Introduction to Bioinformatics

BS (CS – 460)

Lecture Set 06

Dr. Hafeez Ur Rehman

ANN - Parametric Models for Pattern Recognition

- Biological motivations
- Perceptrons
- Leading to ANN (Artificial Neural Networks)
- Leading to Deep Learning

Neural Networks- Biological Motivation

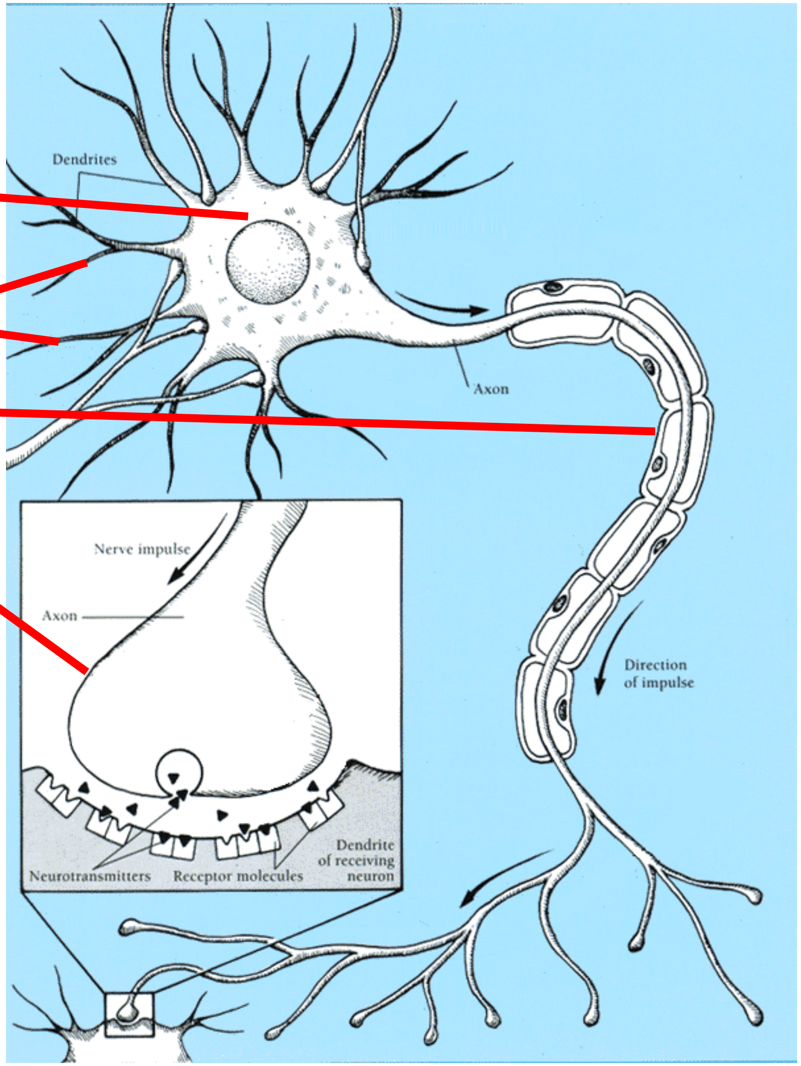
- **Analogy** to **biological** neural systems, the most robust learning systems we know.
- **Attempt** to understand **natural biological systems** through computational modeling.
- **Massive parallelism** allows for computational efficiency.
- Help understand "**distributed**" nature of neural representations (rather than "**localist**" representation) that allow robustness.
- **Intelligent behavior** as an "**emergent**" property of large number of simple units rather than from explicitly encoded symbolic rules and algorithms.

Neural Speed Constraints

- Neurons have a "**switching time**" on the order of a few **milliseconds**, compared to **nanoseconds** for current computing hardware.
- However, neural systems can **perform complex** cognitive tasks (vision, speech understanding) in **tenths of a second**.
- Only time for performing 100 serial steps in this time frame, compared to orders of magnitude more for current computers.
- Must be exploiting "**massive parallelism.**"
- Human brain has about $\mathbf{10^{11}}$ **neurons** with an average of $\mathbf{10^{4}}$ **connections each**.

