19Z601- MACHINE LEARNING

UNIT- 3

NEURAL NETWORKS AND DECISION TREES : **Feed-forward Networks - Network Training - Delta Rule- Gradient Descent – Error Backpropagation - Regularization in Neural Networks - Generalisation - Decision Tree Learning- Representation - Inductive Bias- Issues** (9)

Presented by

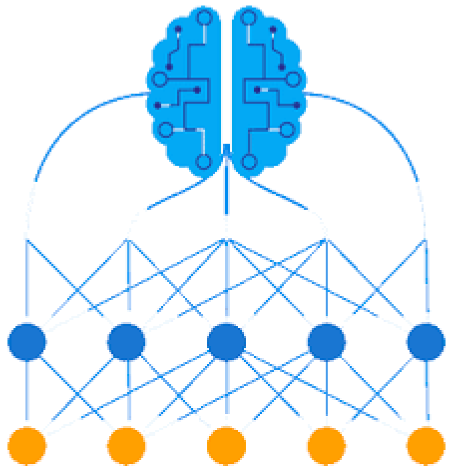
Ms.Anisha.C.D

Assistant Professor

CSE

NEURAL NETWORKS

- Artificial neural networks (ANNs) provide a **general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples**.
- Algorithms such as **BACKPROPAGATION** use **gradient descent** to tune network parameters to best fit a training set of input-output pairs.
- ANN learning is **robust to errors in the training data** and has been successfully applied to problems such as **interpreting visual scenes, speech recognition, and learning robot control strategies**.

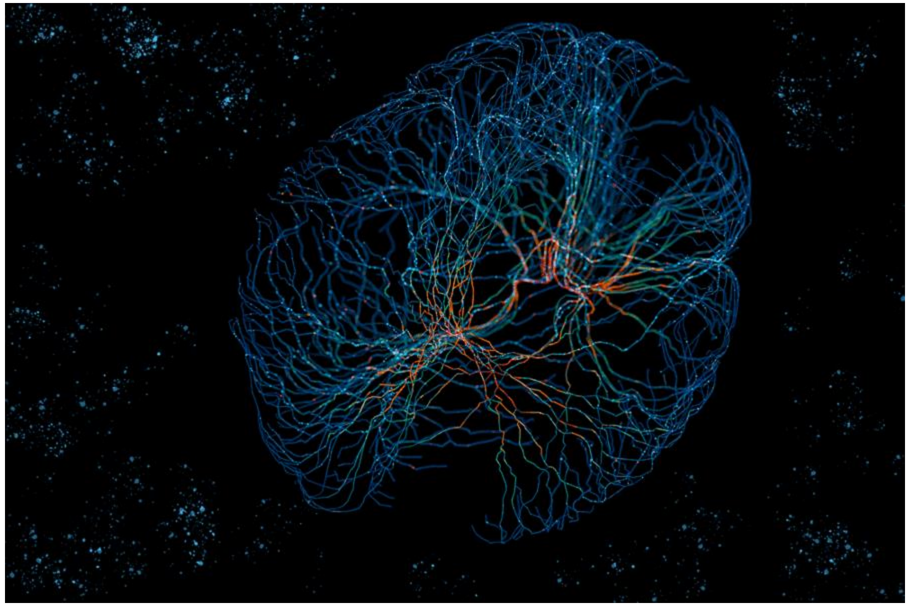


INTRODUCTION

- **Neural network learning methods** provide a **robust approach to approximating real-valued, discrete-valued, and vector-valued target functions.**
- For **certain types of problems, such as learning to interpret complex real-world sensor data**, artificial neural networks are among the most effective learning methods currently known.
- For example**, the BACKPROPAGATION algorithm** has proven surprisingly successful in many practical problems :
  - **such as learning to recognize handwritten characters**
  - **learning to recognize spoken words**
  - **and learning to recognize faces**

