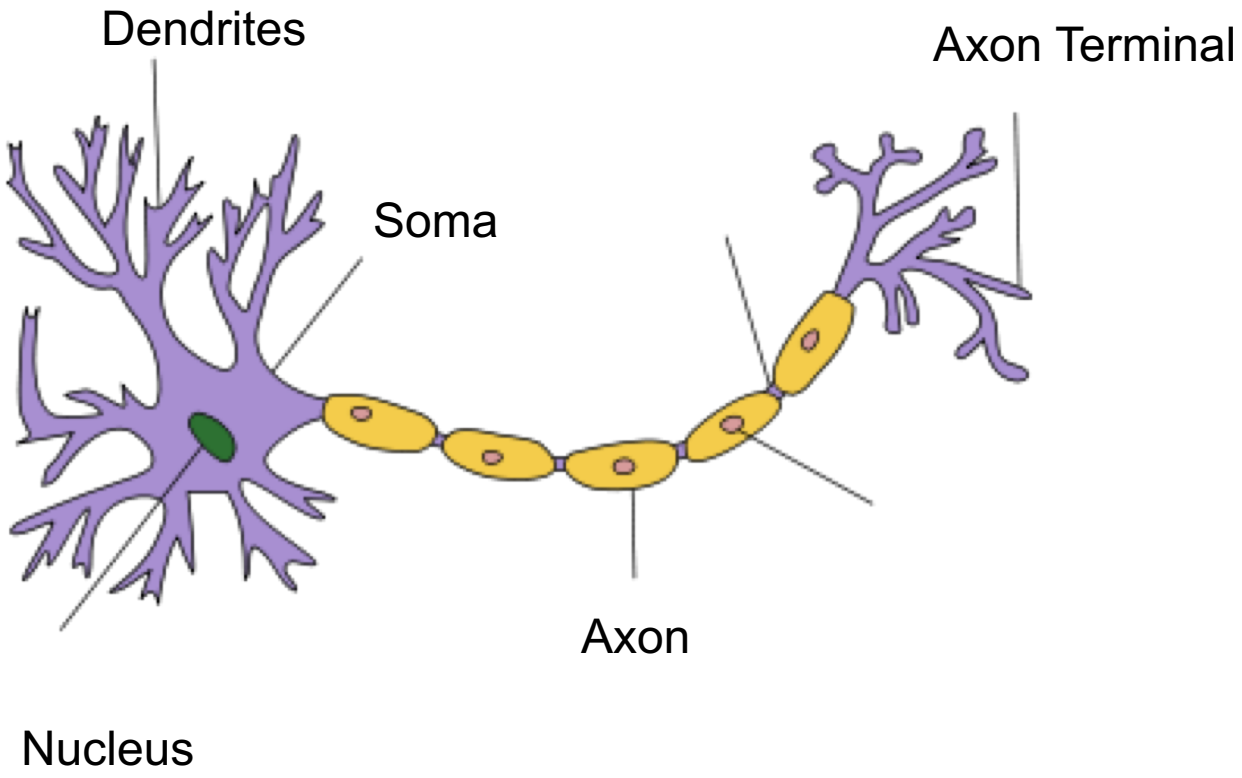
Introduction Problems for Neural Networks

- Examples may be described by a large number of attributes (e.g., pixels in an image).
- The output value can be discrete, continuous, or a vector of either discrete or continuous values.
- Data may contain errors.
- The time for training may be extremely long, although evaluating the network for a new example is relatively fast.
- Interpretability of the final hypothesis is not relevant (the NN is treated as a black box).

