Artificial Intelligence (CS 401)

Machine Learning for Learning based Agents

Chapter 18: Learning from Examples

Dr. Hafeez UR REHMAN
Dept of Computer Science,
National University of Computer and Emerging Sciences, Peshawar, Pakistan.

Outline

- What is Machine Learning?
- Different types of learning problems
- Different types of learning algorithms
- Supervised learning
  - K-Nearest Neighbor (KNN)
  - **Perceptrons, Multi-layer Neural Networks…..Deep Learning**
  - Decision trees
  - Naïve Bayes
  - Boosting
- Unsupervised Learning
  - K-means
- Applications: e.g., learning to recognize digits, alphabets or some other patterns.

Non-Parametric Classifiers

- In non-parametric models, the complexity of the model grows with the increase in the training data.
- They are considered as non-parametric learning algorithms since the number of parameters grows with the size of the training set, the number of parameters may potentially be infinite.
- The typical examples include, K-nearest neighbor (KNN), decision trees, or RBF kernel SVMs.

Parametric Classifiers

- In a parametric model, we have a finite number of parameters.
- The size of the parameters doesn't grow with the increase in the training data.
- The artificial neural networks (ANNs), linear regression, logistic regression, and linear Support Vector Machines are typical examples of a parametric "learners;" here, we have a fixed size of parameters (the weight coefficient.)

1- Artificial Neural Networks (ANNs)

- Biological Motivations
- Perceptrons
- Leading to…
  - Neural Networks
  - a.k.a Multilayer Perceptron Networks
  - But more accurately: Multilayer Logistic Regression Networks

Neural Networks

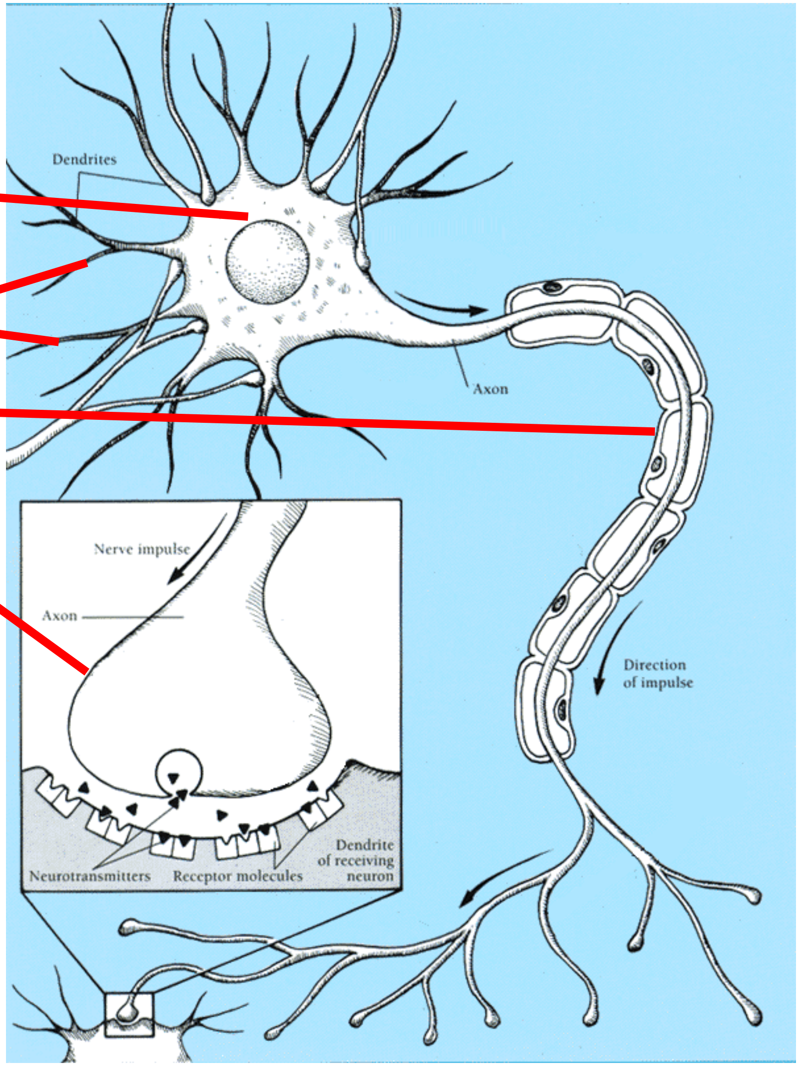
- **Analogy** to **biological** neural systems, the most robust learning systems we know.
- **Attempt** to understand **natural biological systems** through computational modeling.
- **Massive parallelism** allows for computational efficiency.
- Help understand "**distributed**" nature of neural representations (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 **10^{11} neurons** with an average of **10^4 connections each**.

