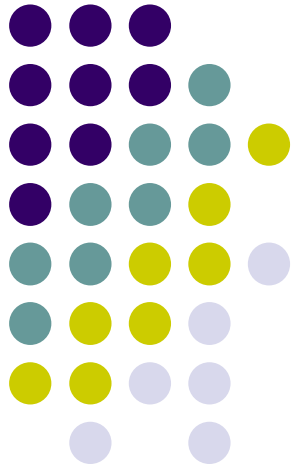
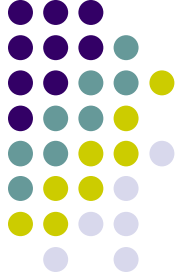
Machine Learning

Neural Networks

Part - 1



Artificial Neural Networks

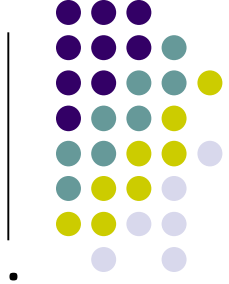


- Computational models inspired by the human brain:
  - Algorithms that try to mimic the brain.
  - Massively parallel, distributed system, made up of simple processing units (neurons)
  - Synaptic connection strengths among neurons are used to store the acquired knowledge.
  - Knowledge is acquired by the network from its environment through a learning process

History

- late-1800's - Neural Networks appear as an analogy to biological systems
- 1960's and 70's – Simple neural networks appear
  - Fall out of favor because the perceptron is not effective by itself, and there were no good algorithms for multilayer nets
- 1986 – Backpropagation algorithm appears
  - Neural Networks have a resurgence in popularity
  - More computationally expensive

Applications of ANNs



- ANNs have been widely used in various domains for:
  - Pattern recognition
  - Function approximation
  - Associative memory

Properties

- Inputs are flexible
  - any real values
  - Highly correlated or independent
- Target function may be discrete-valued, real-valued, or vectors of discrete or real values
  - Outputs are real numbers between 0 and 1
- Resistant to errors in the training data
- Long training time
- Fast evaluation
- The function produced can be difficult for humans to interpret

When to consider neural networks

- Input is high-dimensional discrete or raw-valued
- Output is discrete or real-valued
- Output is a vector of values
- Possibly noisy data
- Form of target function is unknown
- Human readability of the result is not important

Examples:

- Speech phoneme recognition
- Image classification
- Financial prediction

Artificial Neural Network

- An artificial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections.

Artificial Neural Network

- A set of major aspects of a parallel distributed model include:
- a set of processing units (cells).
- a state of activation for every unit, which equivalent to the output of the unit.
- connections between the units. Generally each connection is defined by a weight.
- a propagation rule, which determines the effective input of a unit from its external inputs.
- an activation function, which determines the new level of activation based on the effective input and the current activation.
- an external input for each unit.
- a method for information gathering (the learning rule).
- an environment within which the system must operate, providing input signals and _ if necessary _ error signals.

Computers vs. Neural Networks

"Standard" Computers	Neural Networks
• one CPU	highly parallel processing
•fast processing units	slow processing units
•reliable units	unreliable units
•static infrastructure	dynamic infrastructure

Why Artificial Neural Networks?

- There are two basic reasons why we are interested in building artificial neural networks (ANNs):

- **Technical viewpoint:** Some problems such as character recognition or the prediction of future states of a system require massively parallel and adaptive processing.

- **Biological viewpoint:** ANNs can be used to replicate and simulate components of the human (or animal) brain, thereby giving us insight into natural information processing.

Artificial Neural Networks

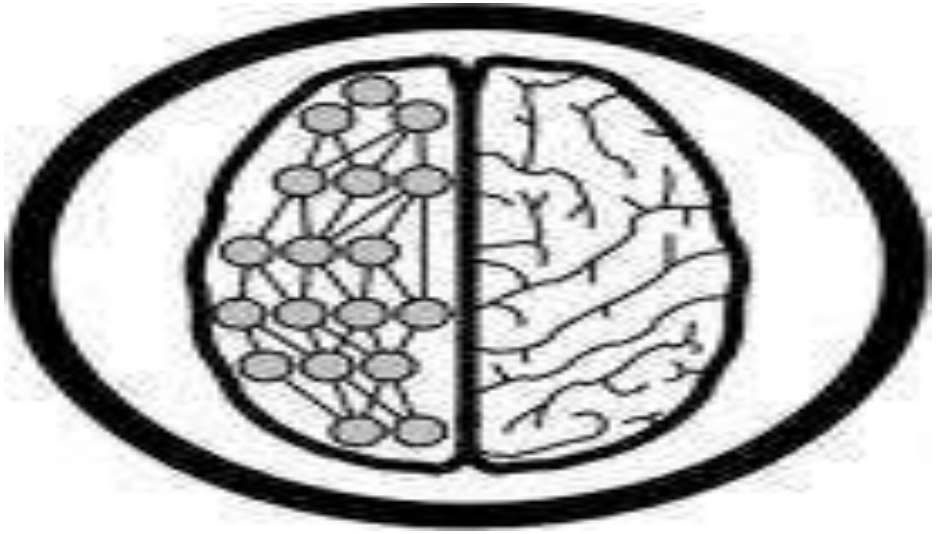
- The “building blocks” of neural networks are the **neurons**.
  - In technical systems, we also refer to them as **units** or **nodes**.
- Basically, each neuron
  - receives **input** from many other neurons.
  - changes its internal state (**activation**) based on the current input.
  - sends **one output signal** to many other neurons, possibly including its input neurons (recurrent network).

Artificial Neural Networks

- Information is transmitted as a series of electric impulses, so-called **spikes**.
- The **frequency** and **phase** of these spikes encodes the information.
- In biological systems, one neuron can be connected to as many as **10,000** other neurons.
- Usually, a neuron receives its information from other neurons in a confined area, its so-called **receptive field**.

How do ANNs work?

- An artificial neural network (ANN) is either a hardware implementation or a computer program which strives to simulate the information processing capabilities of its biological exemplar. ANNs are typically composed of a great number of interconnected artificial neurons. The artificial neurons are simplified models of their biological counterparts.
- ANN is a technique for solving problems by constructing software that works like our brains.