Perceptron as one Type of Linear Discriminants

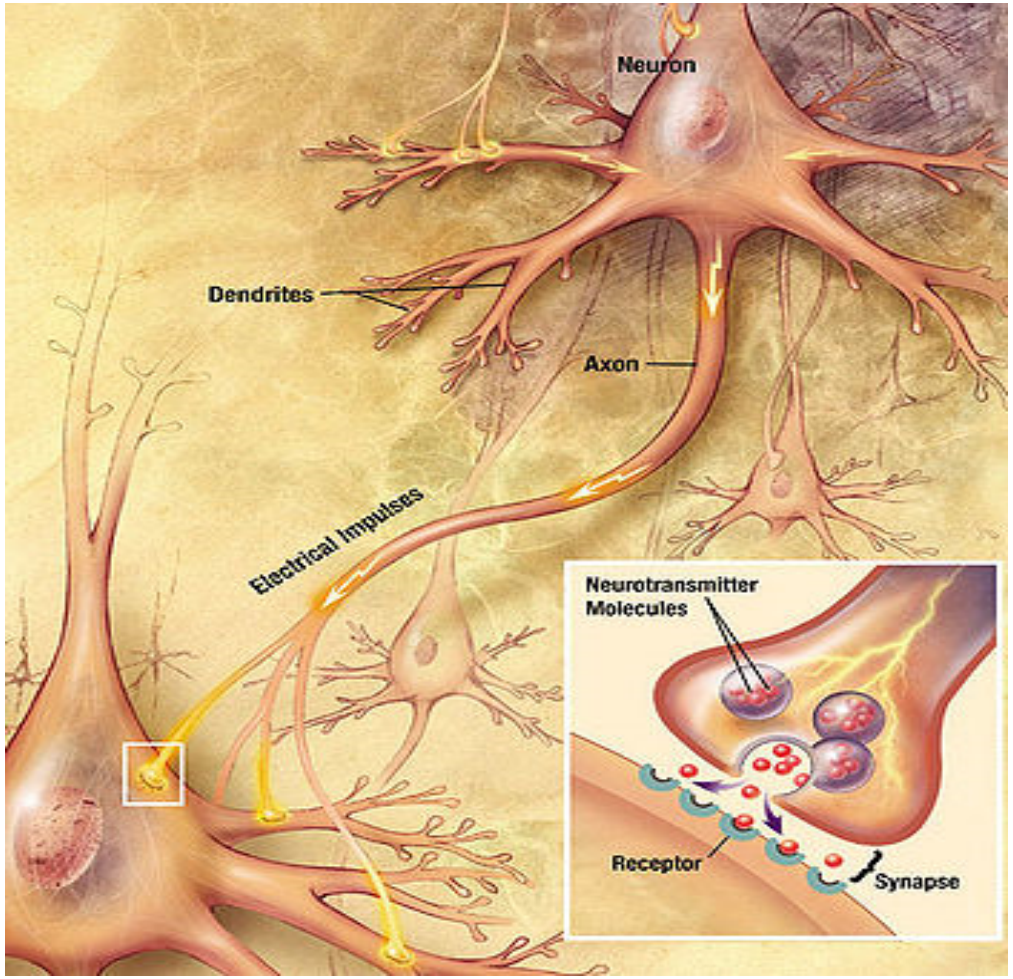
- Introduction
- Design of Primitive Units
  - ✓ Perceptrons
- Security applications of perceptron

Perceptron as one Type of Linear Discriminants



(image copied from wikipedia, entry on “neuron”)

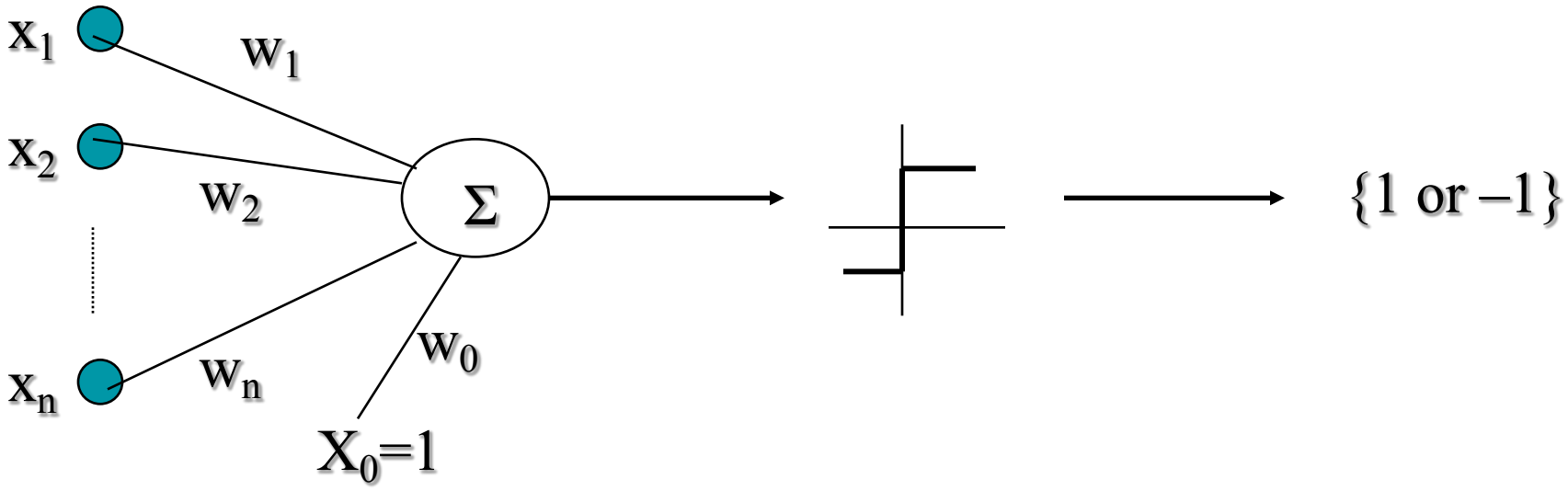
Perceptron as one Type of Linear Discriminants