Real Neurons

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

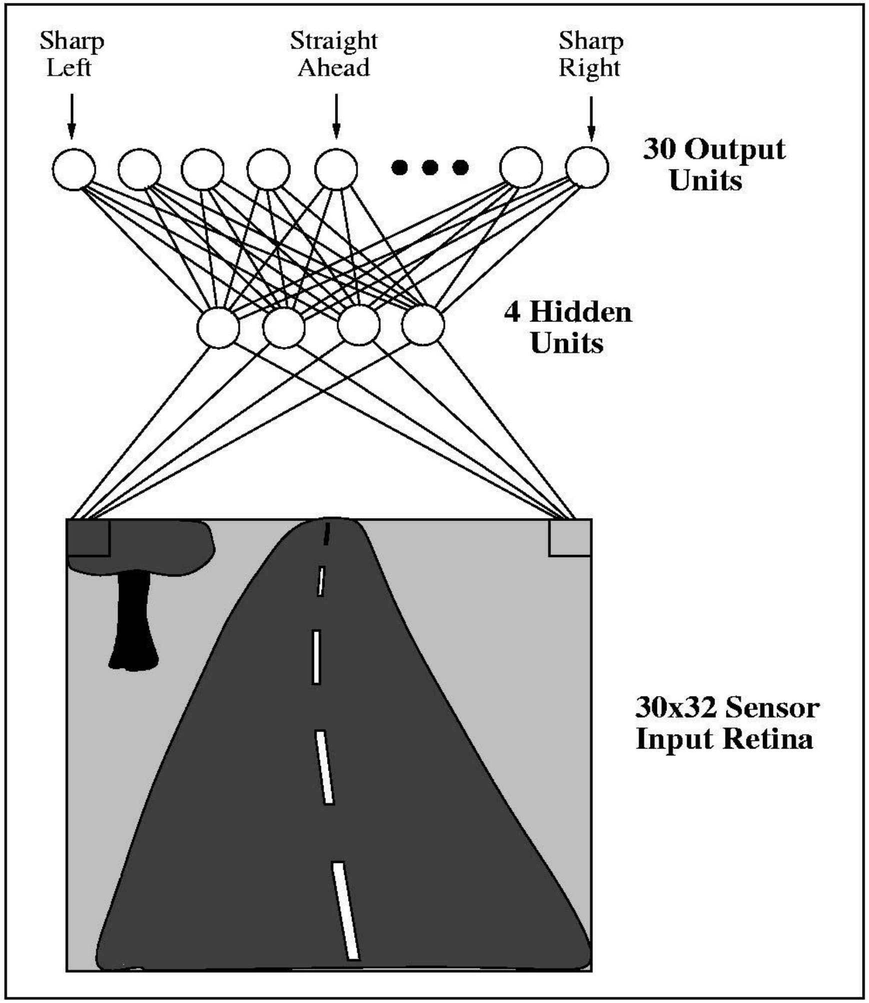


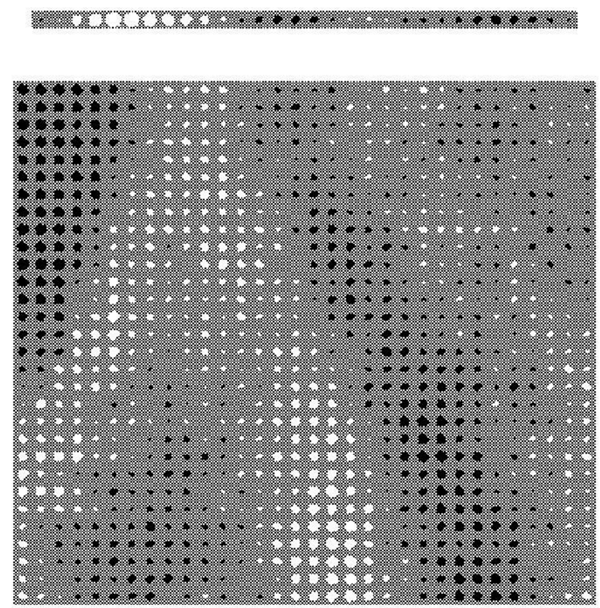
Neural Communication

- 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**.

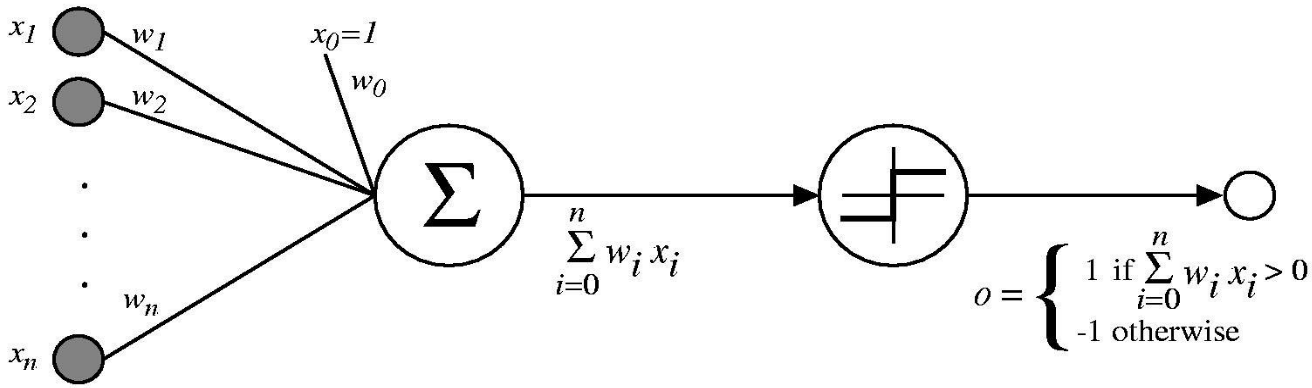
Neural Network Learning

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


Sharp
Left
Straight
Ahead
Sharp
Right
30 Output
Units
4 Hidden
Units
30x32 Sensor
Input Retina



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:

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

Neural Computation

- McCollough and Pitts (1943) showed how such model neurons could compute logical functions and be used to construct finite-state machines.
- Can be used to simulate logic gates:
  - AND: Let all w_{ji} be T_j/n, where n is the number of inputs.
  - OR: Let all w_{ji} be T_j
  - NOT: Let threshold be 0, single input with a negative weight.
- Can build arbitrary logic circuits, sequential machines, and computers with such gates.
- Given negated inputs, two layer network can compute any boolean function using a two level AND-OR network.

