NEURAL NETWORKS

Biological neurons

Neurons are interconnected nerve cells

acting as logic gates

Signals arrive

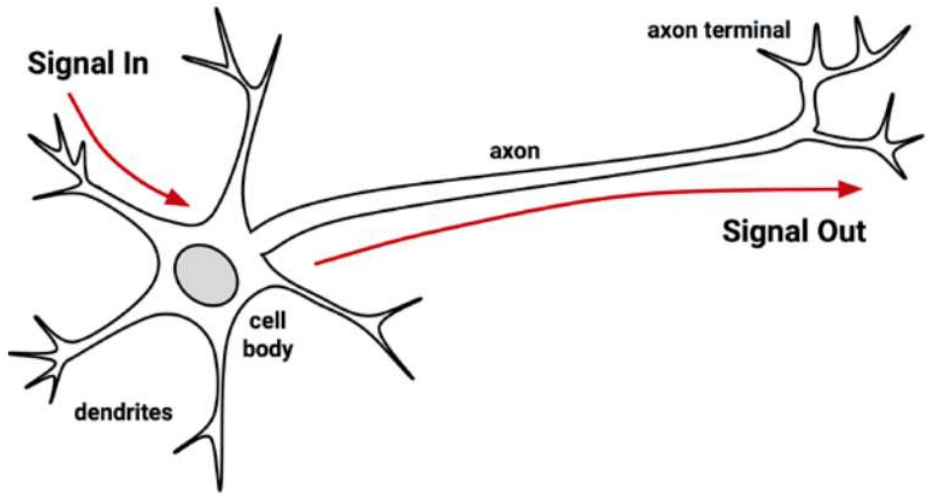
and accumulate

If accumulation

exceeds a

threshold

an output signal is passed on



Artificial neurons

input signals $x_1, \ldots x_k$

output signal y

input signals are

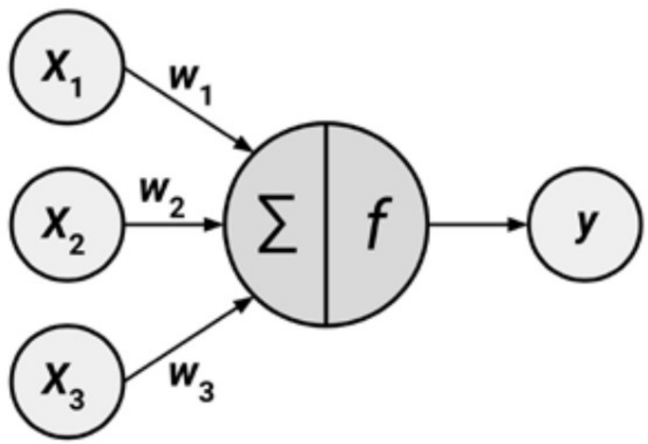
weighted and

summed by the cell

the signal is passed on

using an activation function f

Directed Network



Artificial Neural Network (ANN)