(image copied from wikipedia, entry on “neuron”)

Design of Primitive Units Perceptrons

Definition. It's a step function based on a linear combination of real-valued inputs. If the combination is above a threshold it outputs a 1, otherwise it outputs a –1.



Design of Primitive Units Learning Perceptrons

$$O(x_1,x_2,\ldots,x_n) = \begin{cases} 1 & \text{if } w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

To simplify our notation we can represent the function as follows:

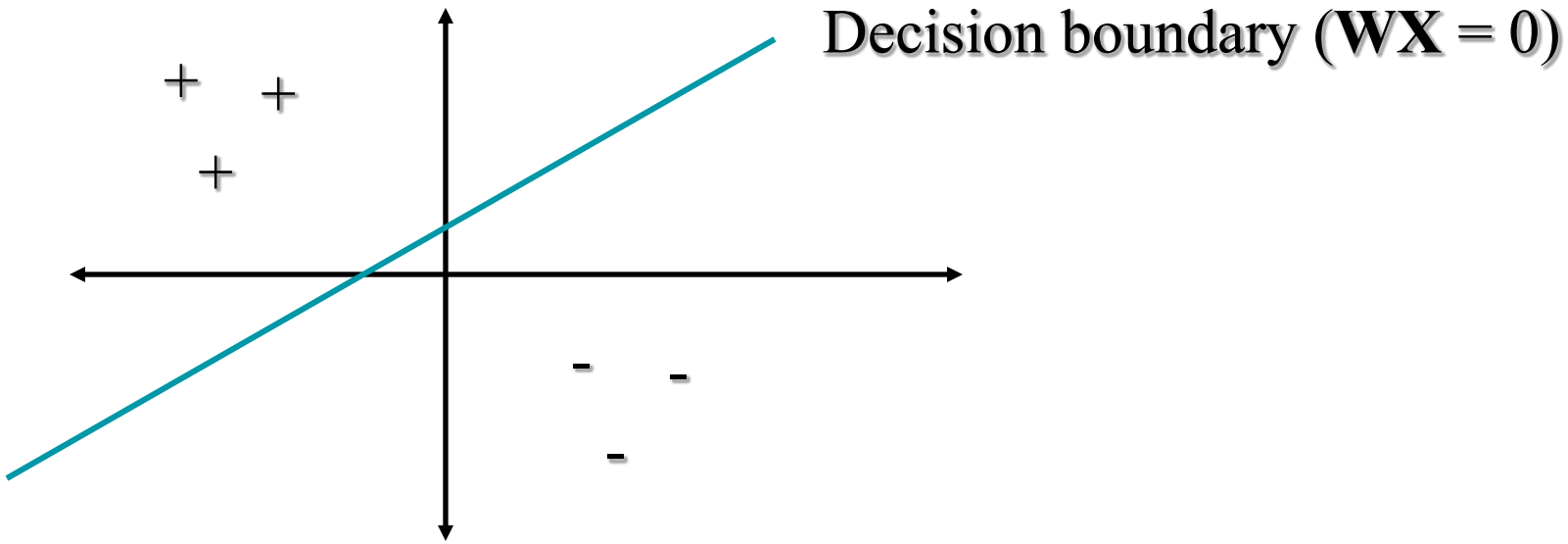
$$O(\mathbf{X}) = \text{sgn}(\mathbf{WX}) \text{ where}$$

$$\text{sgn}(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases}$$

Learning a perceptron means finding the right values for W.
The hypothesis space of a perceptron is the space of all weight vectors.