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 all weights to random values.
2. Until outputs of all training examples are correct:
      Initialize all Δwi's to zero.
      For each training pair, E, do:
       Compute current output oj for E given its inputs
       Compare current output to target value, tj , for E
       and update weight change.
       Δwi = Δwi + η(t - o) xi
      Update synaptic weights (wi) and threshold using
      learning rule:
       wi = wi + Δwi
```

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

$$w_i = w_i + \Delta w_i$$

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

Gradient Descent

To understand, consider simpler *linear unit*, where

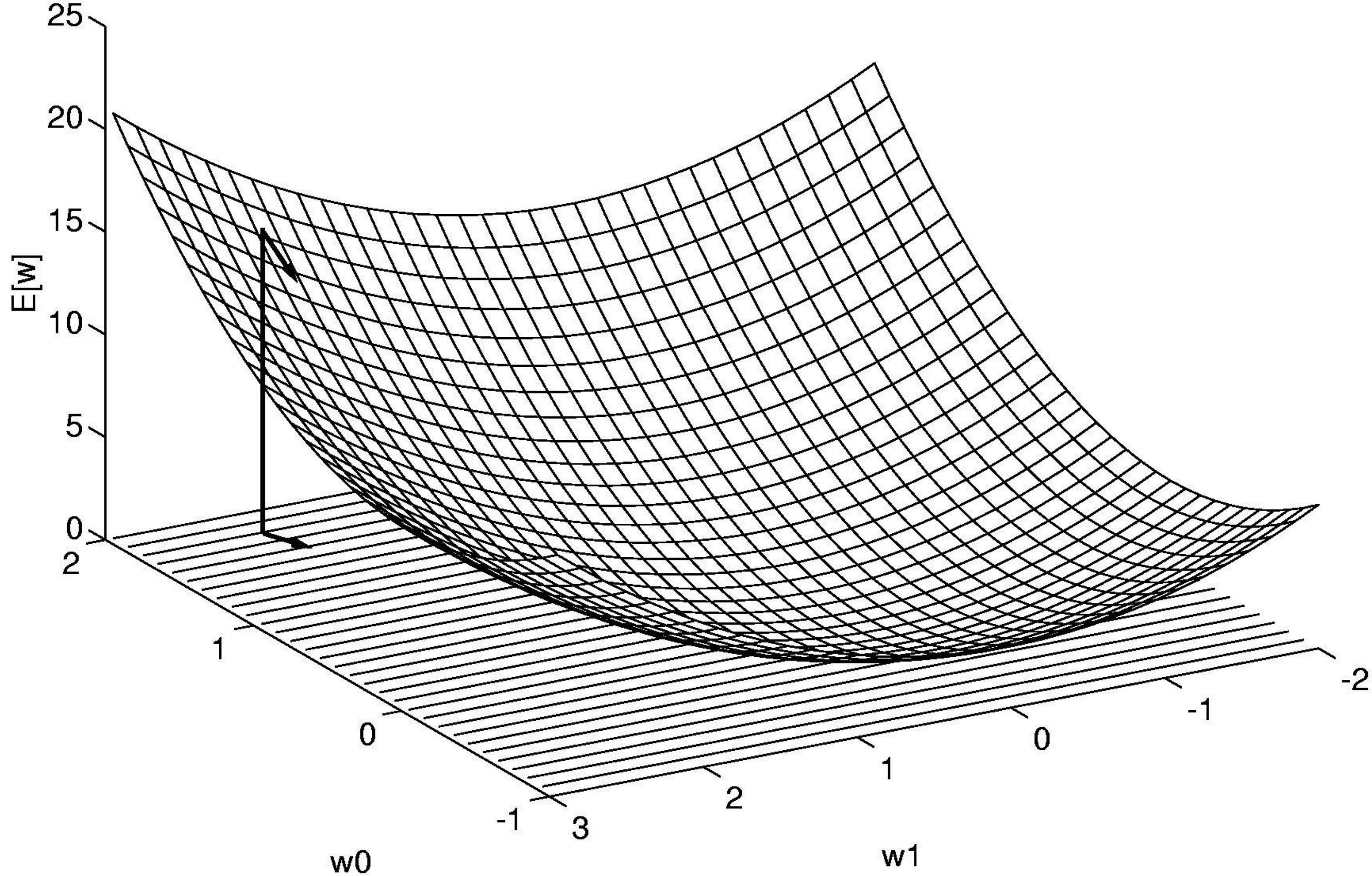
$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

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

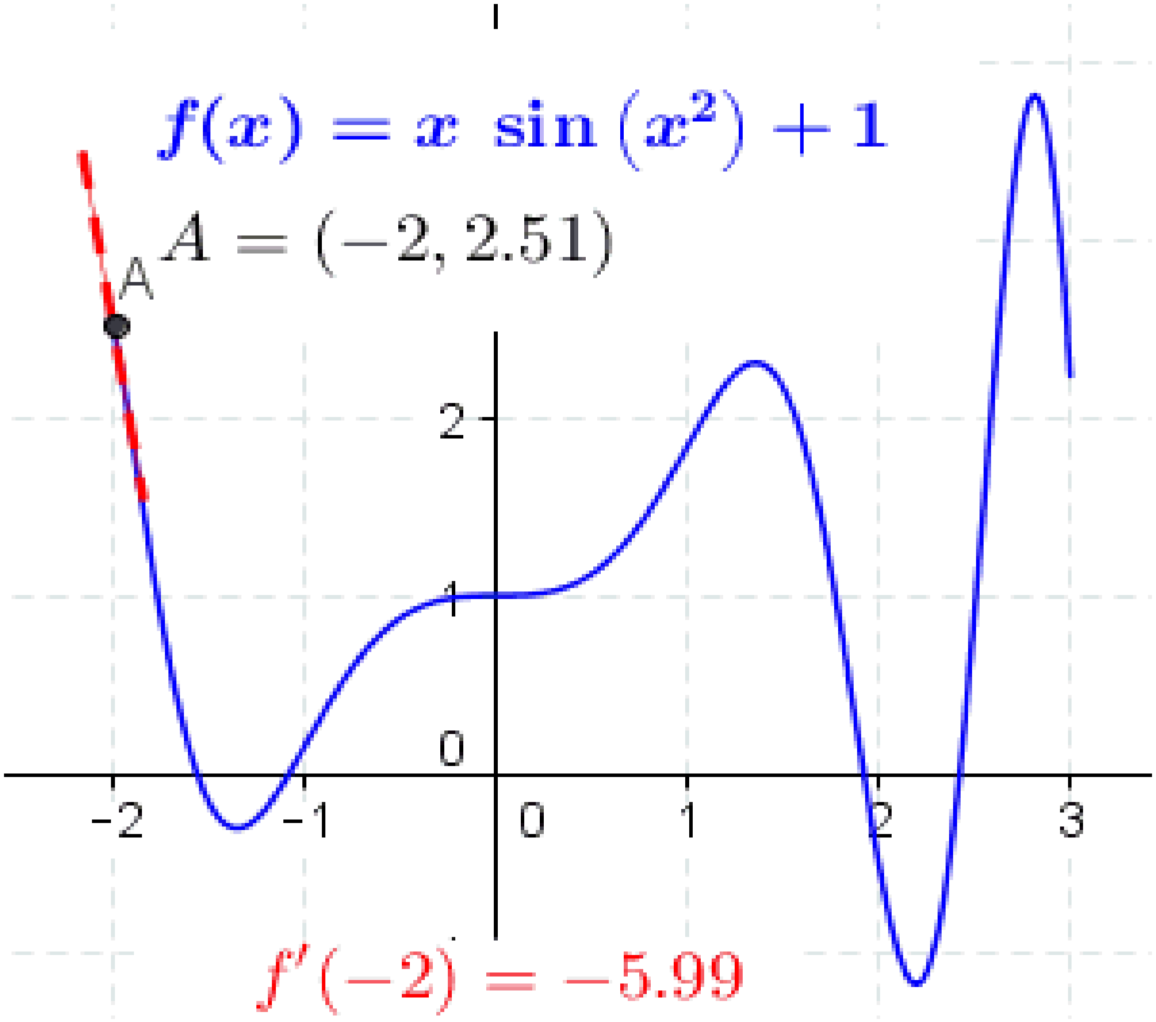