ANN models

the relationship between

a set of input signals

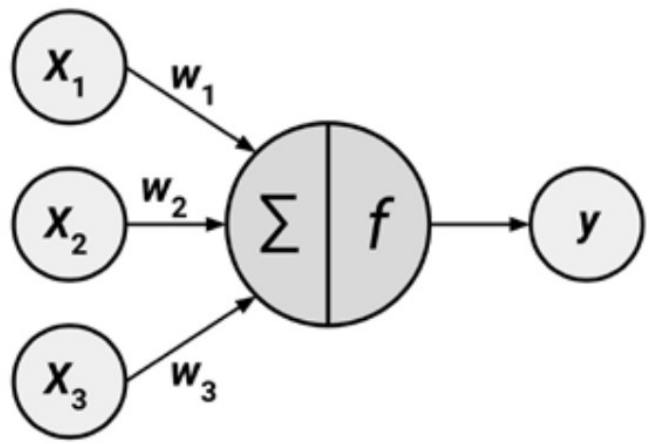
and an output signal

resembling our

understanding of

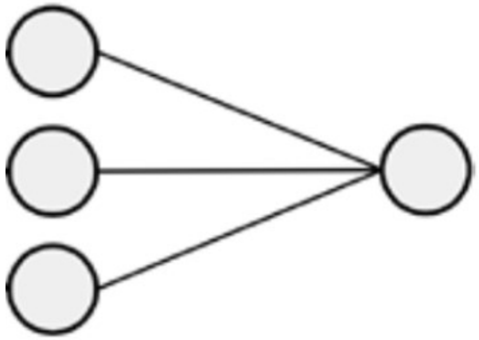
how cells respond to stimuli

from sensory inputs



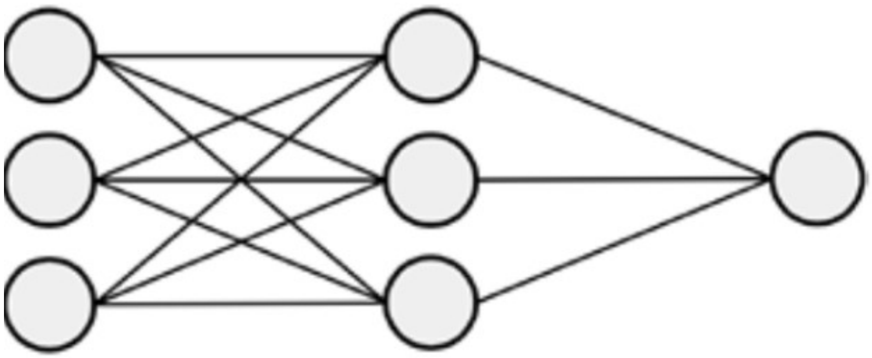
ANN Types

- Single Input Layer ANN



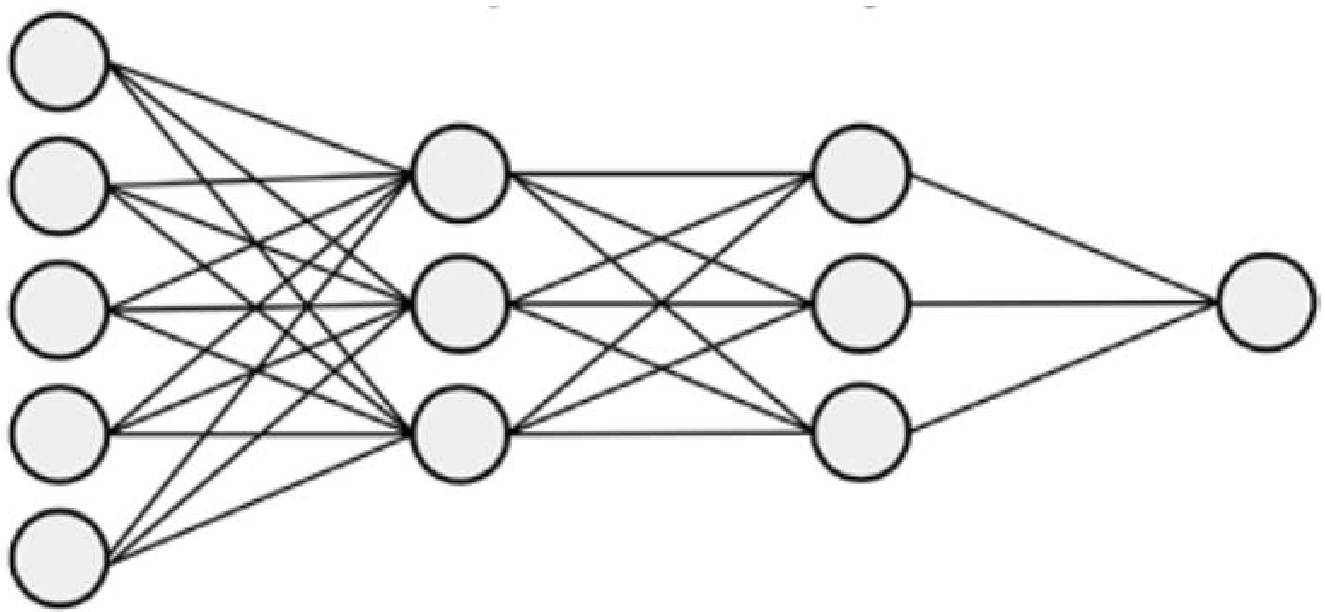
ANN Types

- One input, one hidden layer



ANN Types

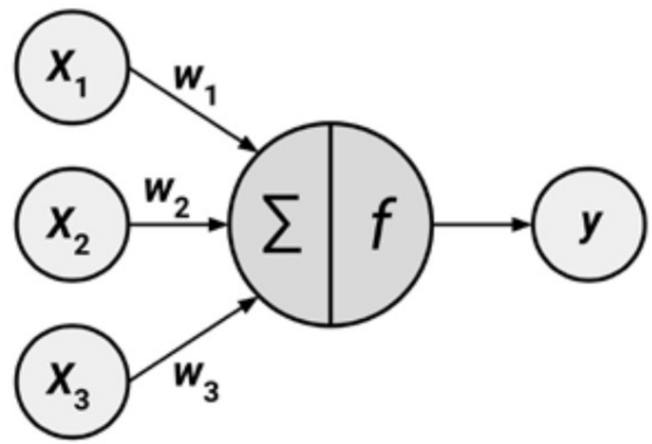
- Multiple hidden layer ANN



Perceptron

Single layer ANN

receives inputs



Perceptron