Real Neurons

- Cell structures
  - Cell body
  - Dendrites
  - Axon
  - Synaptic terminals



Neural Communication

- Electrical potential across cell membrane exhibits spikes called action potentials.
- **Spike** originates in cell body, **travels** down **axon**, and causes **synaptic terminals** to release **neurotransmitters**.
- Chemical diffuses across synapse dendrites of other neurons.
- **Neurotransmitters** can be **excititory** or **inhibitory**.
- If **net input of neurotransmitters** to a neuron from other neurons is excititory and exceeds some threshold, it fires an **action potential**.

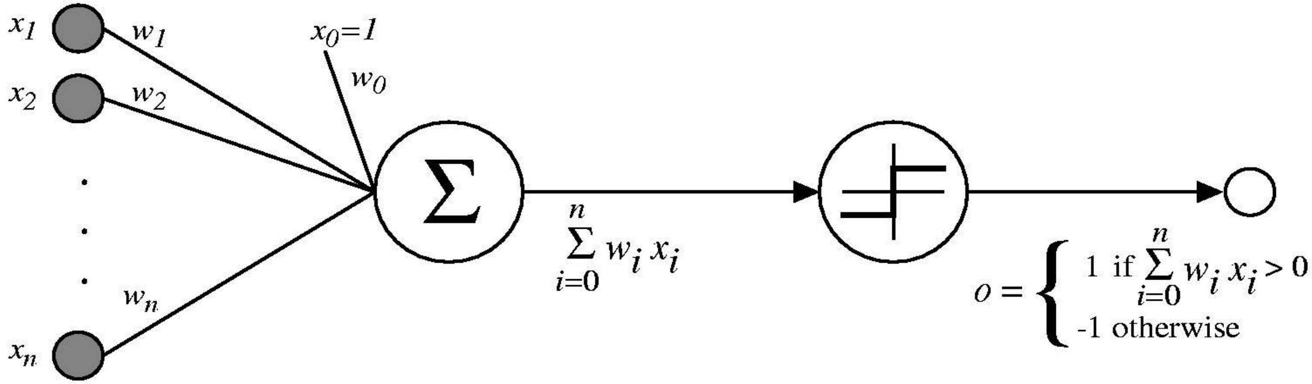
Neural Network Learning

- Learning approach based on modeling adaptation in biological neural systems.
- Perceptron: Initial algorithm for learning simple neural networks (single layer) developed in the 1950s by Frank Rosenblatt.
- Backpropagation: More complex algorithm for learning multi-layer neural networks developed in the 1980's.

Preview

- Perceptrons
- Gradient descent
- Multilayer networks
- Backpropagation

Perceptron



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

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Perceptron Training

- Assume supervised training examples giving the desired output for a unit given a set of known input activations.
- Learn synaptic weights so that unit produces the correct output for each example.
- Perceptron uses iterative update algorithm to learn a correct set of weights.

Perceptron Training Rule

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

where

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

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

Perceptron Training Rule

Can prove it will converge if

- Training data is linearly separable
- η sufficiently small

Perceptron Learning Algorithm

- Iteratively update weights until convergence.

```
1. Initialize weights to random values
2. Until outputs of all training examples are correct
      For each training pair, E, do:
       Compute current output o_j for E given its inputs
       Compare current output to target value, t_j , for E
       and update weight change.
```