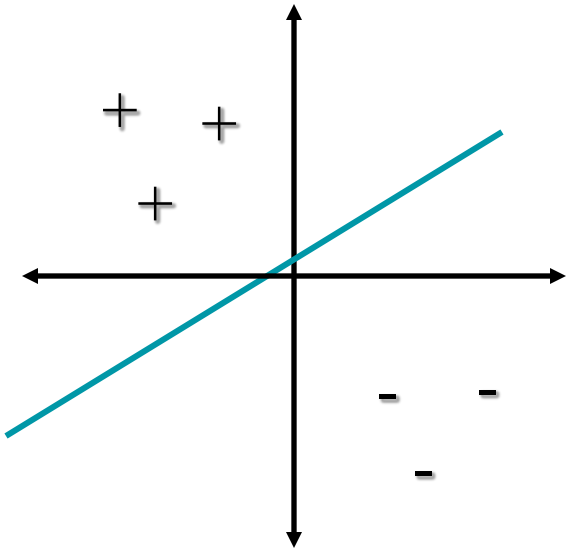
Design of Primitive Units Representational Power

A perceptron draws a hyperplane as the decision boundary over the (n-dimensional) input space.

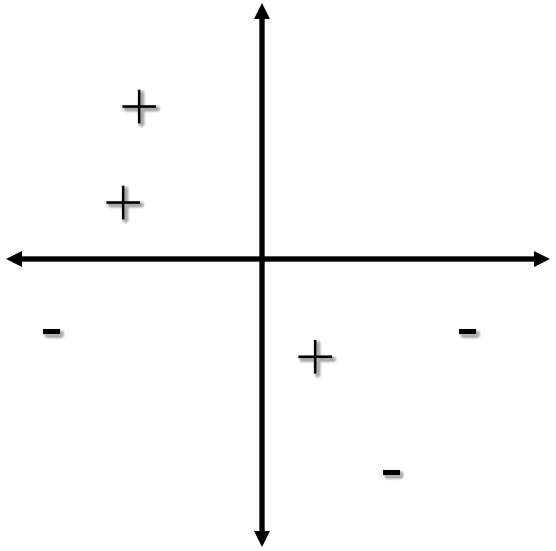


Design of Primitive Units Representational Power

A perceptron can learn only examples that are called "linearly separable". These are examples that can be perfectly separated by a hyperplane.



Linearly separable

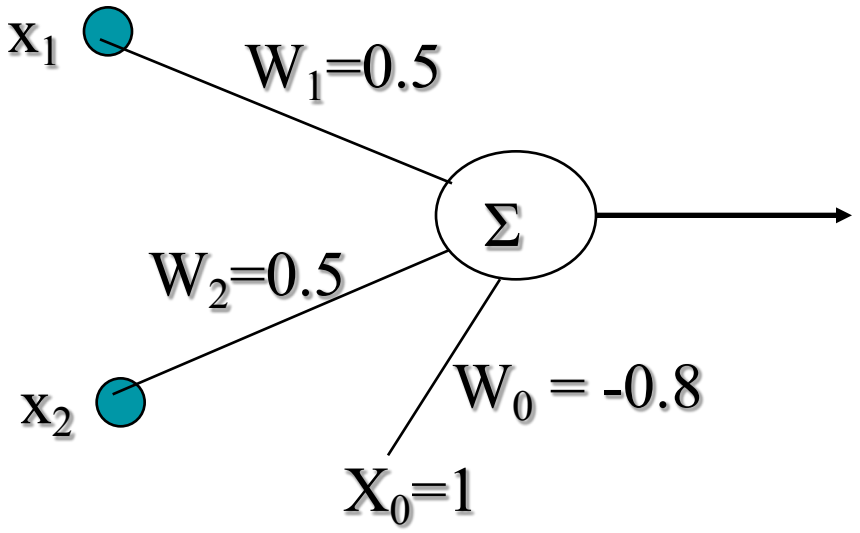


Non-linearly separable

Design of Primitive Units Functions for Perceptrons

- Perceptrons can learn many boolean functions:
  - ✓ AND, OR, NAND, NOR, but not XOR