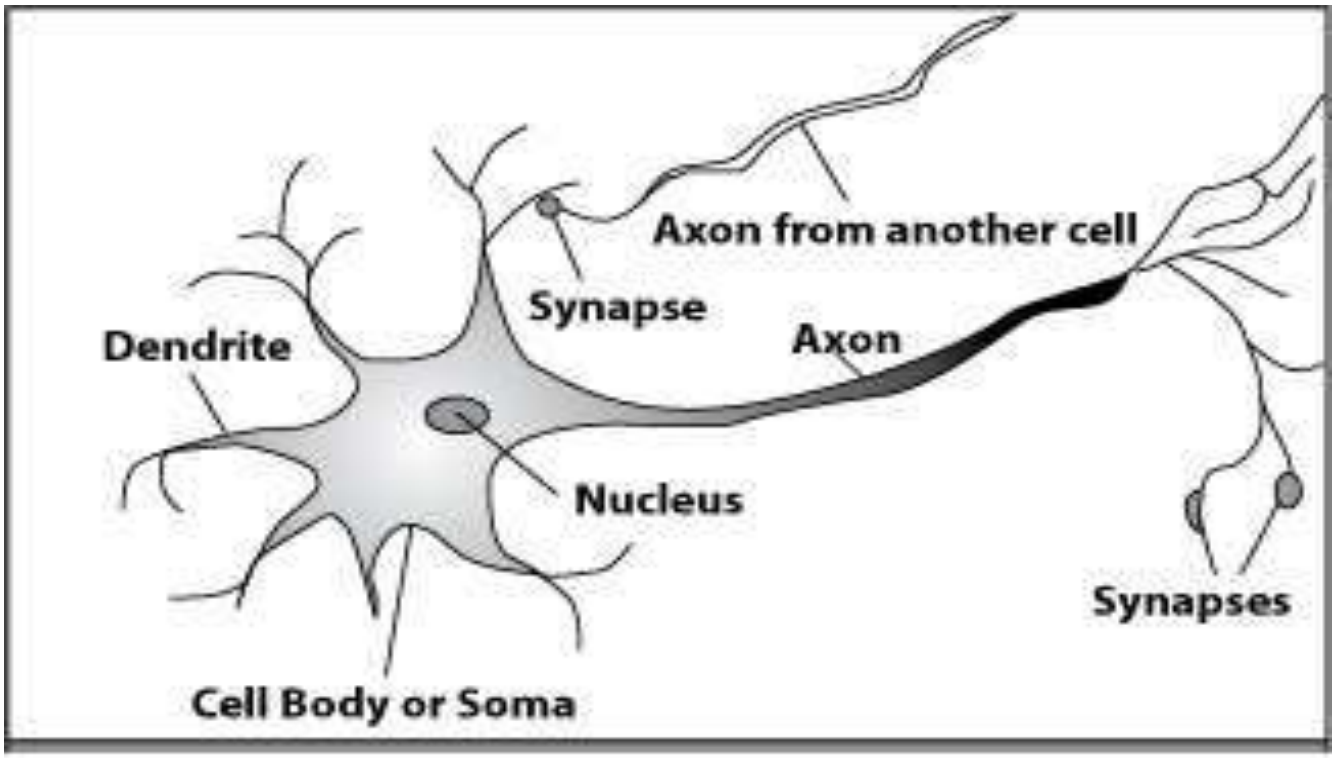


How do our brains work?

- The Brain is A massively parallel information processing system.
- Our brains are a huge network of processing elements. A typical brain contains a network of 10 billion neurons.

How do our brains work?

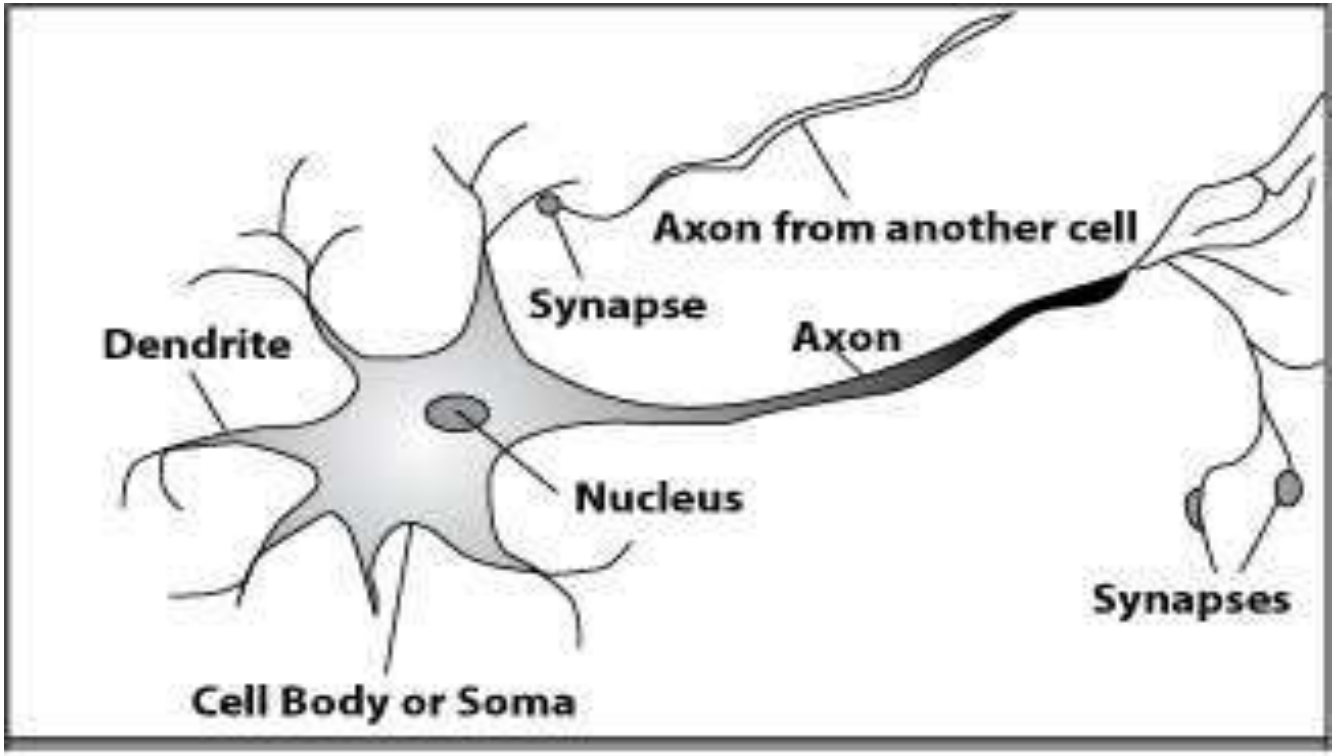
- A processing element



Dendrites: Input
Cell body: Processor
Synaptic: Link
Axon: Output

How do our brains work?