BIOLOGICAL MOTIVATION

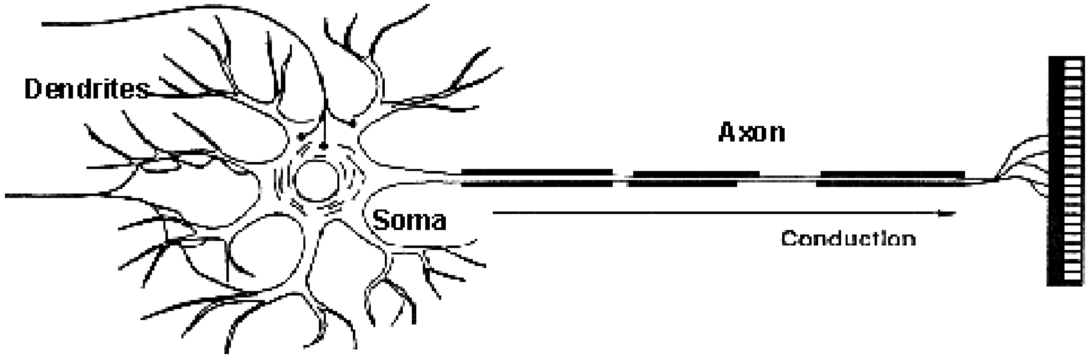
- The **study of artificial neural networks (ANNs)** has been **inspired** in part by the **observation that biological learning systems are built of very complex webs of interconnected neurons.**



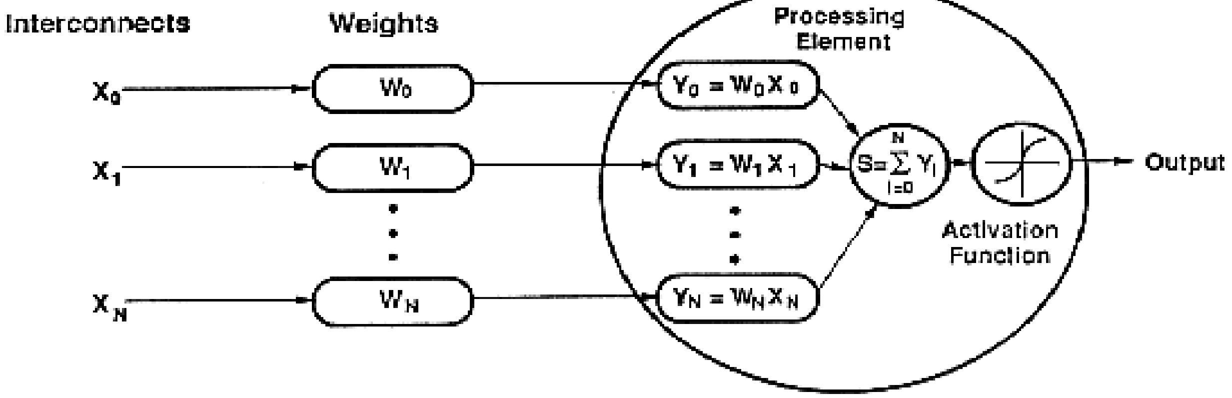
- In rough analogy, **artificial neural networks** are built out of a **densely interconnected set of simple units, where each unit takes a number of real-valued inputs** (possibly the outputs of other units) and produces a **single real-valued output** (which may become the input to many other units).

BIOLOGICAL MOTIVATION – HOW DO ANN WORK?