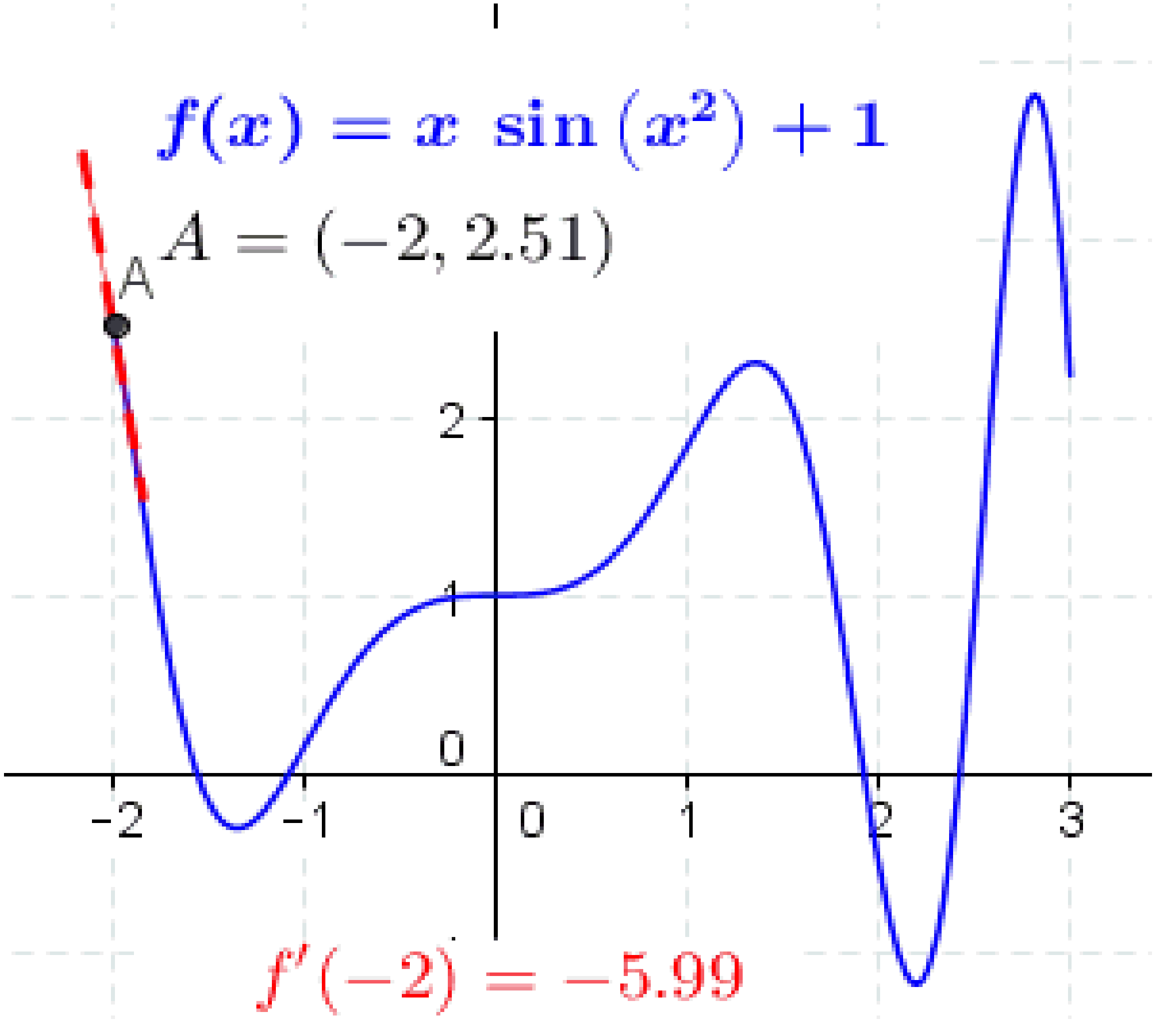
$$\Delta w_i = \Delta w_i + \eta (t - o) x_i$$

```
      Update synaptic weights (wi) and threshold using
      learning rule:
```

$$w_i = w_i + \Delta w_i$$

- Each execution of the outer loop is typically called an ***epoch***.

