Neural Networks 1

Shivaram Kalyanakrishnan

Department of Computer Science and Engineering
Indian Institute of Technology Bombay

February 2023

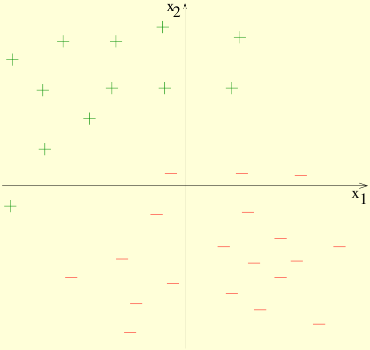
This Lecture

- Non-linear decision boundaries
- Neural networks
- Backpropagation

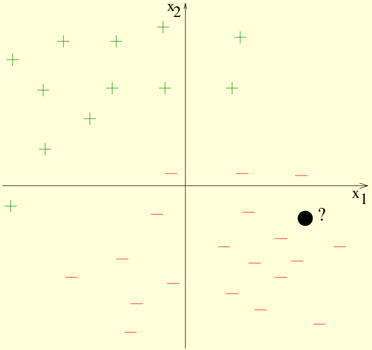
This Lecture

- Non-linear decision boundaries
- Neural networks
- Backpropagation

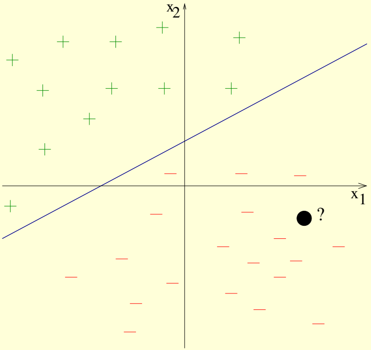
Supervised Learning



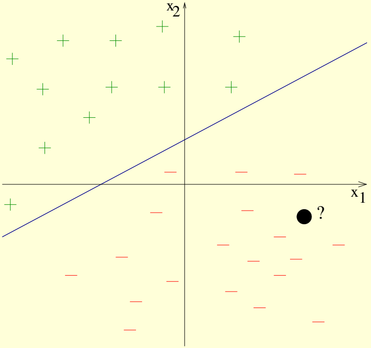
Supervised Learning



Supervised Learning



Supervised Learning



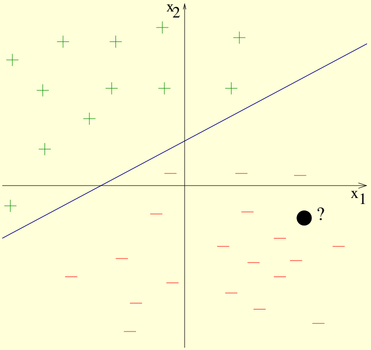
Given labeled training data $\{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$,

Learn a model M such that

for an unseen test point x,

$M(x)$ will be a good prediction of x's (unknown) label y.

Supervised Learning



Given labeled training data $\{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$,

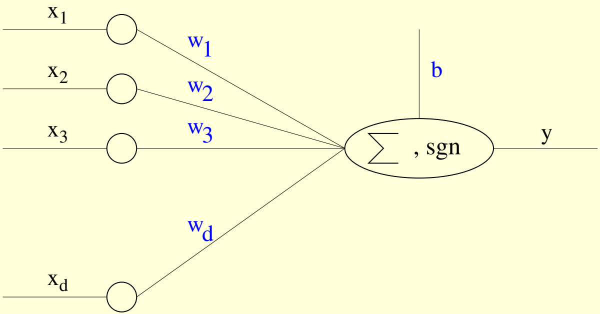
Learn a model M such that

for an unseen test point x,

$M(x)$ will be a good prediction of x's (unknown) label y.

We have seen that if the classes are linearly separable, a perceptron can learn a separating hyperplane.