Biological Neuron



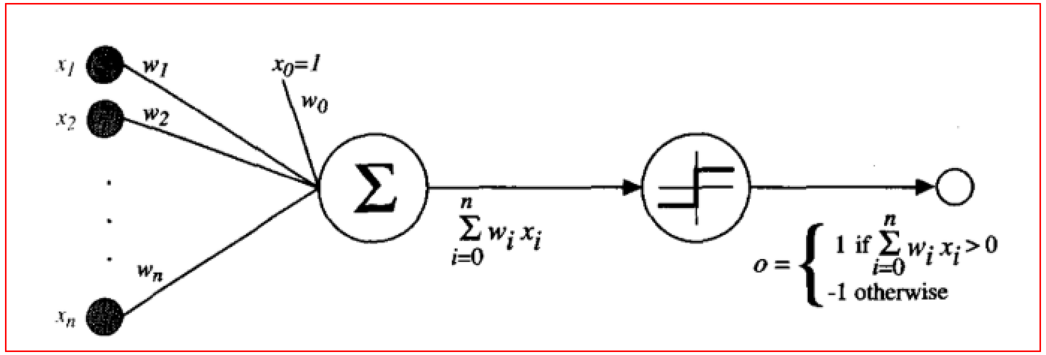
Artificial Neuron



PERCEPTRON

- **One type of ANN system is based on a unit called a perceptron.**

- **A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.**



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

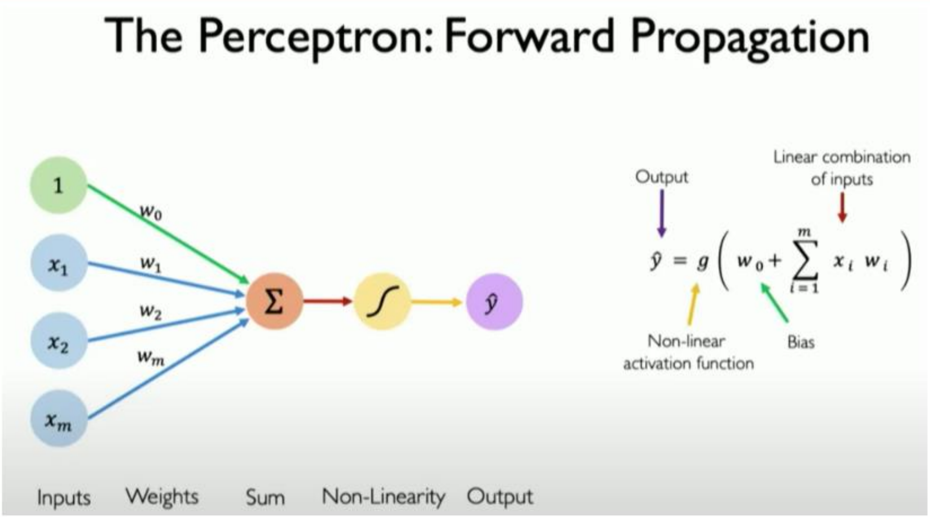
INNER WORKING OF PERCEPTRON

A 'Perceptron' is the basic building block, or single node, of a neural network inspired from the neurons that are found in the brain. It operates by taking in a set of inputs, calculating a weighted sum, adding a bias term, and then applying an activation function to this sum to produce an output. The inner working of a perceptron is as follows:

1. A vector of $x_1,\ldots\ldots x_m$ inputs is passed to the algorithm
2. Weightings $w_1,\ldots\ldots w_m$ are applied to each element of the input vector and a bias is passed along with as represented by w_0
3. Summation of the input and bias terms is performed