Derivative Example



Gradient Descent

To understand, consider simpler *linear unit*, where

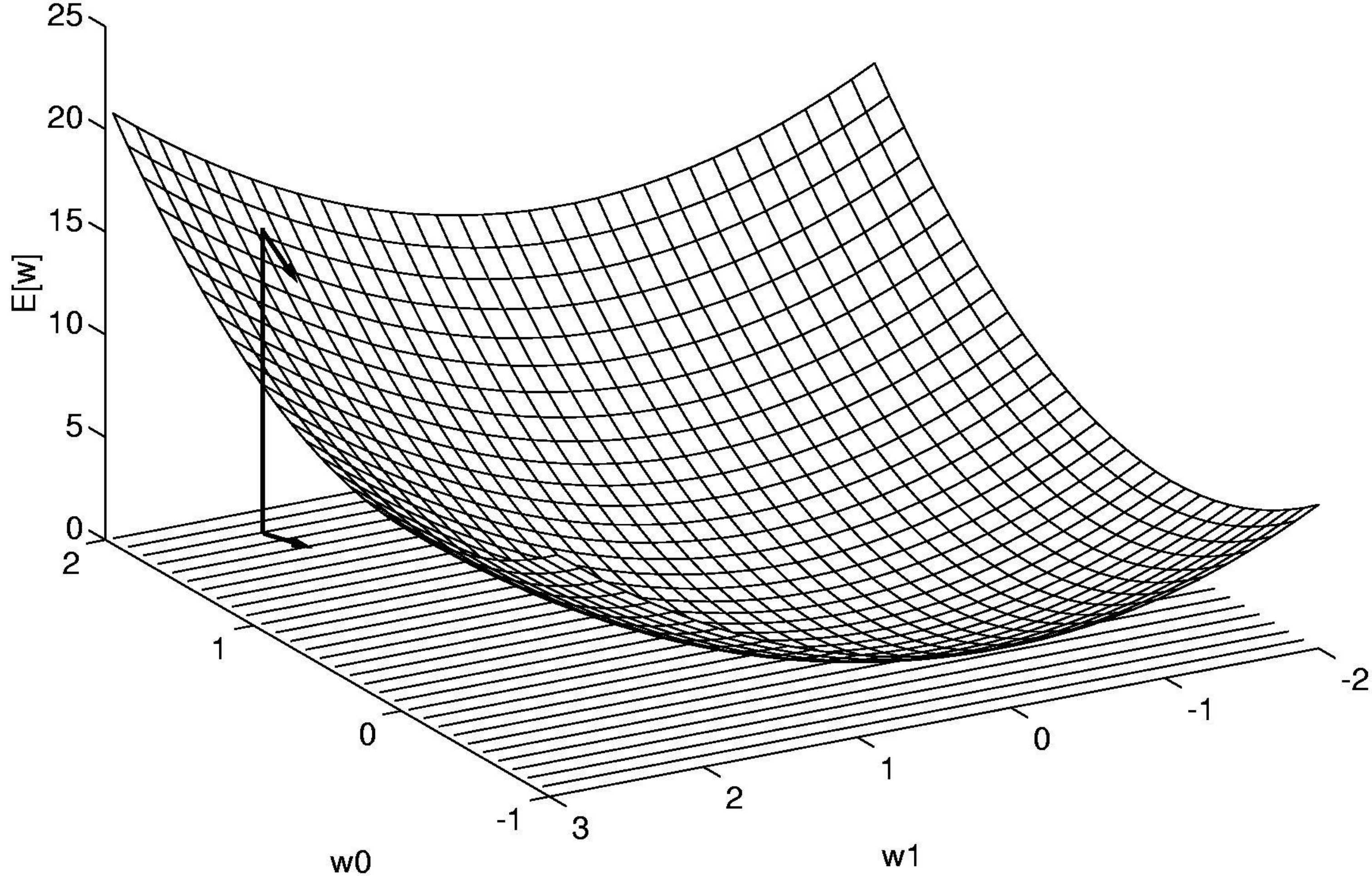
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

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

Where D is set of training examples

Gradient Descent



Gradient:

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

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

I.e.:

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

Gradient Descent

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}
$$

Gradient Descent

GRADIENT-DESCENT$(training_examples, \eta)$

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 instance $\vec{x}$ to unit and compute output o
  - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

- For each linear unit weight w_i, Do

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

Summary

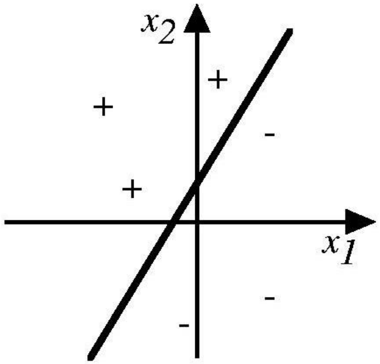
Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

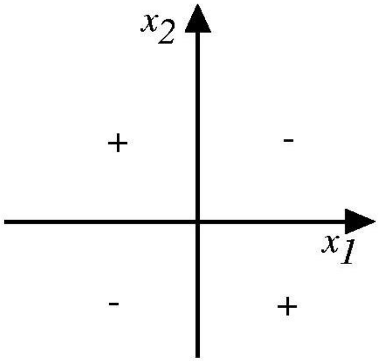
Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

Decision Surface of a Perceptron



(a)



(b)

Represents some useful functions

- What weights represent $g(x_1, x_2) = AND(x_1, x_2)$?

But some functions not representable

- All not linearly separable
- Therefore, we'll want networks of these...

For example $$p \oplus q \quad = \quad (p \vee q) \wedge \neg(p \wedge q)$$

The exclusive disjunction $p \oplus q$ can also be expressed in the following way:

$$p \oplus q \quad = \quad (p \wedge \neg q) \vee (\neg p \wedge q)$$