$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples

Gradient Descent



Derivative Example



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

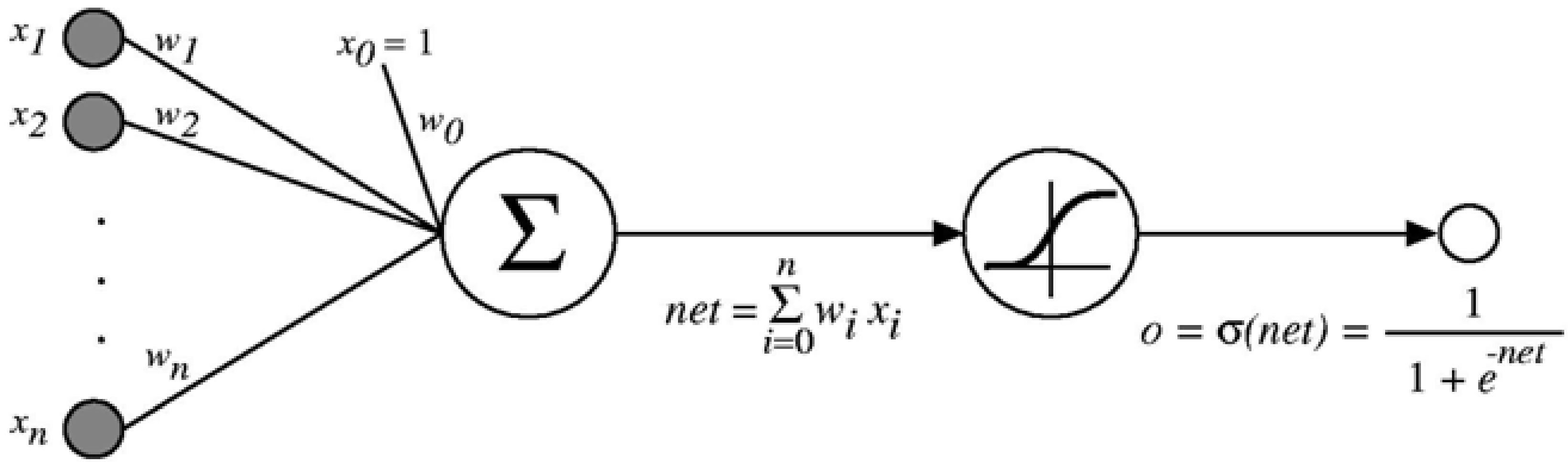
- Initialize each Δw_i to zero.
- For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
  - Input instance $\vec{x}$ to unit and compute output o
  - For each linear unit weight w_i, Do

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