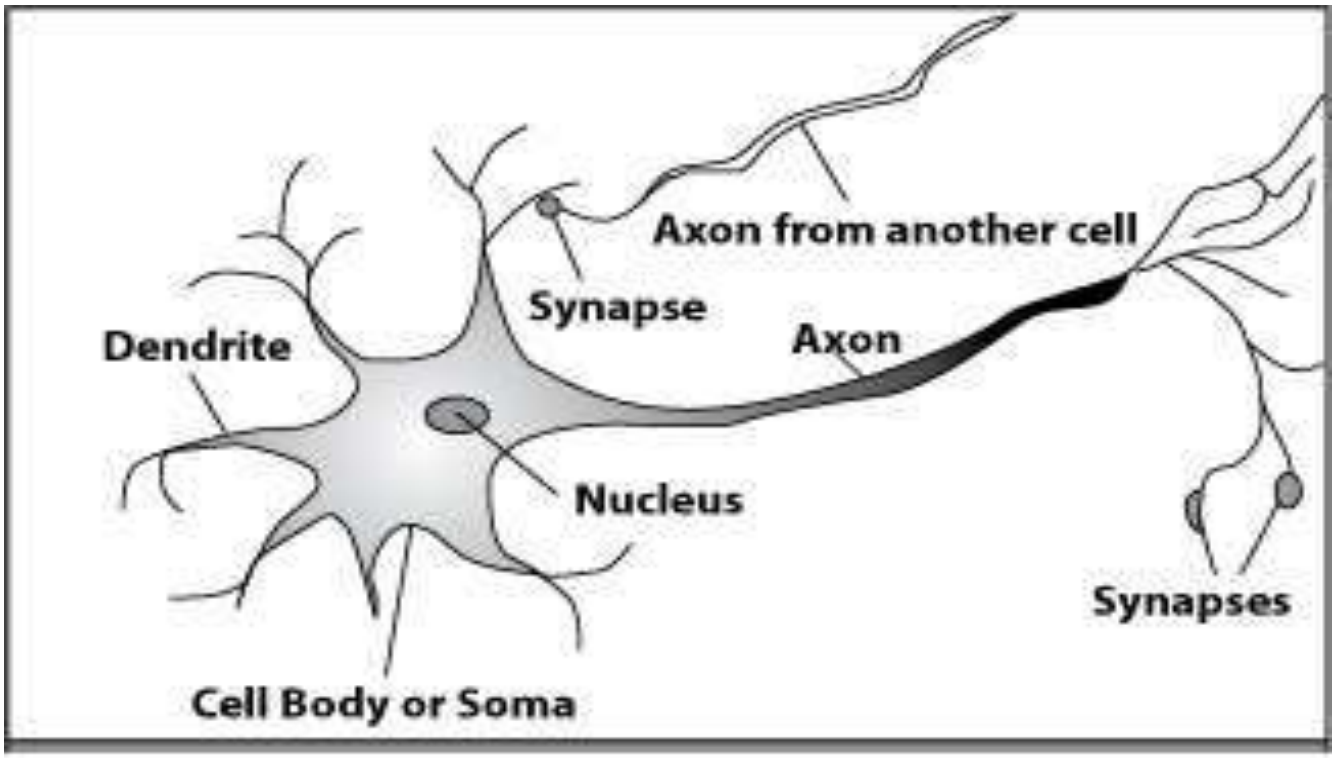
- A processing element



A neuron is connected to other neurons through about *10,000 synapses*

How do our brains work?

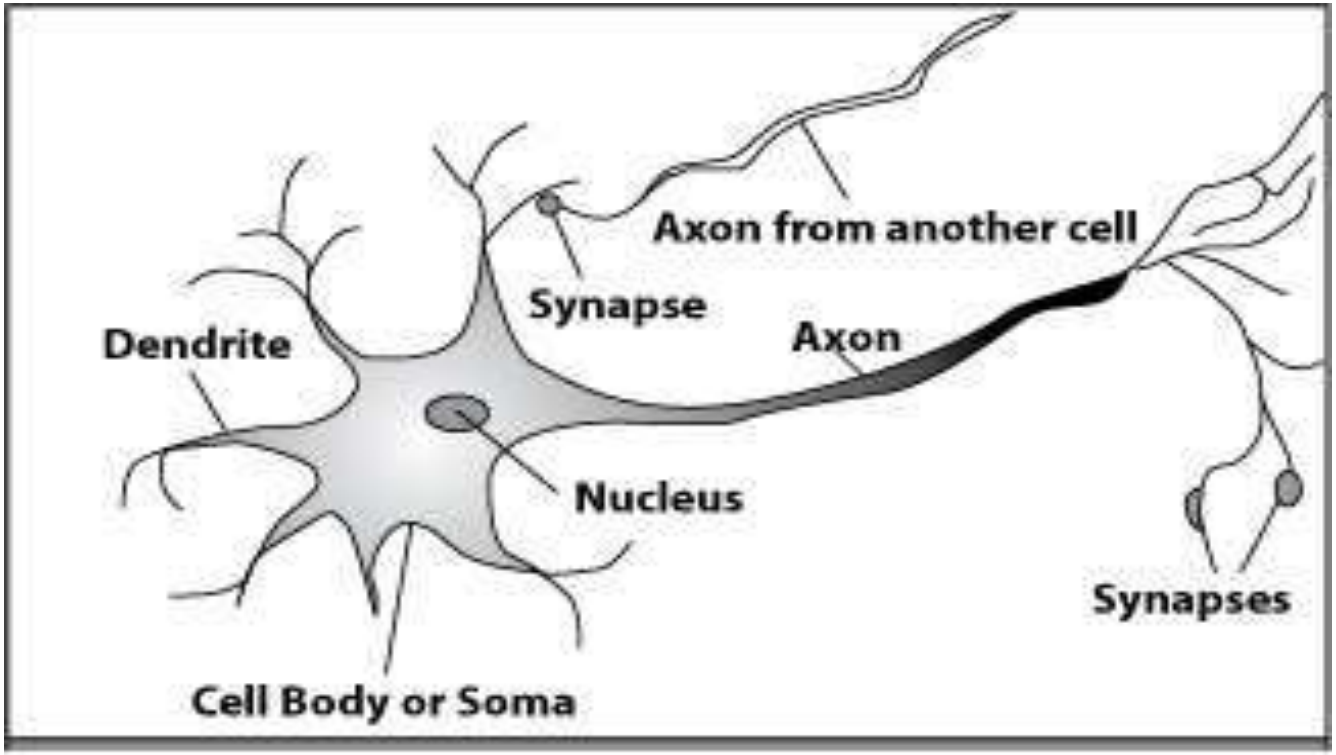
- A processing element



A neuron receives input from other neurons. Inputs are combined.

How do our brains work?