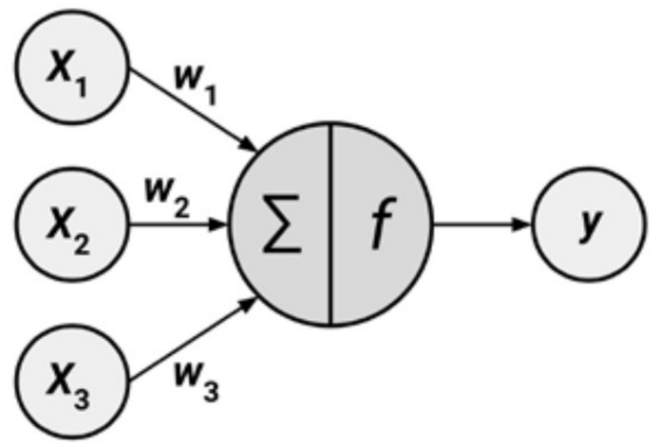
Single layer ANN

receives inputs

combines

inputs with weights

obtain net input



Perceptron

Single layer ANN

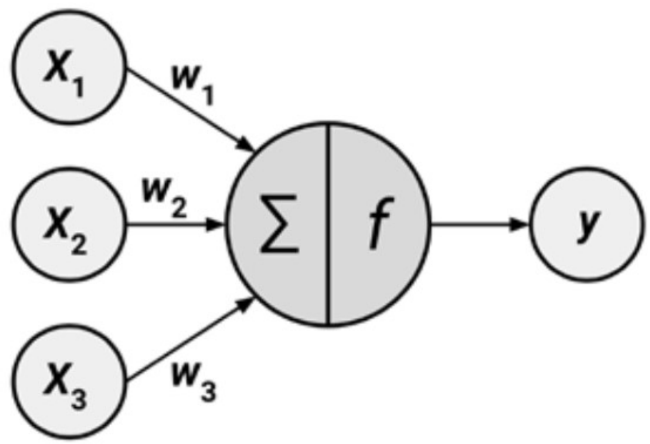
receives inputs

combines

inputs with weights

obtain net input

process it



Perceptron

Single layer ANN

receives inputs

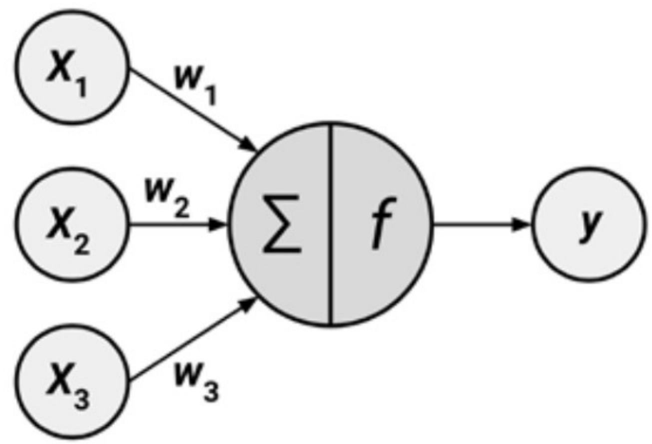
combines

inputs with weights

obtain net input

process it

generate output



Perceptron

Single layer ANN

receives inputs

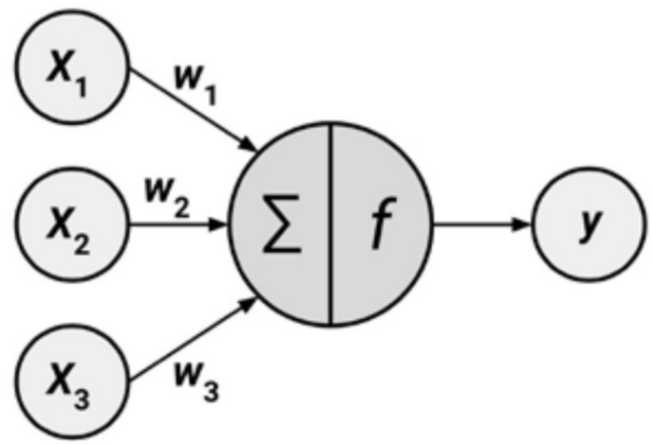
combines

