- Artificial Neural Networks: a family of models that try to imitate biological neural networks in human and animals' brain, are used to estimate or approximate unknown functions that have a large number of inputs. (Wikipedia)
- Earliest models of ANN were proposed as computational tools by McCulloch and Pitts in 1940s.
- In 1958, Rosenblatt proposed the first algorithm based on perceptron. However, in 1960s it was shown that this simple model could not compute some very simple functions.
- Since 1970s, several tools were developed to solve the problems with neural network models. With the advance in computation technology, ANN has become the state-of-the-art methods for many supervised and unsupervised learning problems.

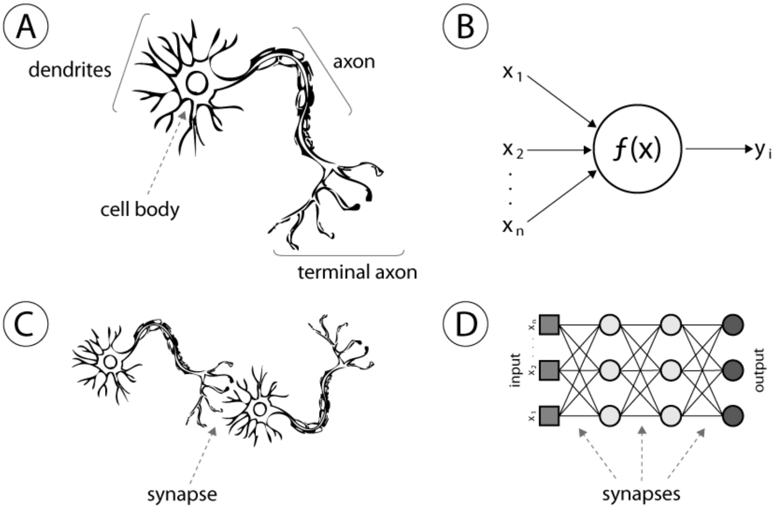


Figure: www.intechopen.com

- Computer: fast computation, few mistakes, very efficient with "linear" tasks.
- Brain: robust, flexible, can deal with complex nonlinear tasks. For example: face recognition task.
- Brain "computing unit": neurons, 10^{10} to 10^{12} unites highly interconnected and parallel.
- Computer computing unit is faster than that of human, but not energy efficient.
- Computer does not get tired quickly.

- ANN is an attempt to emulate the brain to deal with the problem of nonlinearity: the relationship between input data and output is nonlinear.
- Via input - output mapping (approximating a function mapping input space to outspace), ANNs have the learning capabilities with a teacher (supervised) or without a teacher (unsupervised).
- Learning parameters can be adjusted with respect to the surrounding environments (input data).
- Very flexible, can be used to do a wide variety of tasks: classification, regression, self-driving car, chess playing machine, go playing machine.

- Components of ANN: Neuron: Computing units that take n inputs: $x_1, x_2, \cdots, x_n$ and calculate the weighted sum $h = \sum_{i=1}^{n} w_i x_i$. After that the weighted sum is processed via an activation function (transfer) into an output y that decides whether to fire that neuron.
- Example: $h = \sum_{i=1}^{n} w_i x_i$, $y = \begin{cases} 1 & \text{if } h > 0 \\ 0(-1) & \text{otherwise} \end{cases}$
- In this case the activation function is the thresh-hold function.
- Perceptron models are built by layers of neuron connected in the "feed-forward" manner, as in one layer of neurons is connected to the layers after it, but not the other way around.