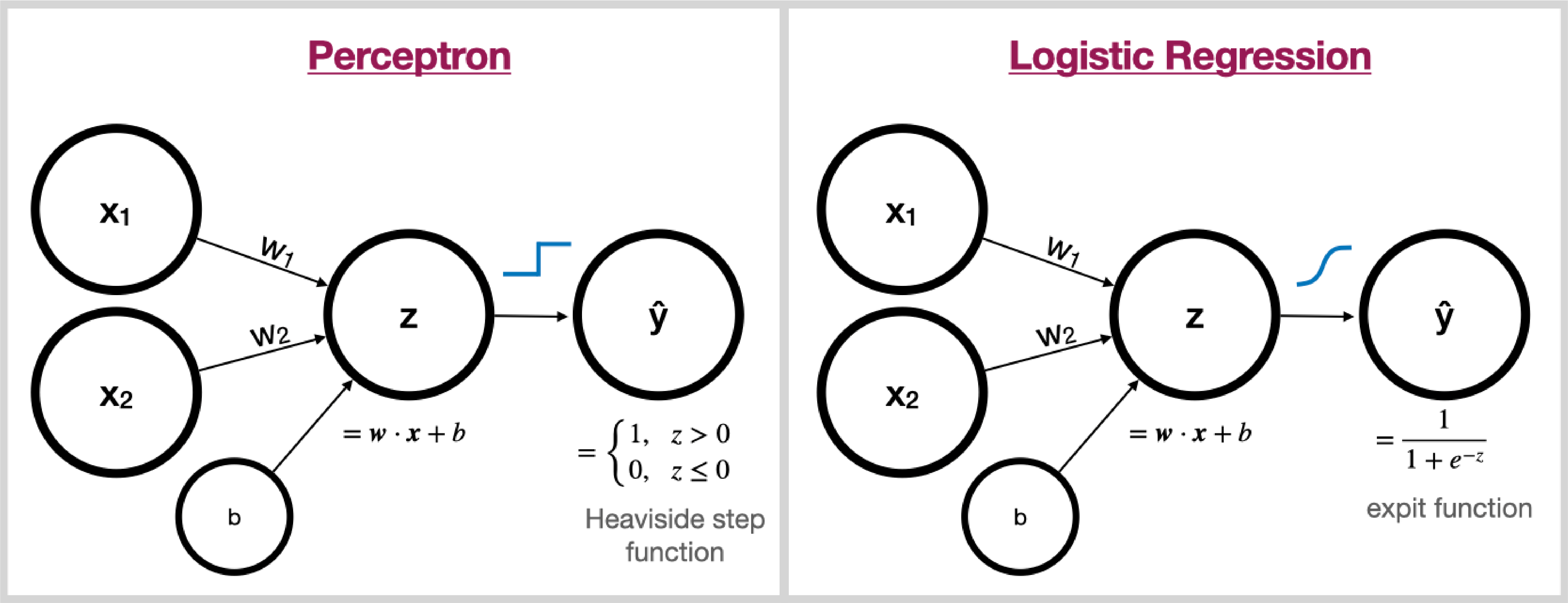
$$w_0 + \sum_{i=1}^{m} x_i\, w_i$$

4. The above sum is passed to an activation function, g.
5. The activation function then returns a output, $\hat{y}$, based on which classification decision is taken



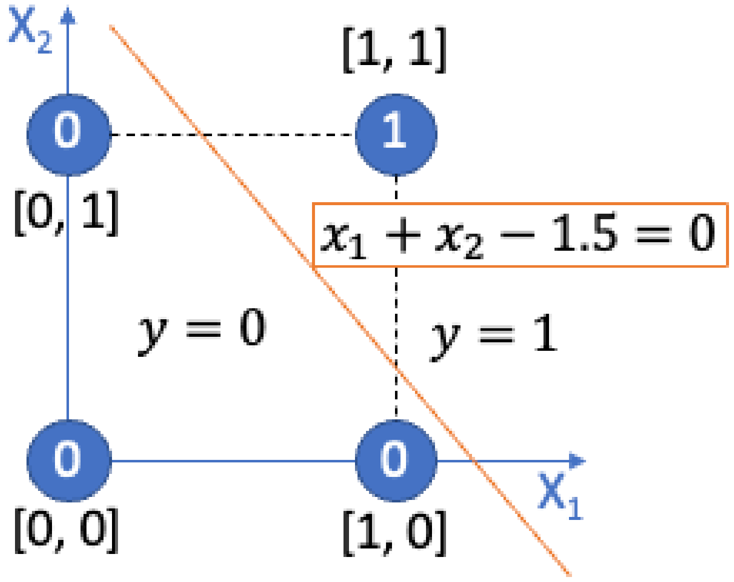
Perceptron, a neuron / node in neural network
Source: *MIT Deep Learning Course*