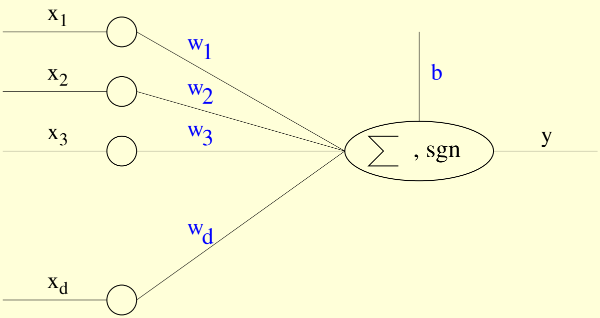
Perceptron Revisited



$$sgn(\alpha) \stackrel{\text{def}}{=} \begin{cases} 0 & \alpha < 0, \\ 1 & \alpha \geq 0. \end{cases}$$

$$y = \text{sgn}(w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b).$$

Perceptron Revisited

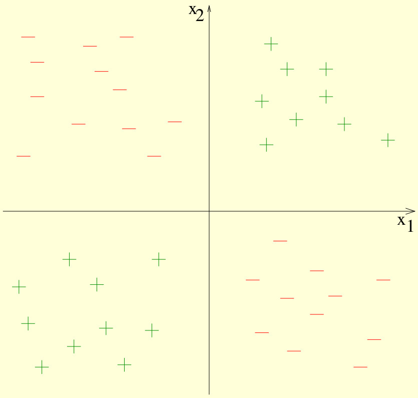


$$sgn(\alpha) \stackrel{\text{def}}{=} \begin{cases} 0 & \alpha < 0, \\ 1 & \alpha \geq 0. \end{cases}$$

$$y = \text{sgn}(w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b).$$

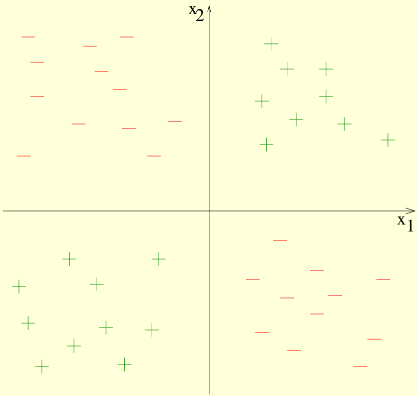
$w = (w_1, w_2, \ldots, w_d)$ and b are parameters learned from data.

The XOR Problem



Minsky and Papert, 1972.

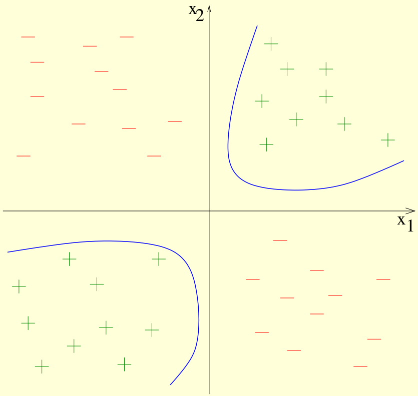
The XOR Problem



Minsky and Papert, 1972.

Can we learn accurate predictors from data that is not linearly separable?