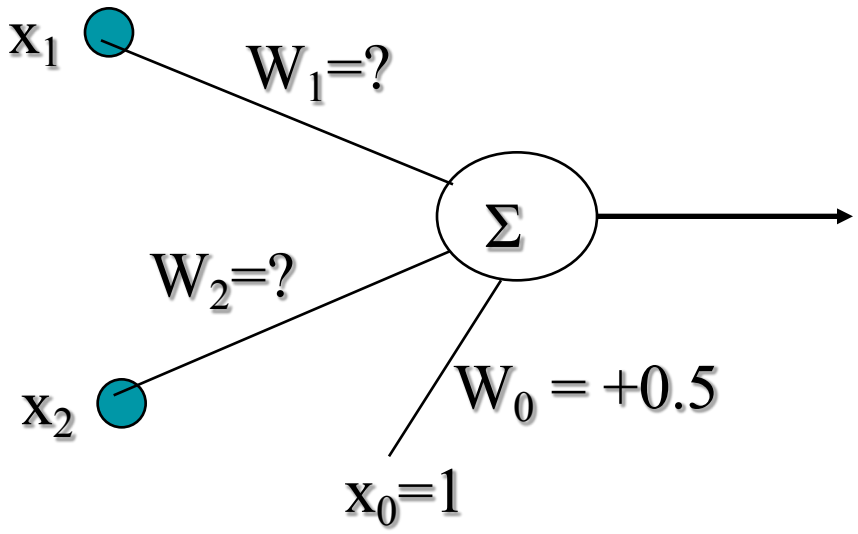
AND:



- Every boolean function can be represented with a perceptron network that has two levels of depth or more.

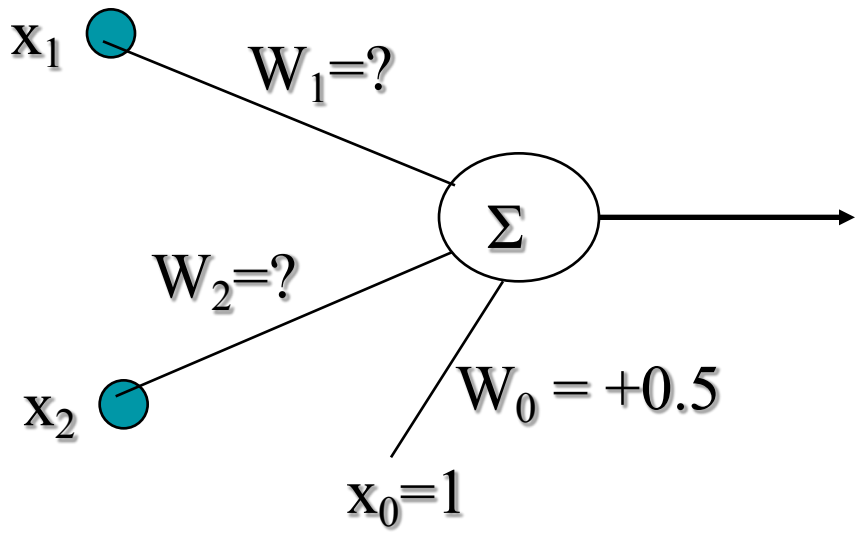
Design of Primitive Units

- Design a two-input perceptron that implements the Boolean function X1 OR ~X2 (X1 OR not X2).
- Assume the independent weight is always +0.5
- (assume W0 = +0.5 and X0 = 1). You simply have to provide adequate values for W1 and W2.