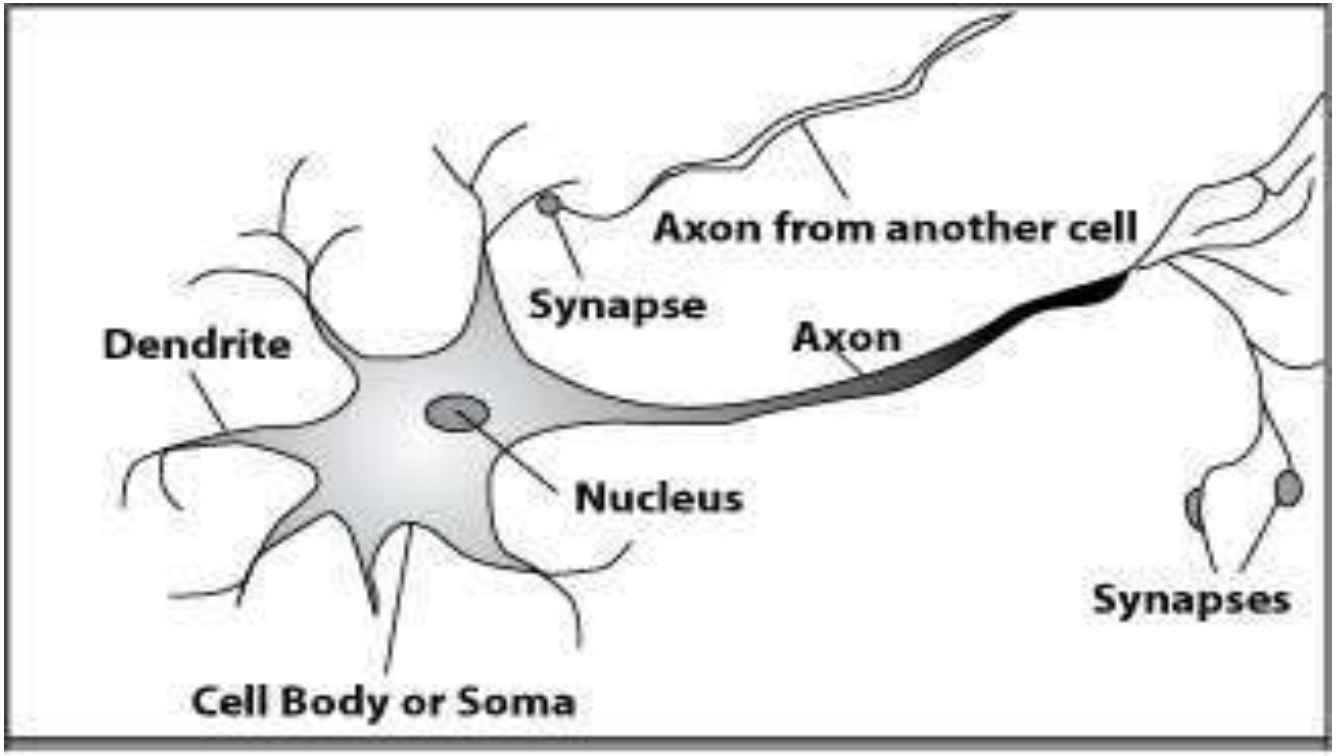
- A processing element



Once input exceeds a critical level, the neuron discharges a spike - an electrical pulse that travels from the body, down the axon, to the next neuron(s)

How do our brains work?

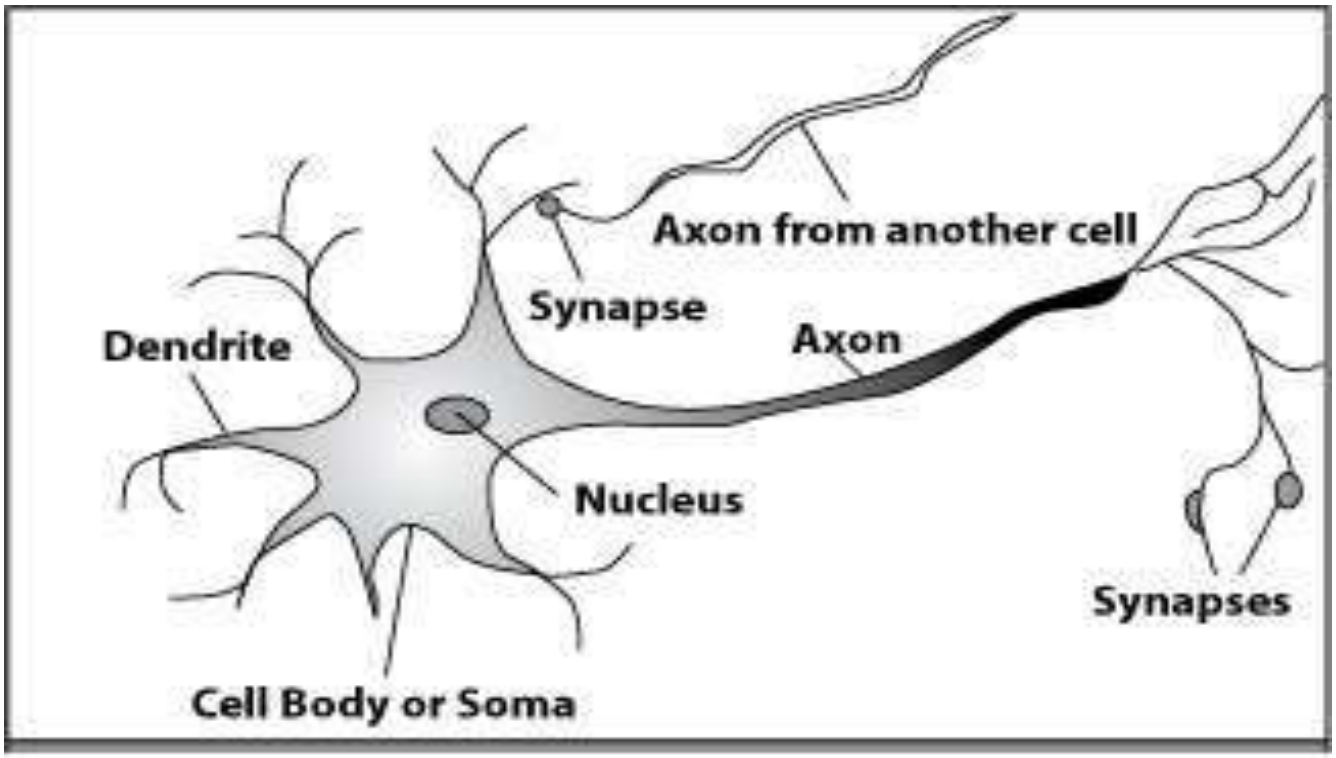
- A processing element



The axon endings almost touch the dendrites or cell body of the next neuron.

How do our brains work?

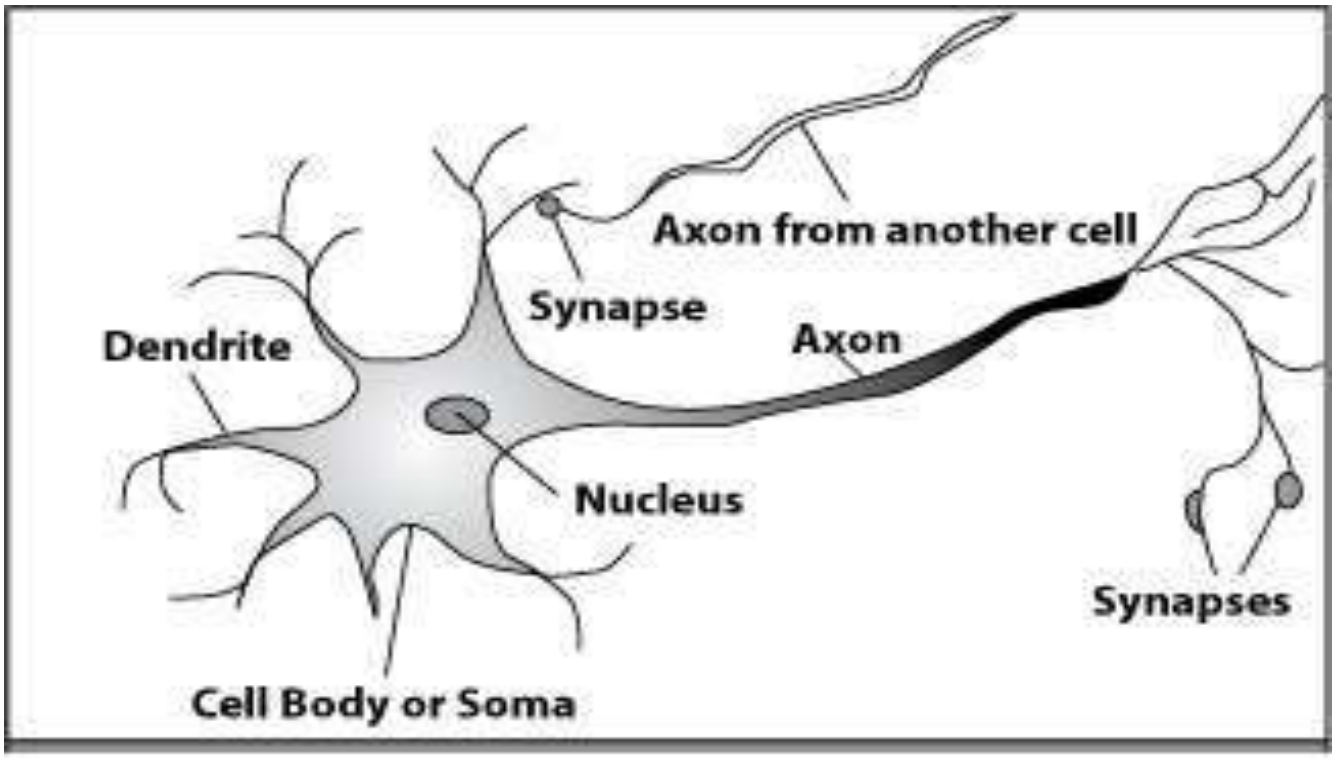
- A processing element



Transmission of an electrical signal from one neuron to the next is effected by neurotransmitters.