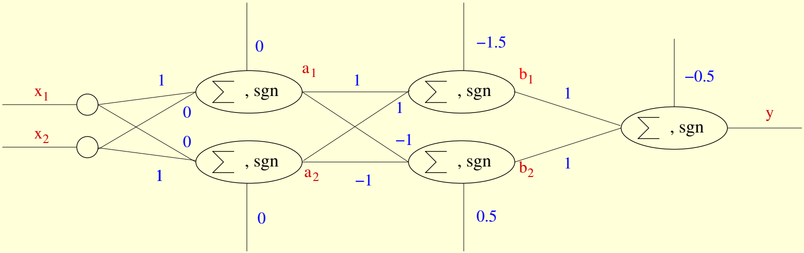
The XOR Problem



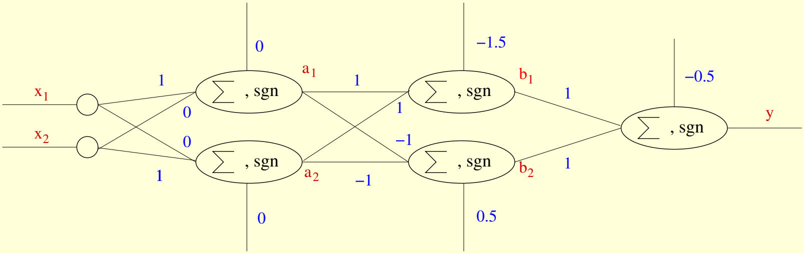
Minsky and Papert, 1972.

Can we learn accurate predictors from data that is not linearly separable?

Idea: Combine Multiple Perceptrons!

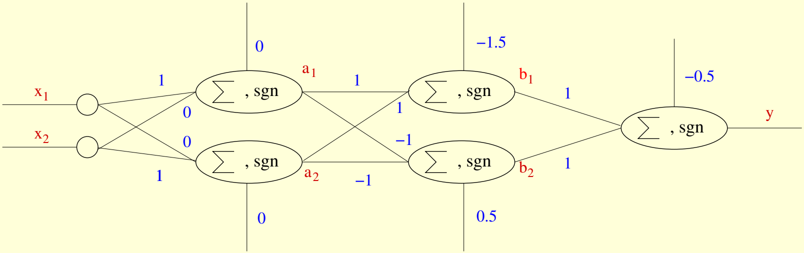


Idea: Combine Multiple Perceptrons!



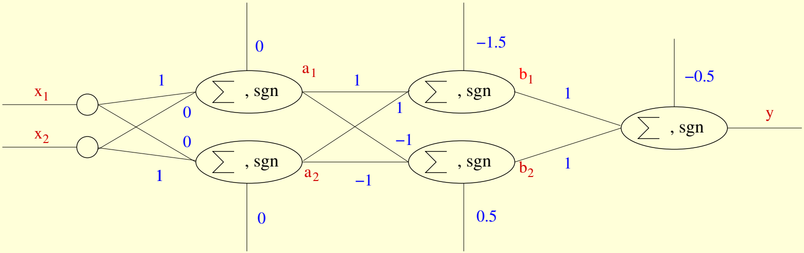
$$a_1 = \text{sign}(x_1).$$

Idea: Combine Multiple Perceptrons!



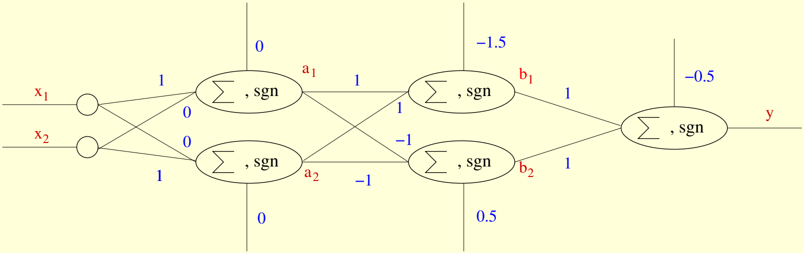
$$a_1 = \text{sign}(x_1).$$
$$a_2 = \text{sign}(x_2).$$

Idea: Combine Multiple Perceptrons!



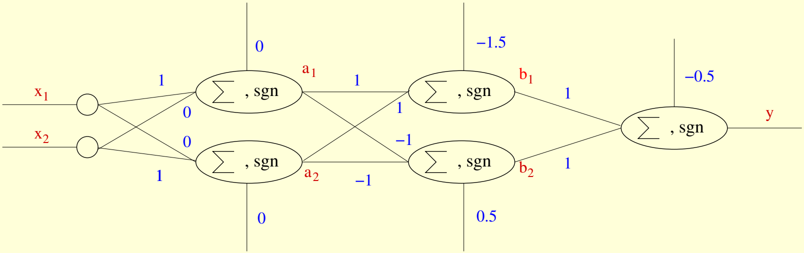
$$a_1 = \text{sign}(x_1).$$
$$a_2 = \text{sign}(x_2).$$
$$b_1 = a_1 \wedge a_2.$$

Idea: Combine Multiple Perceptrons!