inputs with weights

obtain net input

process it

generate output



Perceptron

Single layer ANN

receives inputs

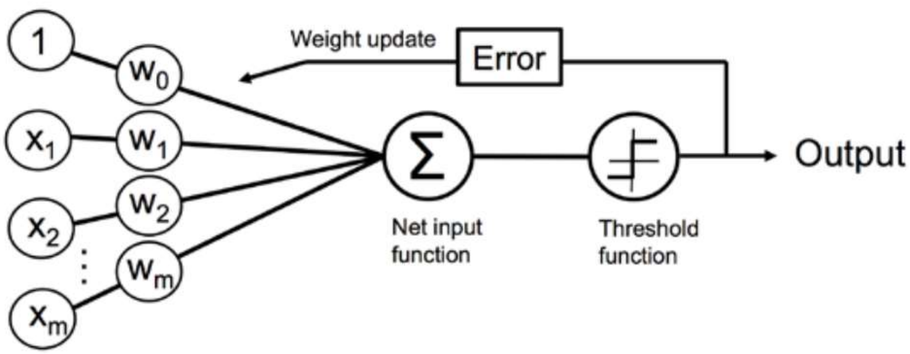
combines

inputs with weights

obtain net input

process it

generate output



Perceptron

Single layer ANN

receives inputs

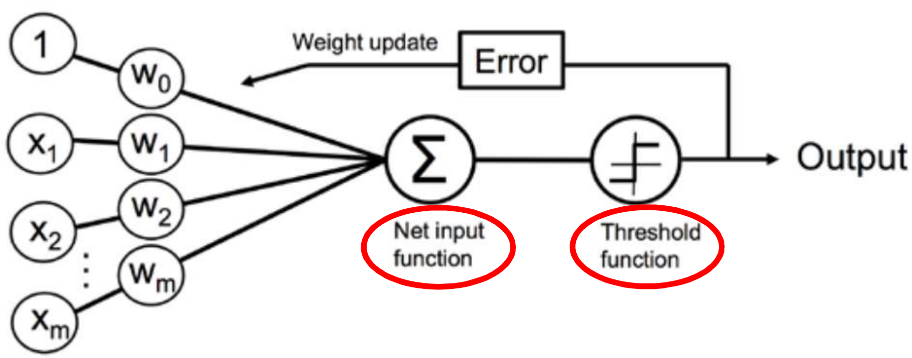
combines

inputs with weights

obtain net input

process it

generate output



Perceptron

net input function $z = w_1 x_1 + \ldots + w_m x_m$

decision function $\phi(z) = \begin{cases} 1 & if\ z \geq \theta \\ -1 & otherwise \end{cases}$