How do our brains work?

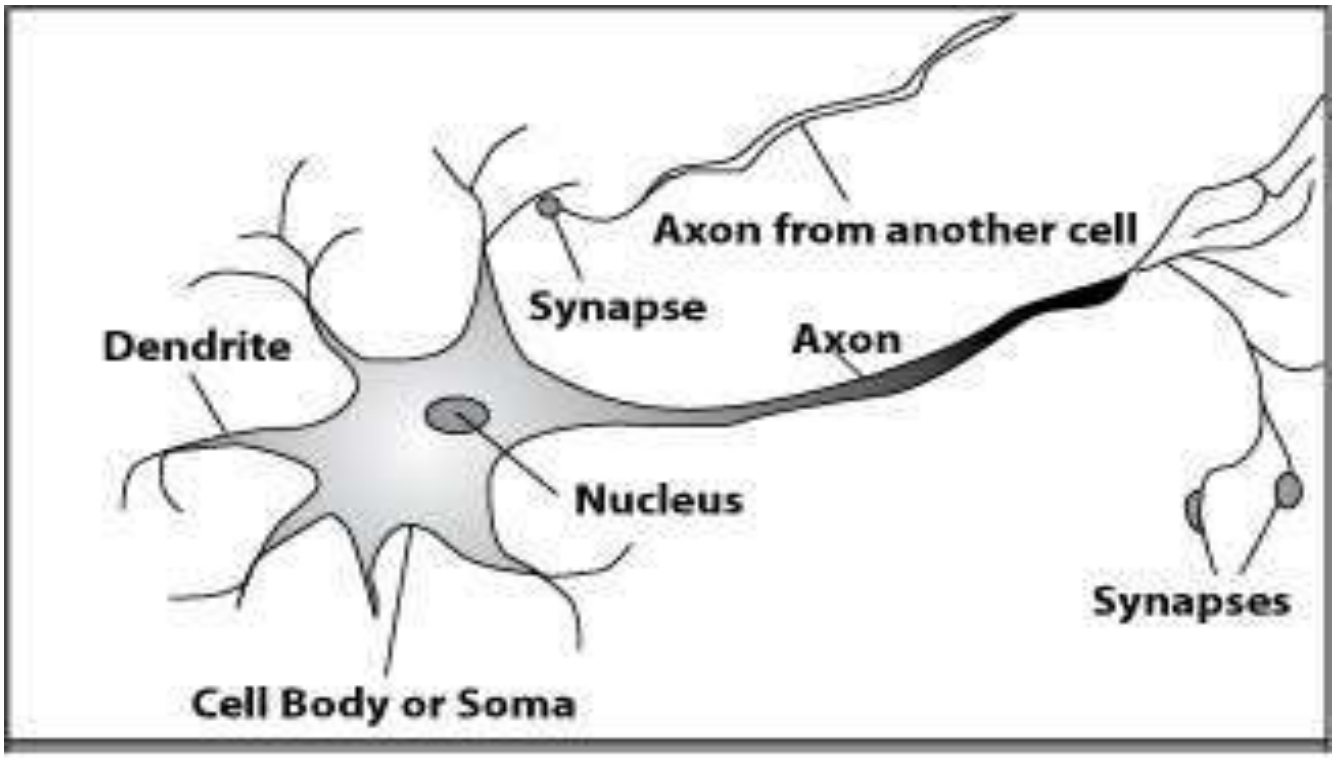
- A processing element



Neurotransmitters are chemicals which are released from the first neuron and which bind to the
Second.

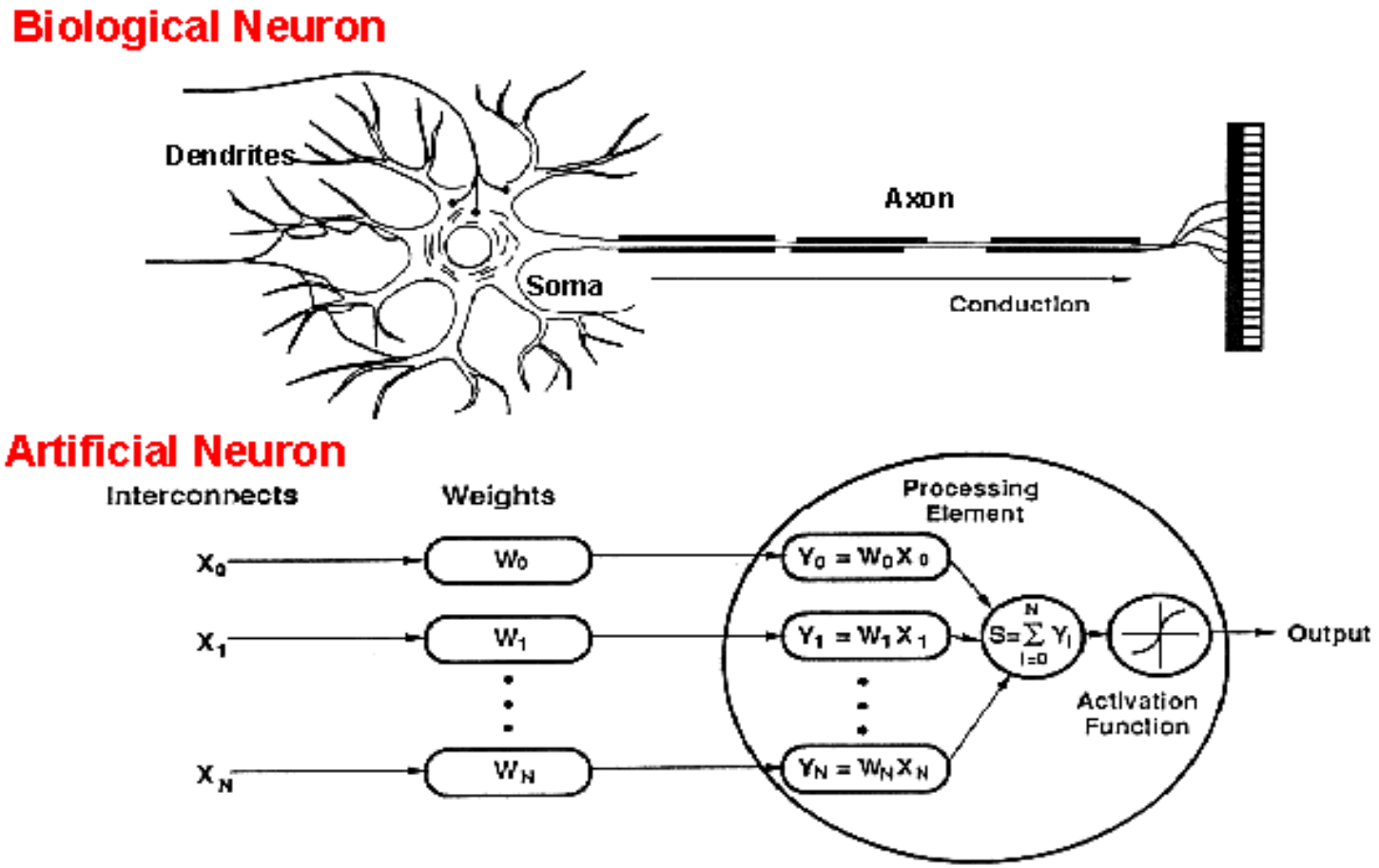
How do our brains work?