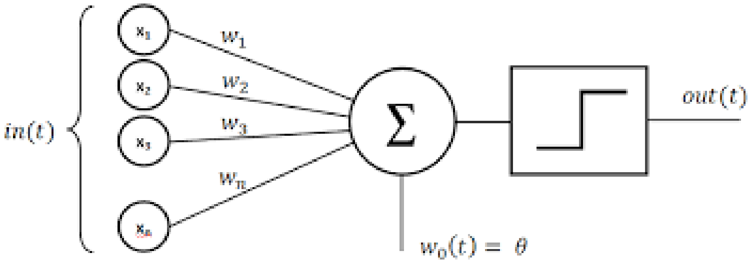


Figure: Slides shared by Hannes Hapke

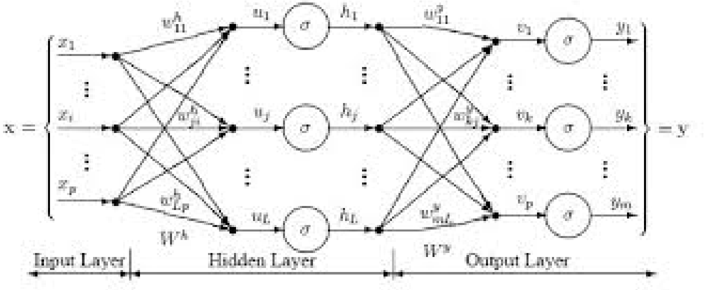


Figure: www.dtreg.com

Future of neural networks:

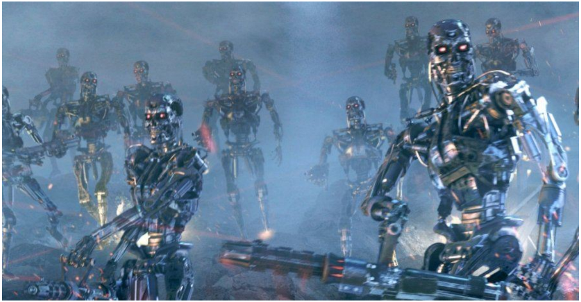


Figure: http://www.hangthebankers.com

- One layer feed forward network: N inputs and an output layer, with no hidden layer.
- For an observation i with p inputs, the output will be calculated as:

$$\hat{y}_i = g(w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip}),$$

  for g is some activation function.
- We can add the bias term w_0:

$$\hat{y}_i = g(w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip}),$$

- It is desired that the output $\hat{y}_i$ and the actual response y_i to be as close as possible.

- Simple perceptron models for classification: Input x_i, y_i where $x_i \in \mathcal{R}^p$, $y_i \in \{-1, 1\}$.
- The weight vector $w_1, w_2, \cdots, w_p$ correspond to each measurement of an observation.
- Activation function g(.): as: g(x)=1 if $\geq$ 0 and g(x)=-1, otherwise. g(x)=sign(x).
- We want $\hat{y}_i$ and y_i to be as close as possible, same as saying: $sign(w_1 x_{i1} + \cdots w_p x_{ip}) = sign(\langle w, x_i \rangle) = y_i$
- Weight w is chosen so that the projection of x_i on w has the same sign as y_i.

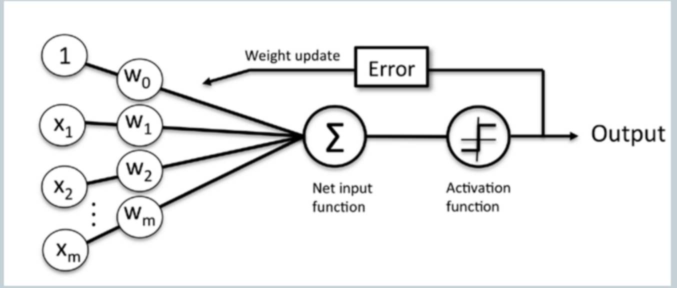
- The boundary between negative and positve projection is the hyperplane: $\langle w, x \rangle = 0$.
- Alternate condition: $\langle w, x_i \rangle y_i > 0$.
- The problem can only be solved when the data is linearly separable, i.e there exists such a plane that separate the two class 1 and -1.
- For now we consider a learning algorithm for separable data.