Perceptron

net input function $$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m$$

$$w_0 = -\theta \text{ and } x_0 = 1$$

decision function $$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

Perceptron

net input function $z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$

$w_0 = -\theta$ and $x_0 = 1$

decision function
$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

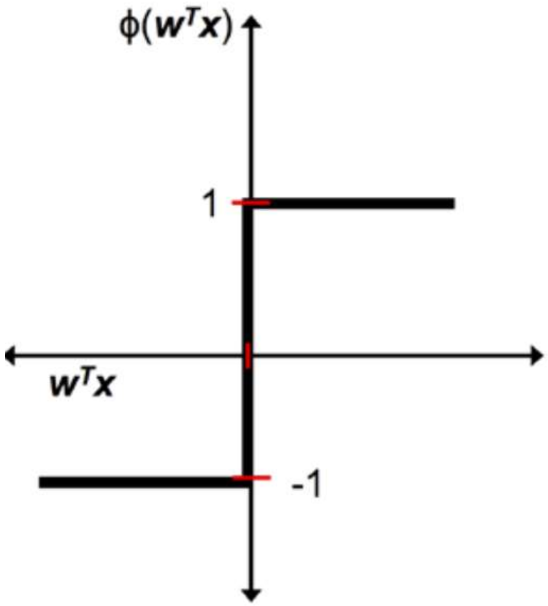
18

Perceptron

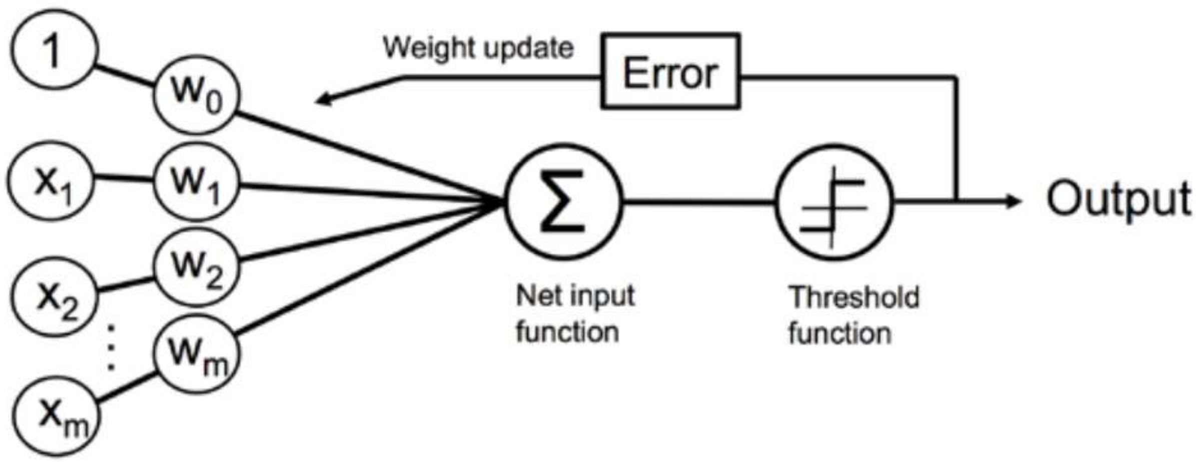
threshold function $\phi(z)$

$$z = \boldsymbol{w}^T \boldsymbol{x}$$

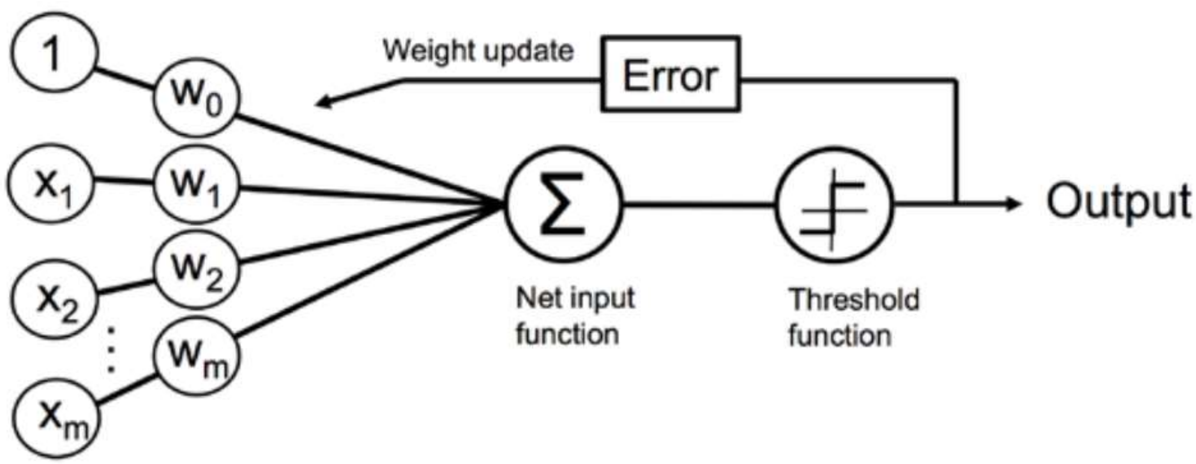
$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$



Perceptron


1
w_0
x_1
w_1
x_2
w_2
x_m
w_m
Weight update
Error
Σ
Net input function
Threshold function
Output

Perceptron learning rule



Perceptron learning rule

- Randomly initialize the weights $w_1, \ldots, w_k$
- For each row $i = 1, \ldots, n$

  find $\hat{y} = \phi(z)$

  find the error $(y - \hat{y})$

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda(y_i - \hat{y}_i)\, x_{ij}$

  for $j = 1, \ldots, p$
- Repeat N times

Perceptron learning rule

- Randomly initialize the weights $w_1, \ldots, w_k$
- For each row $i = 1, \ldots, n$

  find $\hat{y} = \phi(z)$

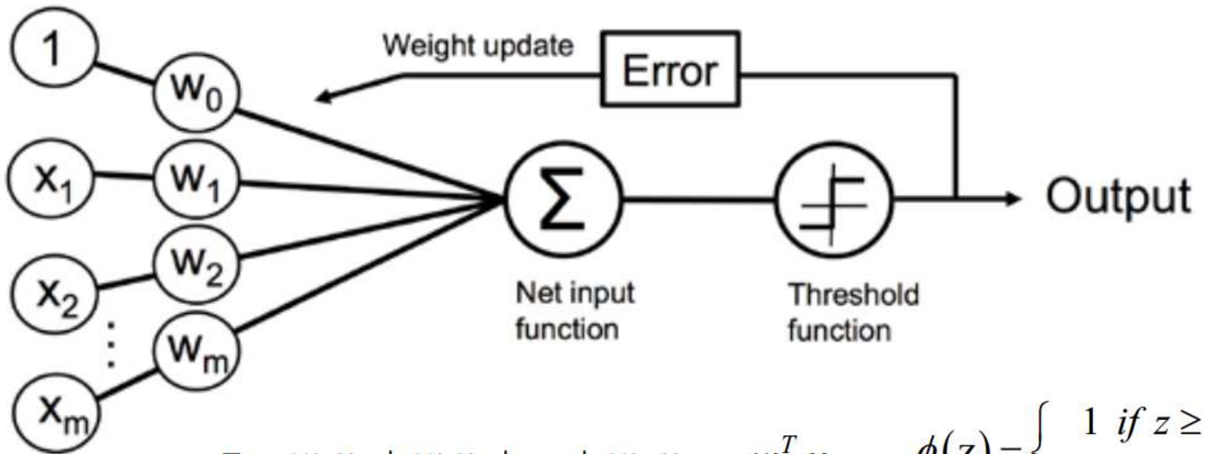
  find the error $(y - \hat{y})$

  update the weights $w_j = w_j + \Delta w_j$

  using $\Delta w_j = \lambda(y_i - \hat{y}_i)\, x_{ij}$

  for $j = 1, \ldots, p$
- Repeat N times

Perceptron learning rule



$$z = w_0 x_0 + w_1 x_1 + \ldots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}$$

$$\phi(z) = \begin{cases} 1 & if\ z \geq 0 \\ -1 & otherwise \end{cases}$$

Notes

- Perceptron works if data is linearly separable
- Decision function = Threshold function
- iterations N = epochs
- $\hat{y} = \phi(z)$