- A processing element



This link is called a synapse. The strength of the signal that reaches the next neuron depends on factors such as the amount of neurotransmitter available.

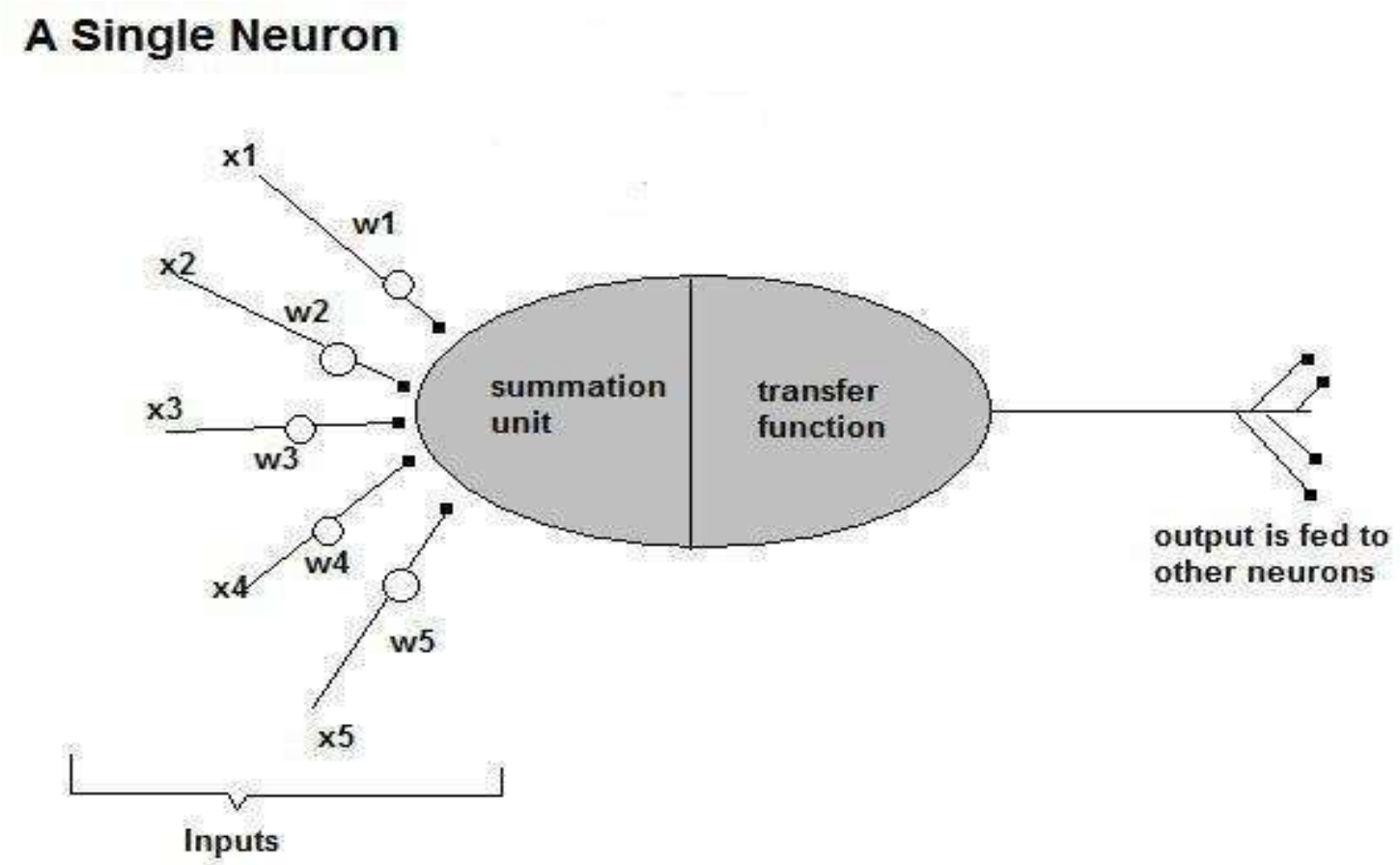
How do ANNs work?



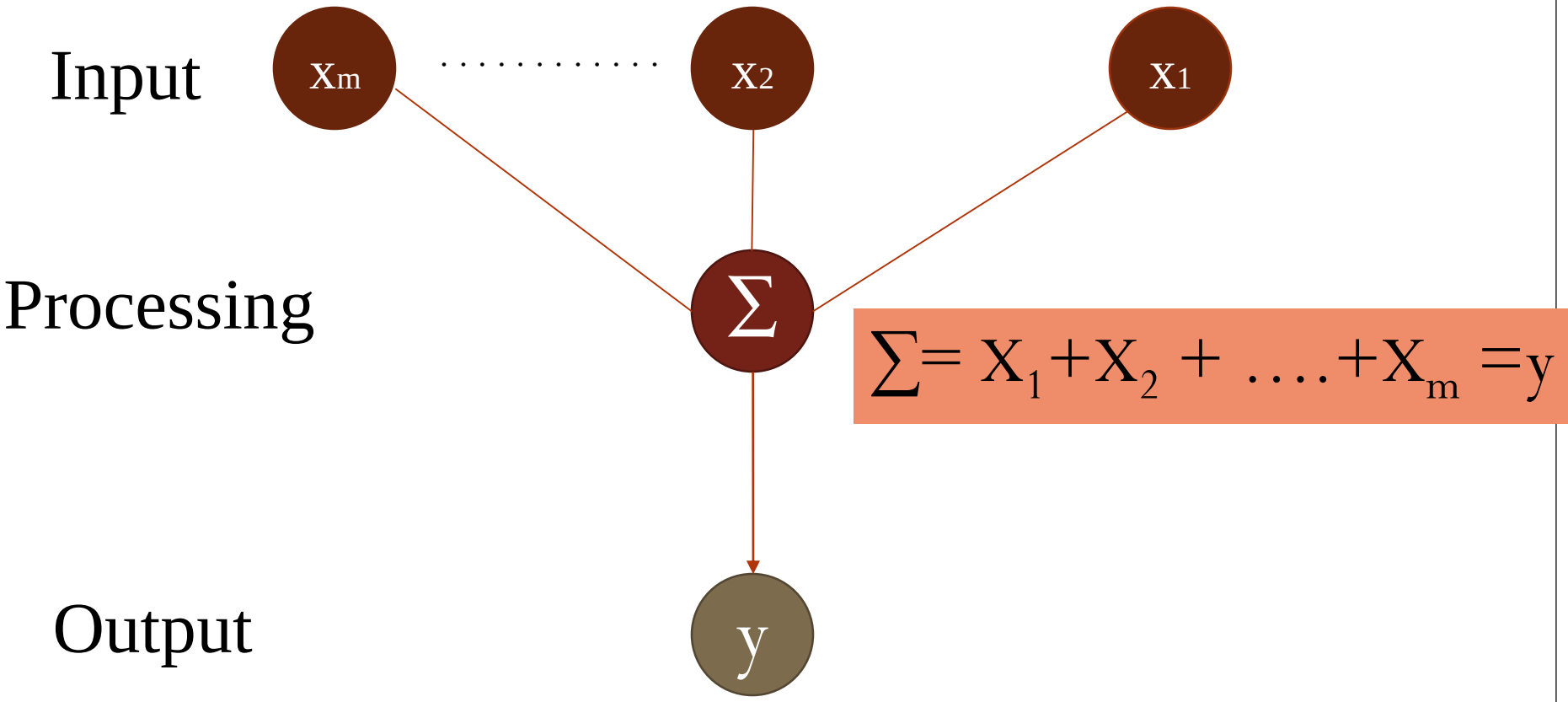
An artificial neuron is an imitation of a human neuron