Design of Primitive Units

X1 OR ~X2 (X1 OR not X2)



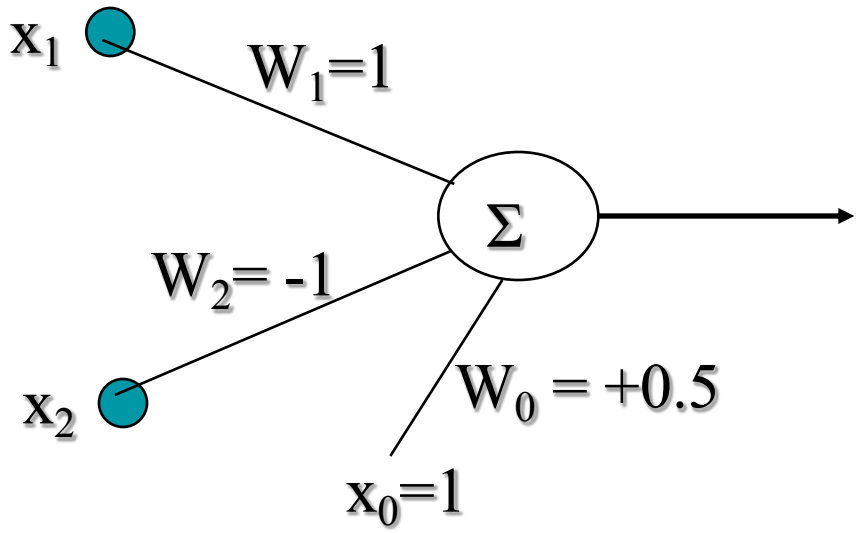
Design of Primitive Units

There are different solutions, but a general solution is that:

$W_2 < -0.5$

$W_1 > |W_2| - 0.5$

where |x| is the absolute value of x.



(X_1 OR not X_2).

Design of Primitive Units Perceptron Algorithms

How do we learn the weights of a single perceptron?

A. Perceptron rule

B. Delta rule

Algorithm for learning using the perceptron rule:

1. Assign random values to the weight vector
2. Apply the perceptron rule to every training example
3. Are all training examples correctly classified?

a) Yes. Quit

b) No. Go back to Step 2.

Design of Primitive Units

A. Perceptron Rule

The perceptron training rule:

For a new training example X = (x1, x2, …, xn), update each weight according to this rule:

$w_i = w_i + \Delta w_i$

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

t: target output

o: output generated by the perceptron

η: constant called the learning rate (e.g., 0.1)

Design of Primitive Units

A. Perceptron Rule

Comments about the perceptron training rule:

•If the example is correctly classified the term (t-o) equals zero, and no update on the weight is necessary.

•If the perceptron outputs –1 and the real answer is 1, the weight is increased.

•If the perceptron outputs a 1 and the real answer is -1, the weight is decreased.

•Provided the examples are linearly separable and a small value for η is used, the rule is proved to classify all training examples correctly (i.e, is consistent with the training data).

Historical Background

Early attempts to implement artificial neural networks:

•McCulloch (Neuroscientist) and Pitts (Logician) (1943)

- ✓ Based on simple neurons (MCP neurons)
- ✓ Based on logical functions

Walter Pitts (right)
(extracted from Wikipedia)

Historical Background

Donald Hebb (1949)

The Organization of Behavior.

“Neural pathways are strengthened every time they are used.”

Picture of Donald Hebb
(extracted from Wikipedia)