ADAptive LInear NEuron (Adaline)

- Another single-layer NN
- net input *z* is transformed using an Activation function

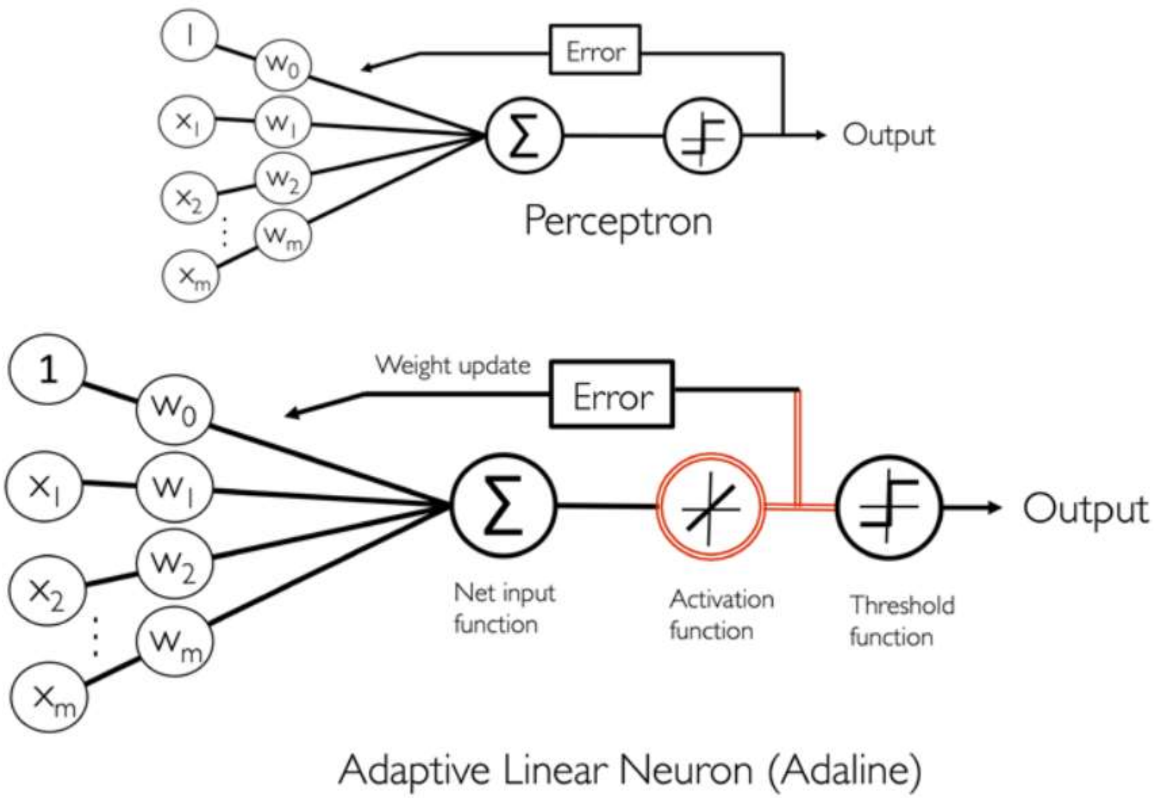
$$z = w_1 x_1 + \ldots + w_m x_m$$

ADAptive LInear NEuron (Adaline)

- Another single-layer NN
- net input *z* is transformed using an Activation function
- For Adaline Activation function is $\phi(z) = z$
- Prediction is given by threshold function

$$\hat{y} = \begin{cases} 1 & \text{if} \quad z > 0 \\ -1 & \text{if} \quad z \leq 0 \end{cases}$$

Adaline vs Perceptron



Adaline stopping condition

- Minimize loss function

$$
\begin{aligned}
J(\boldsymbol{w}) &= \frac{1}{2}\sum_{i=1}^{n}(y_i - \phi(z_i))^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}(y_i - z_i)^2 \\
&= \frac{1}{2}\sum_{i=1}^{n}[y_i - w_1\,x_1 + \cdots + w_p\,x_p]^2
\end{aligned}
$$

quadratic and differentiable

- How to find $w_1, \ldots, w_k$ that minimize the loss?