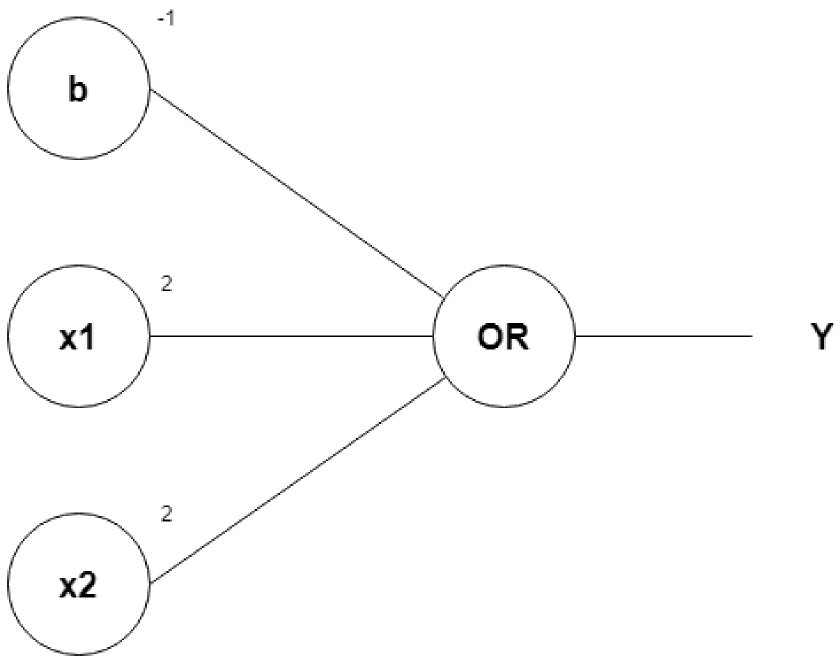
Perceptron Vs Logistic Regression Representation as Perceptron



EXAMPLE : AND GATE REPRESENTATION

X_1	X_2	y
0	0	0
0	1	0
1	0	0
1	1	1





HOMEWORK

Represent OR, NAND, NOR, XOR in graph as shown in previous slide.

Find out if its linearly separable.

If yes represent using perceptron

PERCEPTRON TRAINING RULE

- **Main Objective :** To determine a **weight vector** that causes the perceptron to produce the correct +/-1 output for each of the given training examples.
- Several algorithms are known to solve this learning problem.
- Here **two algorithms** are considered:
  - The perceptron rule and (suitable for linearly separable)
  - The delta rule

PERCEPTRON TRAINING RULE: PERCEPTRON RULE

$$w_i \leftarrow w_i + \Delta w_i$$

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

Here,

t is the target output for the current training example,

o is the output generated by the perceptron, and

η is a positive constant called the learning rate.

PERCEPTRON TRAINING RULE:
DELTA RULE (GRADIENT DESCENT)