- A simple algorithm: Start with any separating hyperplane w, go through all the observations one by one, for each observation ask whether the output is the same as the response: $y_i = \hat{y}_i$. If it is the case, do not change w and move on to the next observation. If not, update vector w:

$$w = w + 2\nu(y_i - \hat{y}_i)x_i,$$

where ν is a small constant called learning rate.
- This procedure is repeated untill all observations are correctly classified.
- The equation is called perceptron learning rule. Intuitively, each iteration tries to adjust the weight vector w to correct the error that is made.

- It can be shown that this algorithm converges (one way of saying it will give you a decent solution) after a finite number of steps if the data is linearly separable and the learning rate is small enough.


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

- Example: Suppose we can predict the stock price of a company X by a model

$$Price(x) = 10 + 4x_1 - 2x_2.$$

- Suppose the current price is 80, we are going to specify a perceptron model that output 1 if the stock is predicted to go up or unchanged, and return -1 if the stock is predicted to go down. There are two possible outcomes:
    1. $10 + 4x_1 - 2x_2 < 80$ (price will go down).
    2. $10 + 4x_1 - 2x_2 \geq 80$ (price will go up).
- We can represent the prediction using an activation function: g(z)=sign(z) where $z = -70 + 4x_1 - 2x_2$.
- When $x_1 = 25, x_2 = 5$, z=-70 + 4*25-2*t=20, g(z)=1 , predict stock goes up or unchanged.
- When $x1 = 20, x_2 = 10$, z=-70+4*20-2*10=-10, g(z)=-1, predict stock goes down.

- With perceptron model, update the weights when we have observation: $y = 1, x_1 = 25, x_2 = 5$, with $\nu = .1$ From learning rule:
  1. As shown above, z=20, g(z)=1, so $\hat{y} = 1$.
  2. Prediction is correct. w remains unchanged.
- New data: $y = 1, x_1 = 20, x_2 = 10$
  1. As shown above, z=-10, g(z)=-1, so $\hat{y} = -1$.
  2. Update w: $w = w + \nu(y - \hat{y})x = [-70, 4, 2] + 0.1[1, 20, 10]$
  3. w=[69.9,6,-1].

- Adaptive linear neurons (Adaline): proposed by Widrow and Hoff in 1960.
- In the previous model, there is no explit cost function. Adaline model illustrated the concept of defining and minimizing cost function.
- Activation function is identity function: $g(\langle w, x \rangle) = \langle w, x \rangle$
- Define loss functionfor an observation i: $J(w) = \frac{1}{2}(y_i - \langle w, x \rangle)^2$.
- Overall observation loss function: $J(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \langle w, x \rangle)^2$

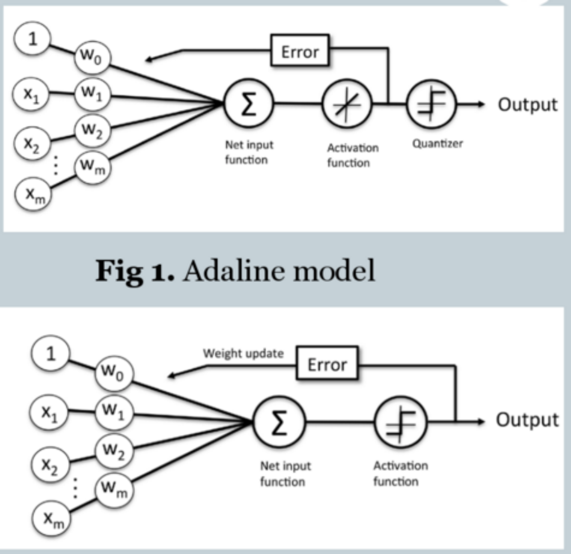


Fig 1. Adaline model

- Use gradient descent method to find the weight that minimize the loss function J(w):

$$w^{k+1} = w^k - \nu \nabla J(w^k)$$

- Example:
- The weight update is calculated with all observations in the training set (as opposed to one-by-one update in perceptron model).