Adaline stopping condition

loss function $$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}[y_i - w_1 x_1 + \cdots + w_p x_p]^2$$

learning algorithm

- Randomly initialize the weights $w_1, \ldots, w_k$
- Update the weights using gradient descent

$$w_j = w_j + \Delta w_j$$

Gradient descent

loss function $$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{n}[y_i - w_1 x_1 + \cdots + w_p x_p]^2$$

gradient vector $\nabla \boldsymbol{J}(\boldsymbol{w})$ is vector of partial derivatives

- Randomly initialize the weights $w_1, \ldots, w_k$
- Update the weights using gradient descent

$$w_j = w_j + \Delta w_j$$

- where $\Delta \boldsymbol{w} = -\lambda \nabla \boldsymbol{J}(\boldsymbol{w})$

Gradient descent

- The partial derivatives are

$$\frac{\partial J}{\partial w_j} = -\sum_{i=1}^{n} (y_i - \phi(z_i))\, x_{ij}$$

- The weight change for feature *j* is

$$\begin{aligned}\Delta w_j &= -\lambda \frac{\partial J}{\partial w_j} \\ &= \lambda \sum_{i=1}^{n} (y_i - \phi(z_i))\, x_{ij}\end{aligned}$$

Notes

- Adaline iterates until loss function is minimized

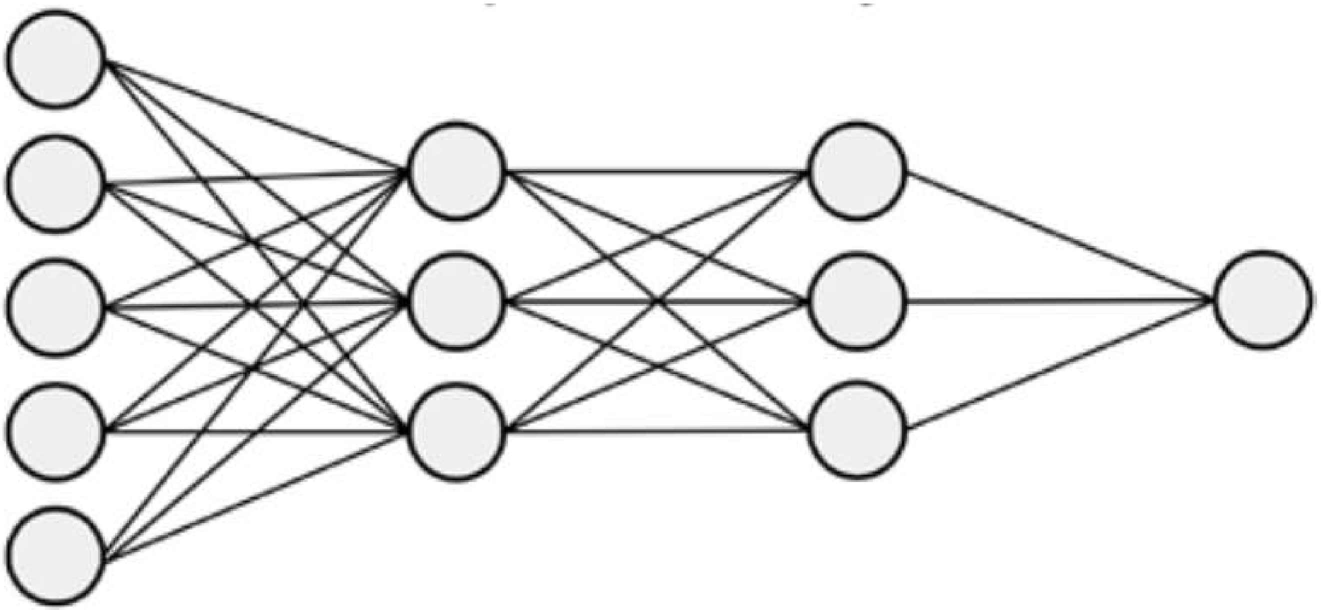
  (until convergence)