Design of Primitive Units

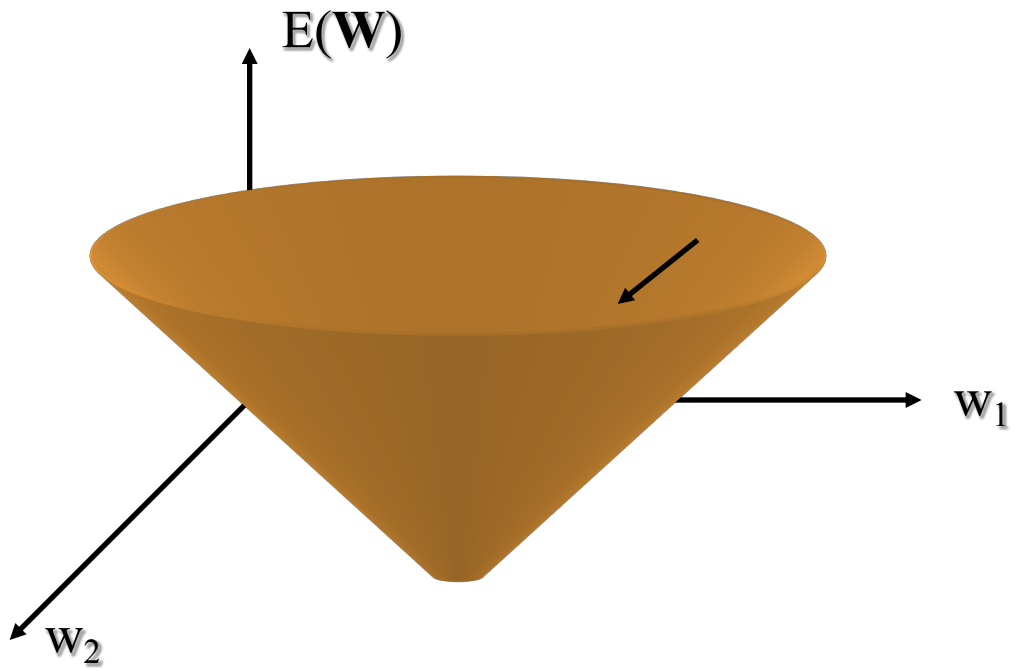
B. The Delta Rule

- What happens if the examples are not linearly separable?
- To address this situation we try to approximate the real concept using the delta rule.
- The key idea is to use a *gradient descent search*
  - ✓ We will try to minimize the following error: $E = \frac{1}{2} \sum_i (t_i - o_i)^2$
  - ✓ where the sum goes over all training examples.
  - ✓ Here oi is the inner product WX and not sgn(WX) as with the perceptron algorithm.

Design of Primitive Units

B. The Delta Rule

The idea is to find a minimum in the space of weights and the error function E:



Design of Primitive Units

Derivation of the Rule

The gradient of E with respect to weight vector W, denoted as $\nabla E(W)$:

$$\Delta E(W) = \left[\frac{E'}{w_0}, \frac{E'}{w_1}, \ldots, \frac{E'}{w_n} \right]$$

$\nabla E(W)$ is a vector with the partial derivatives of E with respect to each weight w_i.

Key concept:

The gradient vector points in the direction with the steepest increase in E.

Design of Primitive Units

B. The Delta Rule

The delta rule:

For a new training example X = (x1, x2, …, xn),

update each weight according to this rule:

$$w_i = w_i + \Delta w_i$$

Where $$\Delta w_i = -\eta \frac{E'(W)}{w_i}$$

η: learning rate (e.g., 0.1)

Design of Primitive Units

The gradient

How do we compute $\nabla E(W)$?

It is easy to see that

$$\frac{E'(W)}{w_i} = \sum_i (t_i - o_i)(-x_i)$$

So that gives us the following equation:

$$\Delta w_i = \eta \Sigma_i (t_i - o_i) x_i$$

Design of Primitive Units The algorithm using the Delta Rule

Algorithm for learning using the delta rule:

1. Assign random values to the weight vector
2. Continue until the stopping condition is met
   a) Initialize each Δw_i to zero
   b) For each example:
      Update Δw_i:
      $\Delta w_i = \Delta w_i + n\,(t - o)\,x_i$
   c) Update w_i:
      $w_i = w_i + \Delta w_i$
3. Until error is small

Historical Background

- Frank Rosenblatt (1962)
- He showed how to make incremental changes on the strength of the synapses.
- He invented the Perceptron.

Picture of Marvin Minsky (2008)
(extracted from Wikipedia)

- Minsky and Papert (1969)
- Criticized the idea of the perceptron.
- Could not solve the XOR problem.
- In addition, training time grows exponentially with the size of the input.

Design of Primitive Units Difficulties with Gradient Descent

There are two main difficulties with the gradient descent method:

- ✓ Convergence to a minimum may take a long time.
- ✓ There is no guarantee we will find the global minimum.

Design of Primitive Units Stochastic Approximation

Instead of updating every weight until all examples have been observed, we update on every example:

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

In this case we update the weights "incrementally".

Remarks:
- When there are multiple local minima stochastic gradient descent may avoid the problem of getting stuck on a local minimum.
- Standard gradient descent needs more computation but can be used with a larger step size.

Design of Primitive Units Difference between Perceptron and Delta Rule

- The perceptron is based on an output from a step function, whereas the delta rule uses the linear combination of inputs directly.

- The perceptron is guaranteed to converge to a consistent hypothesis assuming the data is linearly separable. The delta rules converges in the limit but it does not need the condition of linearly separable data.