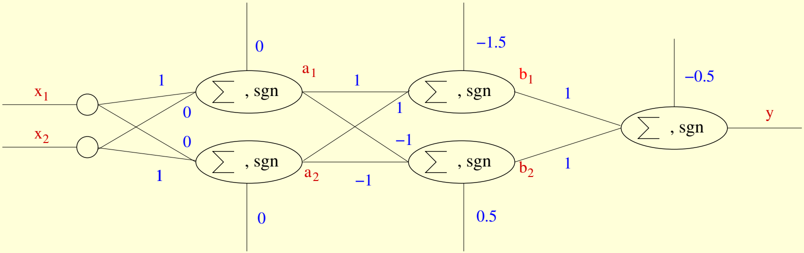
$$a_1 = \text{sign}(x_1).$$
$$a_2 = \text{sign}(x_2).$$
$$b_1 = a_1 \wedge a_2.$$
$$b_2 = \neg a_1 \wedge \neg a_2.$$

Idea: Combine Multiple Perceptrons!



$$a_1 = \text{sign}(x_1).$$
$$a_2 = \text{sign}(x_2).$$
$$b_1 = a_1 \wedge a_2.$$
$$b_2 = \neg a_1 \wedge \neg a_2.$$
$$y = b_1 \vee b_2.$$

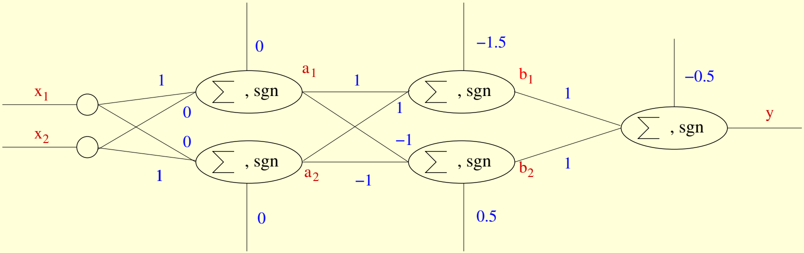
Idea: Combine Multiple Perceptrons!



$$
\begin{aligned}
a_1 &= \text{sign}(x_1).\\
a_2 &= \text{sign}(x_2).\\
b_1 &= a_1 \wedge a_2.\\
b_2 &= \neg a_1 \wedge \neg a_2.\\
y &= b_1 \vee b_2.
\end{aligned}
$$

We solved the XOR problem! What's the catch?

Idea: Combine Multiple Perceptrons!

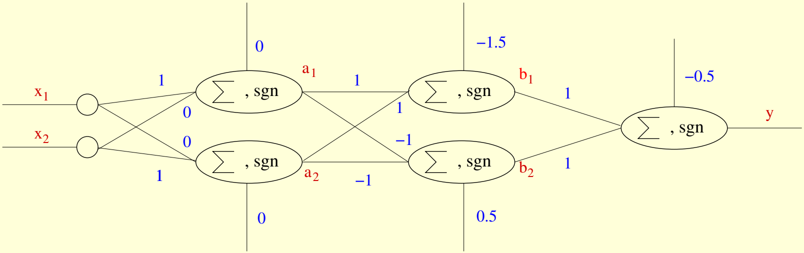


$$
\begin{aligned}
a_1 &= \mathrm{sign}(x_1).\\
a_2 &= \mathrm{sign}(x_2).\\
b_1 &= a_1 \wedge a_2.\\
b_2 &= \neg a_1 \wedge \neg a_2.\\
y &= b_1 \vee b_2.
\end{aligned}
$$

We solved the XOR problem! What's the catch?