Step 1 : First stage of Perceptron : Linear Unit

$$\boxed{o(\vec{x}) = \vec{w} \cdot \vec{x}}$$

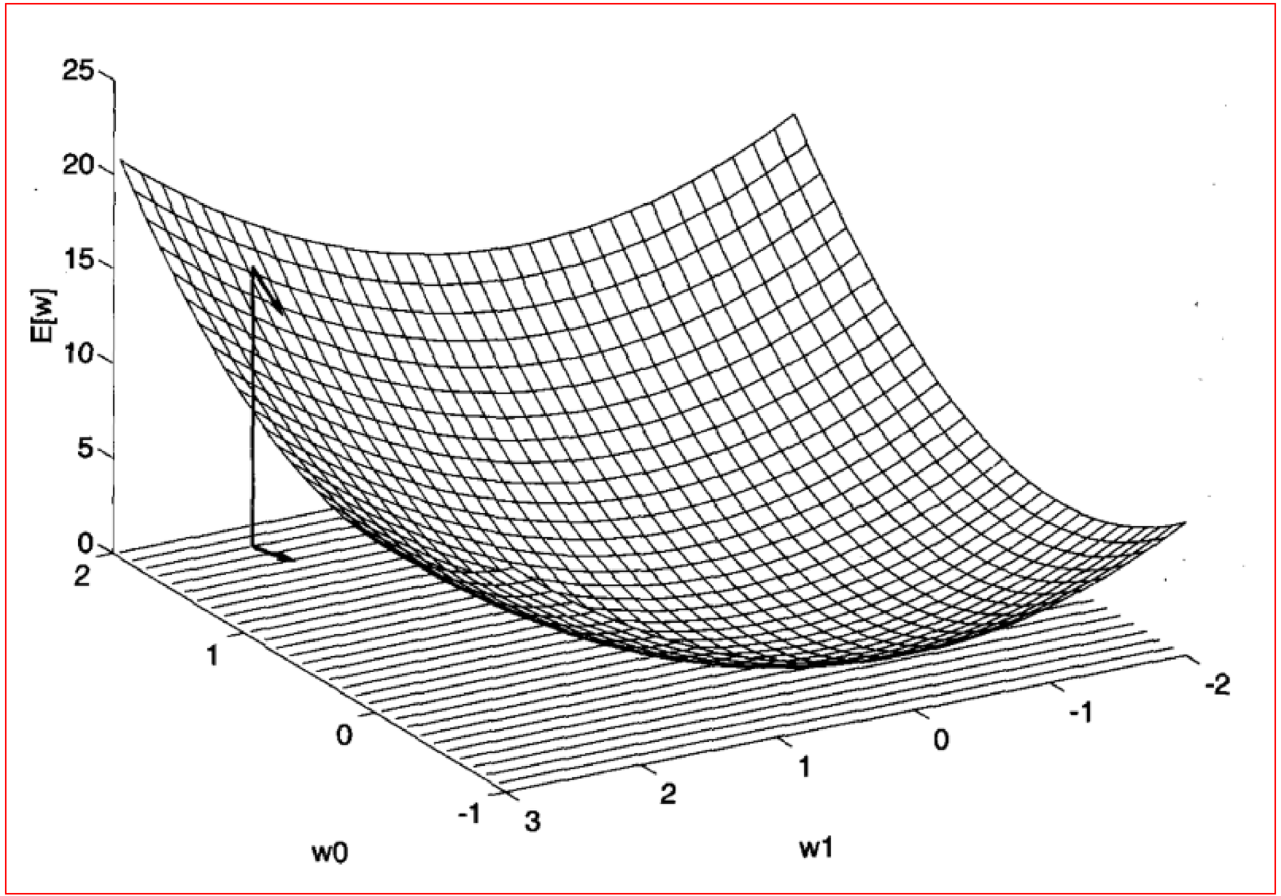
Step 2 : Specifying a measure for the training error of a hypothesis (weight vector)

$$\boxed{E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2}$$

where D is the set of training examples,
t_d is the target output for training example d, and
o_d is the output of the linear unit for training example d.

VISUALIZING THE HYPOTHESIS SPACE

To visualize the entire hypothesis space of possible weight vectors and their associated E values



DERIVATION OF GRADIENT DESCENT

- **How can we calculate the direction of steepest descent along the error surface?**

  This direction can be found by computing the derivative of E with respect to each component of the vector $\vec{w}$ This vector derivative is called the gradient of E with $\vec{w}$, written as :

$$\boxed{\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]}$$

When interpreted as a vector in weight space, the gradient specifies the direction that produces the steepest increase in E. The negative of this vector therefore gives the direction of steepest decrease.

DERIVATION OF GRADIENT DESCENT

Step 4 : The training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

$$\Delta\vec{w} = -\eta \nabla E(\vec{w})$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

DERIVATION OF GRADIENT DESCENT

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d-o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d-o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d-o_d)\frac{\partial}{\partial w_i}(t_d-o_d) \\
&= \sum_{d\in D}(t_d-o_d)\frac{\partial}{\partial w_i}(t_d-\vec{w}\cdot\vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_{d\in D}(t_d-o_d)(-x_{id})
\end{aligned}$$

$$\boxed{\Delta w_i = \eta\sum_{d\in D}(t_d-o_d)\, x_{id}}$$

GRADIENT DESCENT ALGORITHM

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i \quad \text{(T4.1)}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \quad \text{(T4.2)}$$