- For each linear unit weight w_i, Do

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

Sigmoid Unit

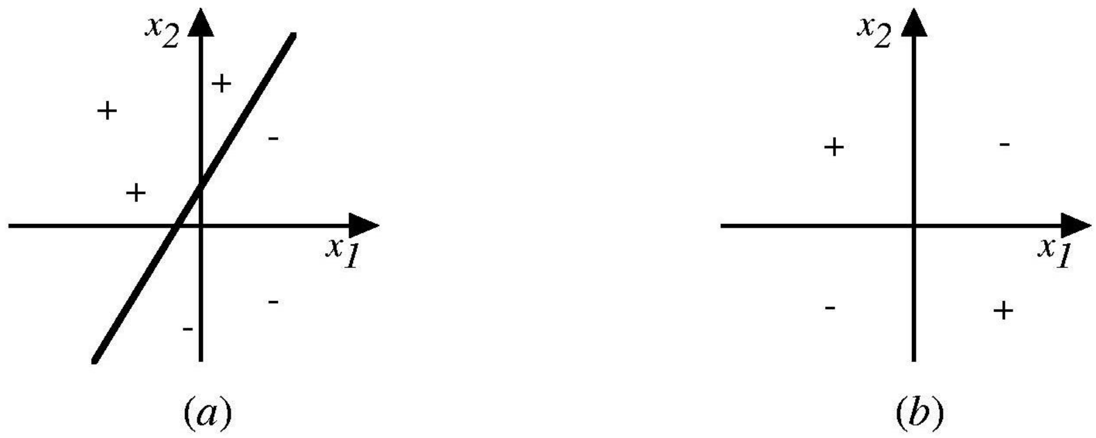


$\sigma(x)$ is the sigmoid function

$$\frac{1}{1+e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

Decision Surface of a Perceptron



Represents some useful functions

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

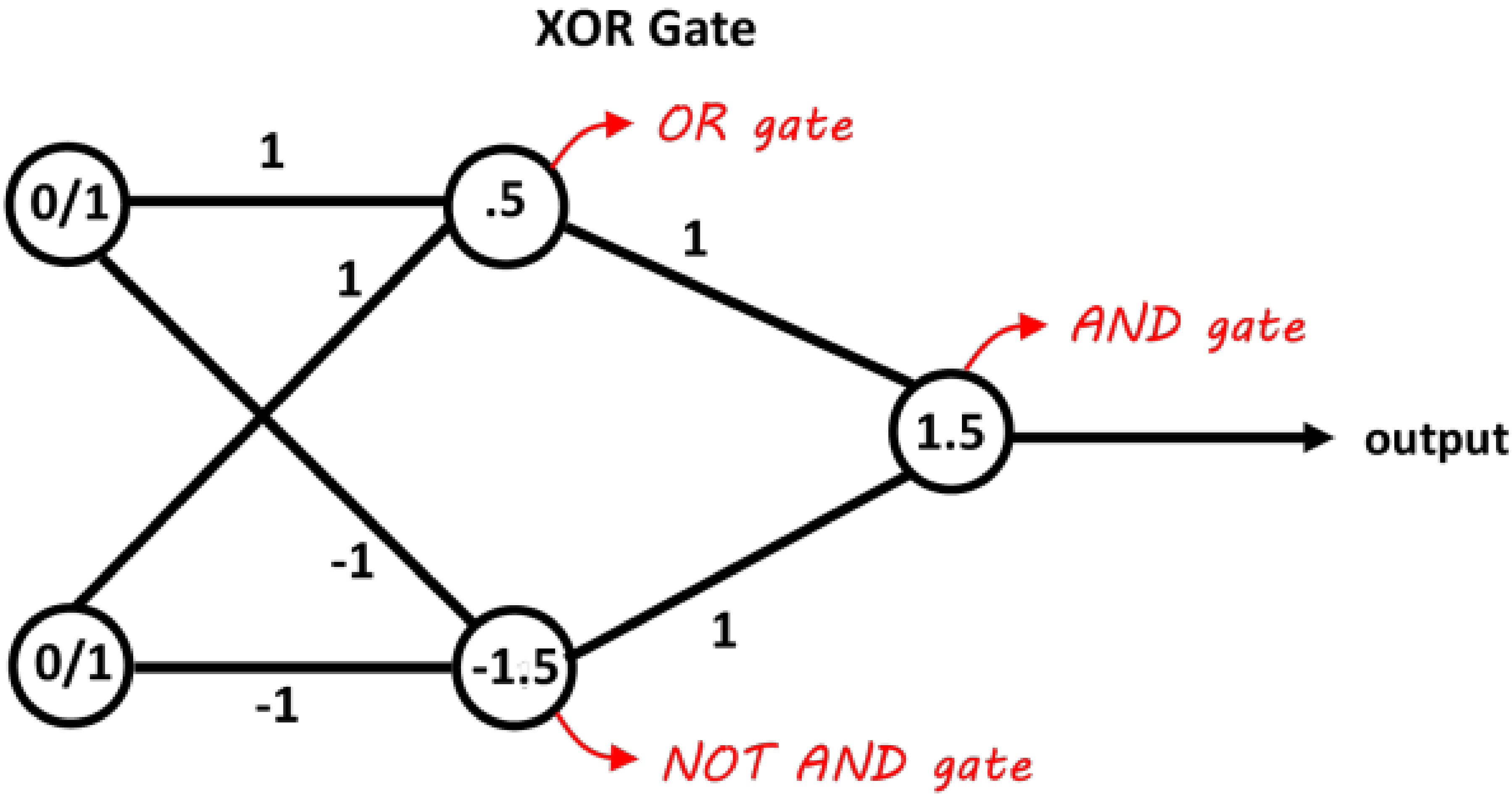
But some functions not representable