We don't know how to learn these weights for this network of perceptrons.

Idea: Combine Multiple Perceptrons!



$$a_1 = \text{sign}(x_1).$$
$$a_2 = \text{sign}(x_2).$$
$$b_1 = a_1 \wedge a_2.$$
$$b_2 = \neg a_1 \wedge \neg a_2.$$
$$y = b_1 \vee b_2.$$

We solved the XOR problem! What's the catch?

We don't know how to learn these weights for this network of perceptrons.

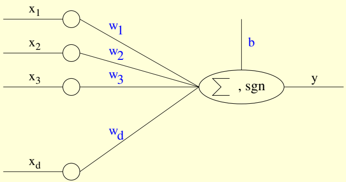
But we'll make an effort . . . .

This Lecture

- Non-linear decision boundaries
- Neural networks
- Backpropagation

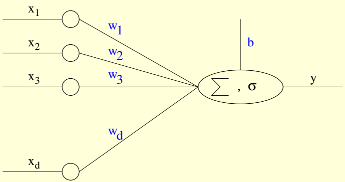
Artificial Neuron

Perceptron

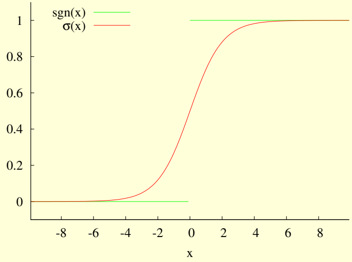


$$sgn(x) \stackrel{\text{def}}{=} \begin{cases} 0 & x < 0, \\ 1 & x \geq 0. \end{cases}$$

Artificial neuron



$$\sigma(x) \stackrel{\text{def}}{=} \frac{1}{1+\exp(-x)}.$$



Activation Function

- $\sigma()$ is called an activation function (input any real number; output in $[0, 1]$).

Activation Function

- $\sigma()$ is called an activation function (input any real number; output in $[0, 1]$).
- There is a subtle but important difference between $\mathrm{sgn}()$ and $\sigma()$? What is it?

Activation Function

- $\sigma()$ is called an activation function (input any real number; output in $[0, 1]$).
- There is a subtle but important difference between sgn() and $\sigma()$? What is it?
- $\sigma()$ is differentiable, while sgn() is not.

Activation Function

- $\sigma()$ is called an activation function (input any real number; output in $[0, 1]$).
- There is a subtle but important difference between sgn() and $\sigma()$? What is it?
- $\sigma()$ is differentiable, while sgn() is not.
  Since $\sigma(x) = \frac{1}{1+e^{-x}}$, we get

$$\sigma'(x) = \frac{d}{dx}(\sigma(x)) = \frac{-e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)).$$

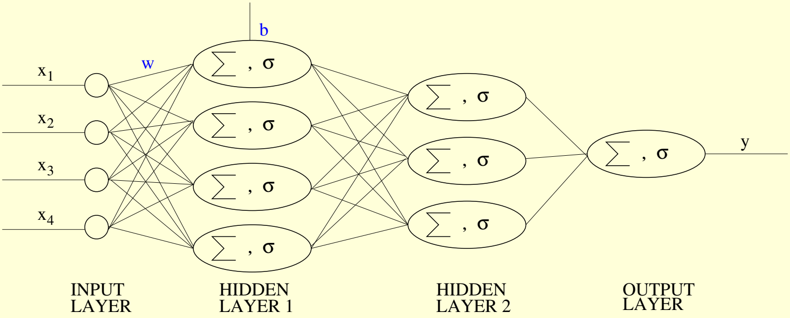
Activation Function

- $\sigma()$ is called an activation function (input any real number; output in $[0, 1]$).
- There is a subtle but important difference between sgn() and $\sigma()$? What is it?
- $\sigma()$ is differentiable, while sgn() is not.
  Since $\sigma(x) = \frac{1}{1+e^{-x}}$, we get