- Logistic regression is Adaline network with activation

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

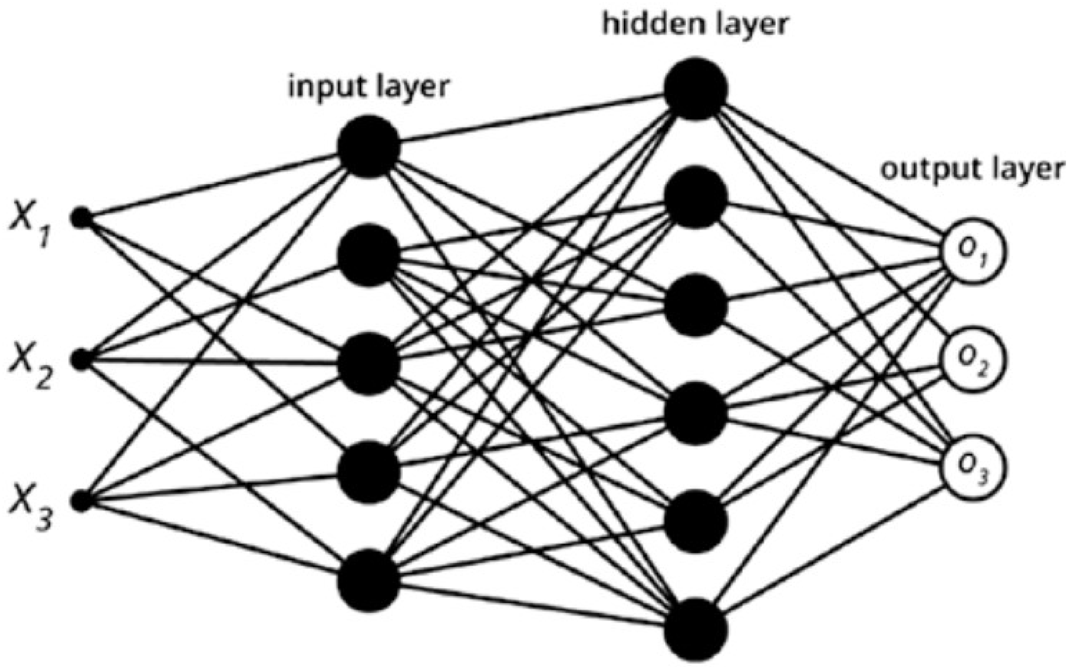
Multilayer NN

- One-node output layer



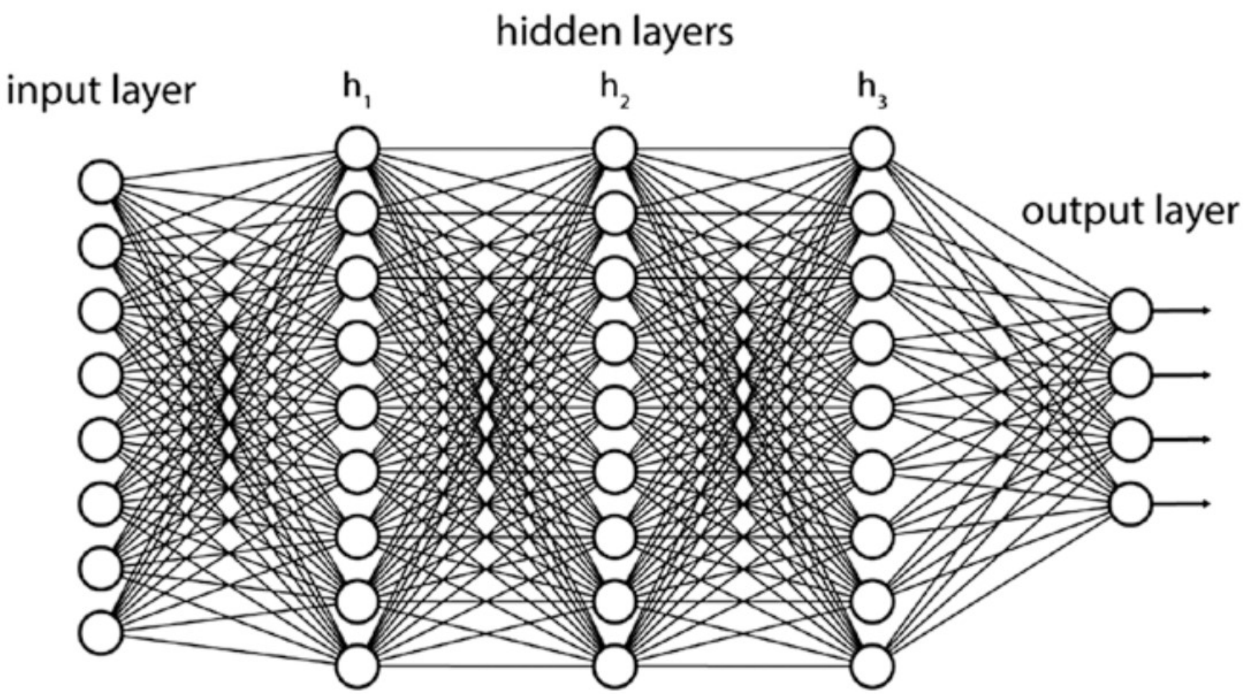
Multilayer NN

- Three-node output layer



Deep Neural Network

- Three hidden and a four-node output layer



Notes

- NN layer is called Dense if all nodes are connected with neighbor layers
- NN is called Multilayer Perceptron (MLP) if all layers are dense
- NN is called Deep if the number of hidden layers is large (usually 8 or more)

Notes

- For small datasets use a small number of hidden layers otherwise risk of overfitting
- May consider increasing the number of hidden layers with the dataset size
- Fine tune number of epochs, number of layers, learning rate, etc.

Other Neural Networks

- Convolutional Neural Networks (convnets)
- Recurrent Neural Networks