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

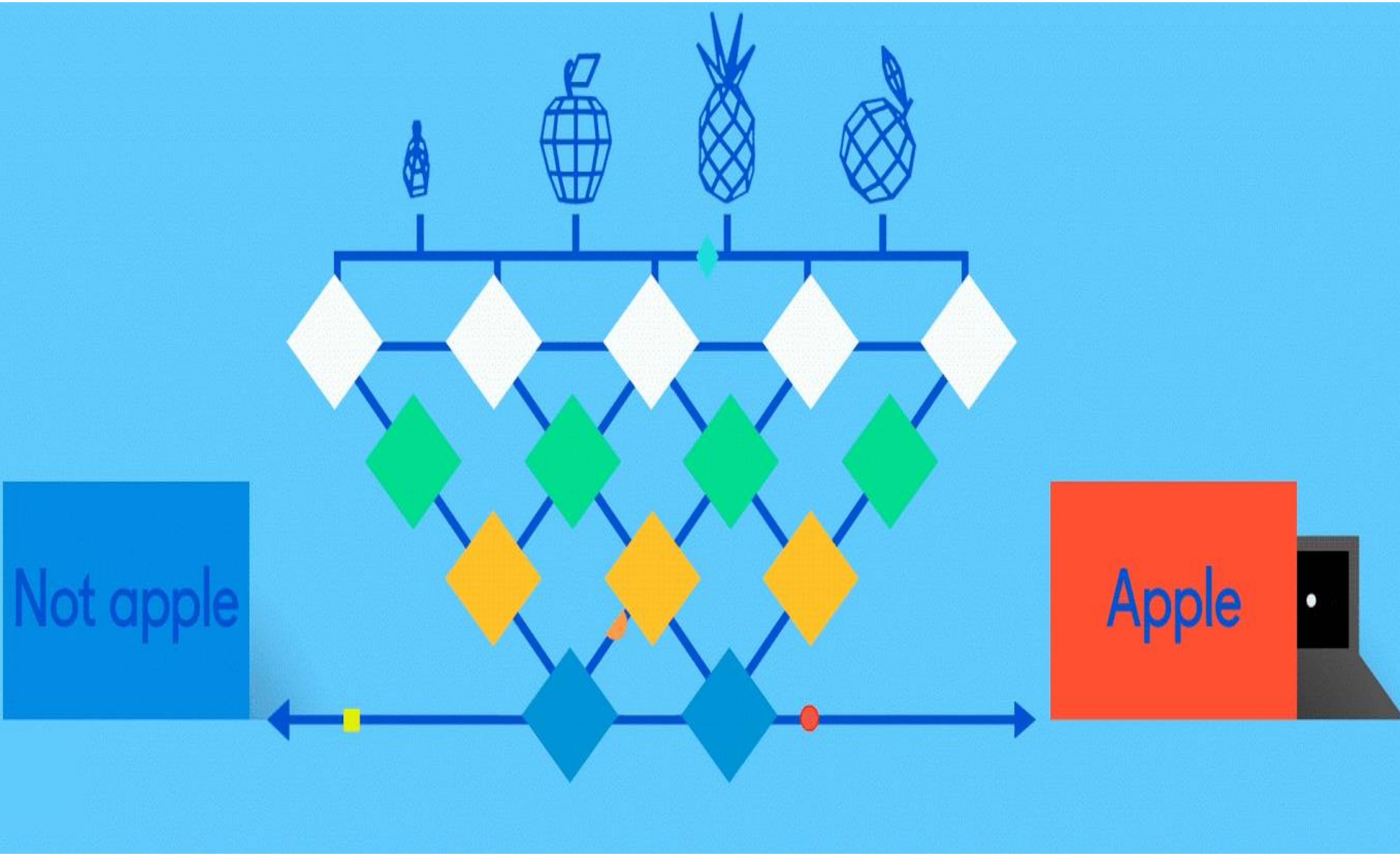
For example $p \oplus q \quad = \quad (p \vee q) \wedge \neg(p \wedge q)$

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

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


XOR Gate
OR gate
AND gate
NOT AND gate
0/1
0/1
.5
-1.5
1.5
1
1
1
1
-1
-1
output

XOR = (PVQ)∧ ~(P∧Q)


Not apple
Apple

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