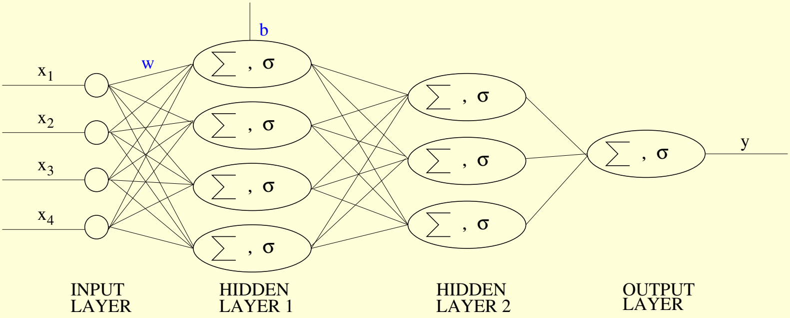
$$\sigma'(x) = \frac{d}{dx}\left(\sigma(x)\right) = \frac{-e^{-x}}{(1+e^{-x})^2} = \sigma(x)(1-\sigma(x)).$$

- There are many other choices of activation function.

Artificial Neural Networks



Artificial Neural Networks



Artificial neural networks are Universal Approximators.

For any function on a finite data set, there exists a single-hidden-layer neural network that fits it exactly (Hornik *et al.*, 1989).

This Lecture

- Non-linear decision boundaries
- Neural networks
- Backpropagation

Backpropagation Algorithm (Rumelhart *et al.*, 1986)

We are given a training data set $\{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$.

Let us start with some initial weights $\mathbf{w}$.

Backpropagation Algorithm (Rumelhart *et al.*, 1986)

We are given a training data set $\{(x^1, y^1), (x^2, y^2), \dots, (x^n, y^n)\}$.
Let us start with some initial weights $\mathbf{w}$.

For each data point r,

the true label is y^r,

the prediction is $p^r(\mathbf{w})$; and

thus, the error is $(y^r - p^r(\mathbf{w}))^2$.

The aggregate error $E(\mathbf{w})$ is $\sum_{r=1}^{n}(y^r - p^r(\mathbf{w}))^2$.
We move a step in the direction minimising error:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w}),$$

and iterate until convergence.

Backpropagation Algorithm (Rumelhart *et al.*, 1986)

We are given a training data set $\{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}$.
Let us start with some initial weights $\mathbf{w}$.

For each data point r,

the true label is y^r,

the prediction is $p^r(\mathbf{w})$; and

thus, the error is $(y^r - p^r(\mathbf{w}))^2$.

The aggregate error $E(\mathbf{w})$ is $\sum_{r=1}^{n}(y^r - p^r(\mathbf{w}))^2$.
We move a step in the direction minimising error:

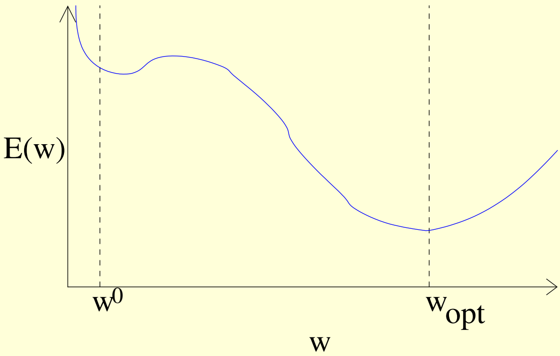
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} E(\mathbf{w}),$$

and iterate until convergence.

For a given neural network, $\nabla_{\mathbf{w}} E(\mathbf{w})$ can be easily computed (coming up).

Convergence of Backprop

Backprop will converge to a local minimum: it need not find $w_{\text{opt}} = \text{argmin}_w \, E(\mathbf{w})$.



References

- Chapter 10, **A Course in Machine Learning**, Hal Daumé III. Available on-line at `http://ciml.info/`.