How do ANNs work?

- Now, let us have a look at the model of an artificial neuron.



How do ANNs work?



How do ANNs work?

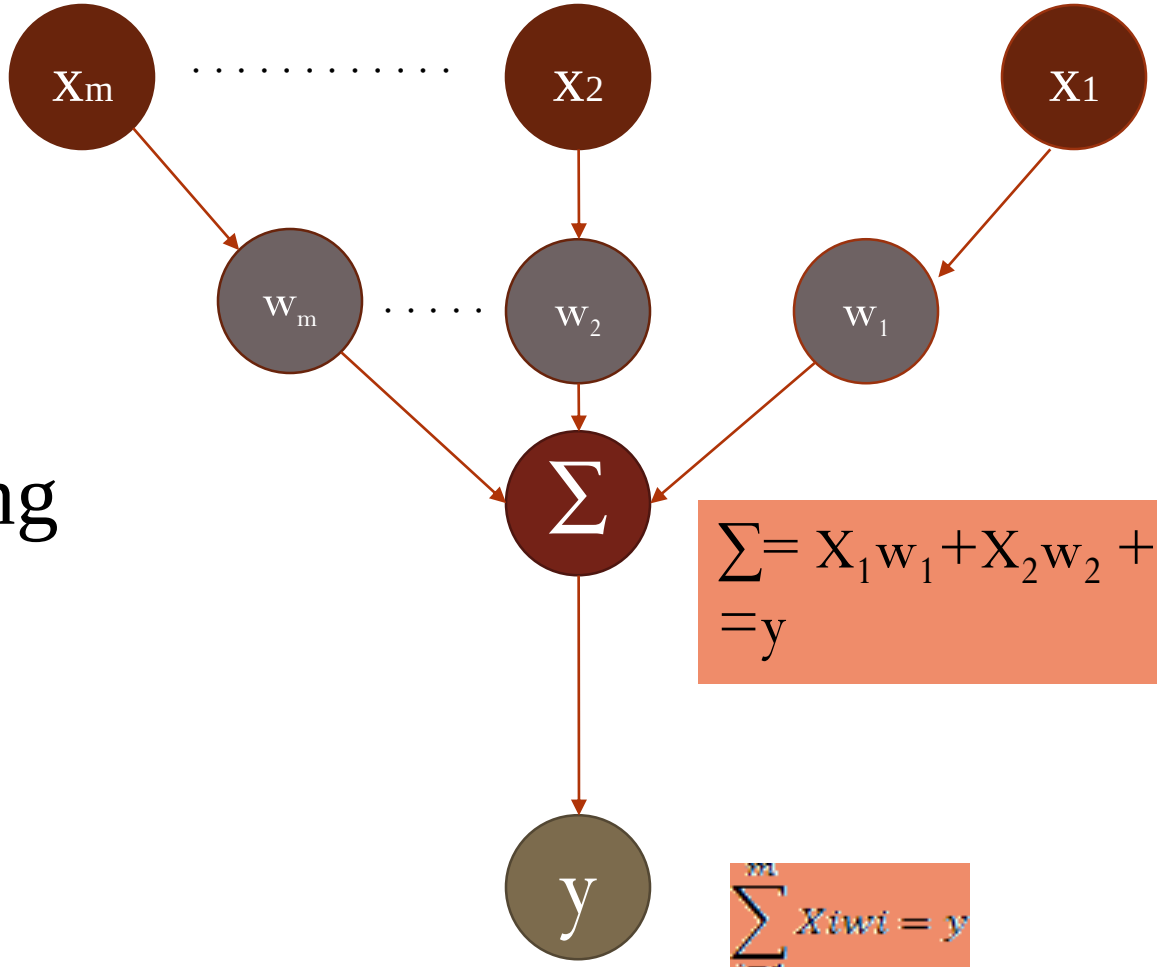
Not all inputs are equal

Input

weights

Processing

Output

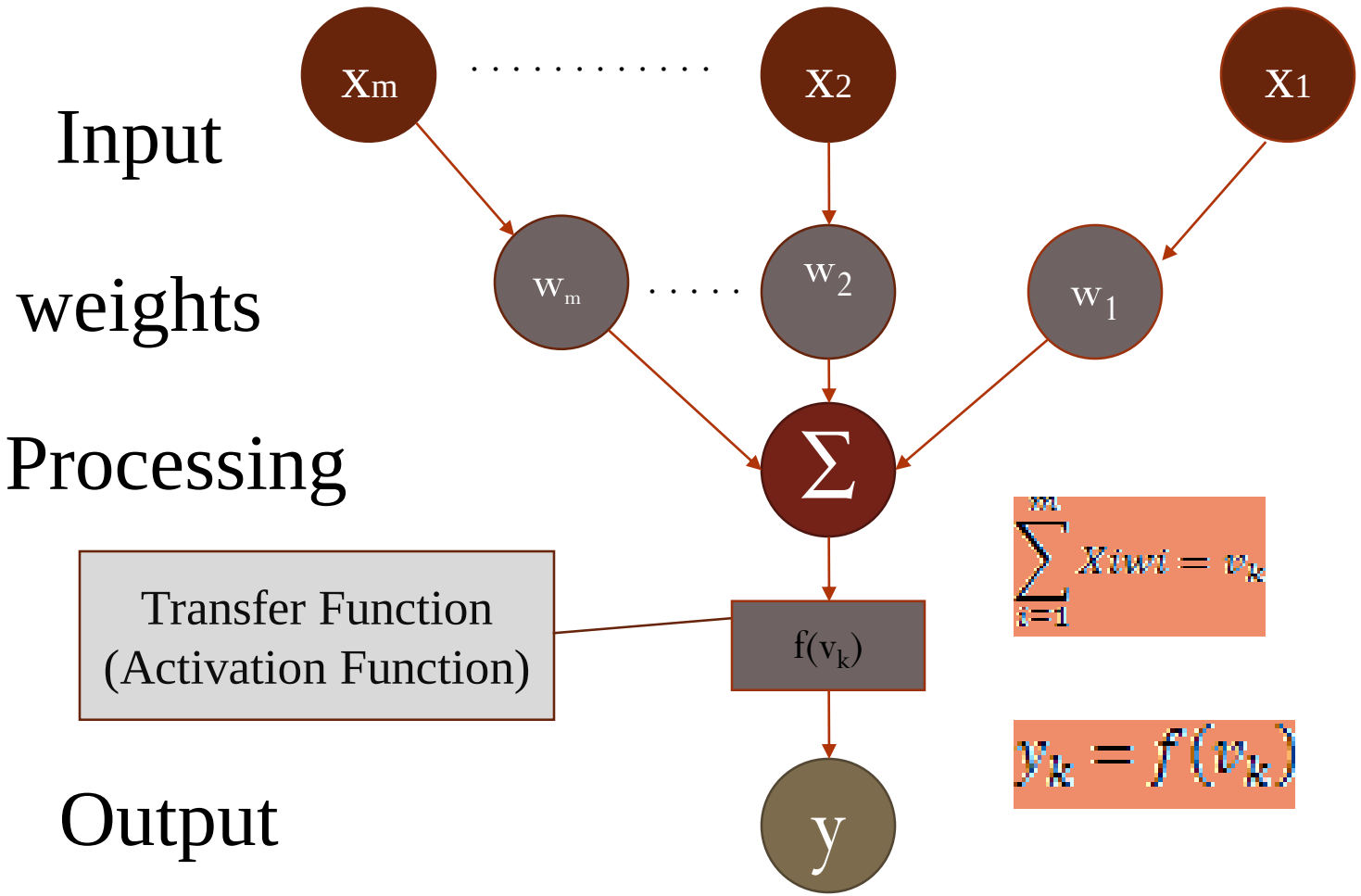


$$\sum = X_1w_1 + X_2w_2 + \ldots + X_mw_m = y$$

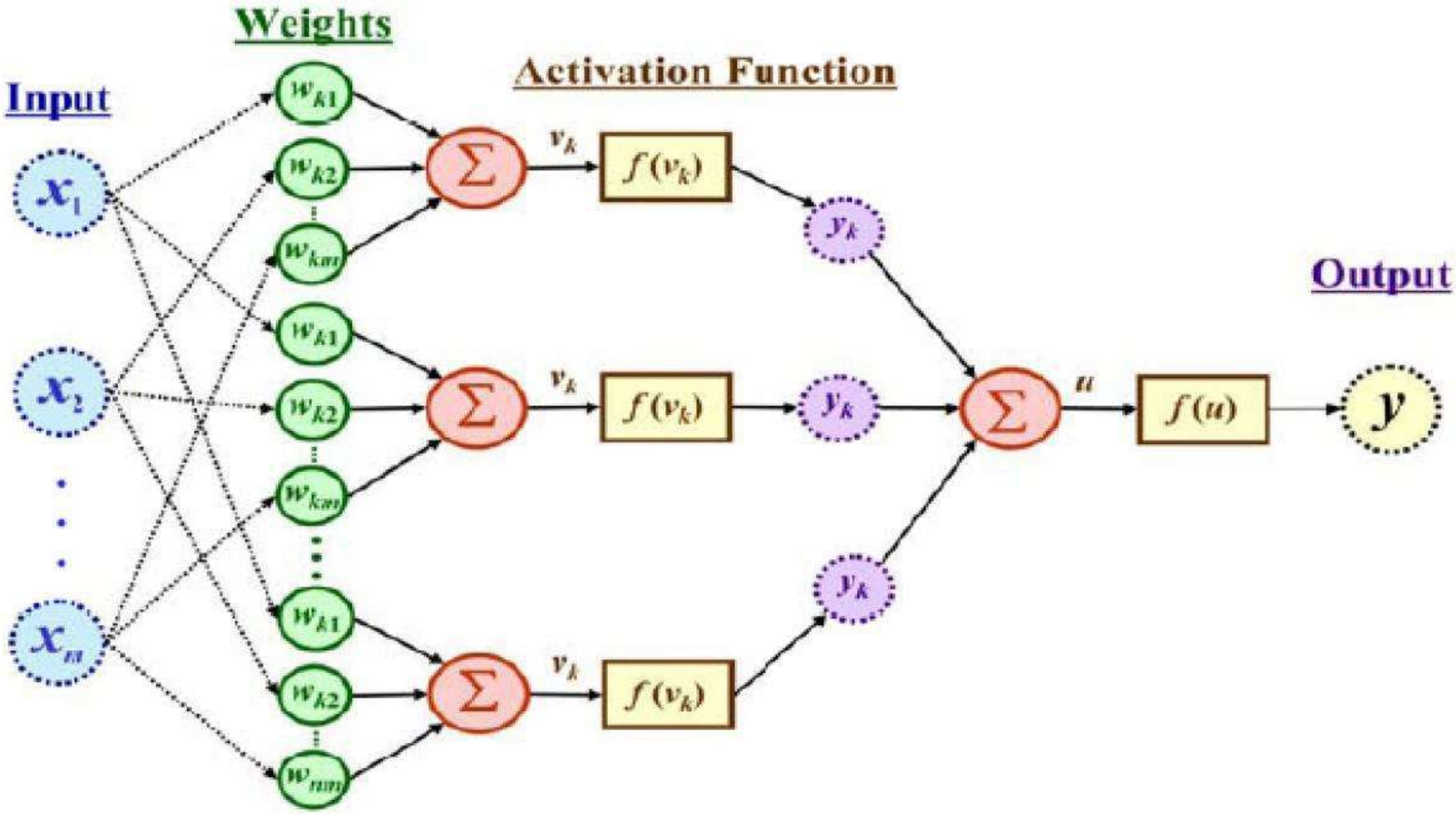
$$\sum_{i=1}^{m} Xiwi = y$$

How do ANNs work?

The signal is not passed down to the next neuron verbatim



The output is a function of the input, that is affected by the weights, and the transfer functions



Artificial Neural Networks

- An ANN can:
  1. compute *any computable* function, by the appropriate selection of the network topology and weights values.
  2. learn from experience!
     - Specifically, by trial-and-error

Learning by trial-and-error

Continuous process of:

- Trial:

  Processing an input to produce an output (In terms of ANN: Compute the output function of a given input)

- Evaluate:

Evaluating this output by comparing the actual output with the expected output.

- Adjust:

  Adjust the *weights*.

How it works?

- Set initial values of the weights randomly.
- Input: truth table of the XOR
- Do
  - Read input (e.g. 0, and 0)
  - Compute an output (e.g. 0.60543)
  - Compare it to the expected output. (Diff= 0.60543)
  - Modify the weights *accordingly.*
- Loop until a condition is met
  - Condition: certain number of iterations
  - Condition: error threshold

Design Issues

- Initial weights (small random values $\in[-1,1]$)
- Transfer function (How the inputs and the weights are combined to produce output?)
- Error estimation
- Weights adjusting
- Number of neurons
- Data representation
- Size of training set

Transfer Functions

- **Linear:** The output is proportional to the total weighted input.
- **Threshold:** The output is set at one of two values, depending on whether the total weighted input is greater than or less than some threshold value.
- **Non-linear:** The output varies continuously but not linearly as the input changes.

Error Estimation

- The **root mean square error (RMSE)** is a frequently-used measure of the differences between values predicted by a model or an estimator and the values actually observed from the thing being modeled or estimated

Weights Adjusting

- After each iteration, weights should be adjusted to minimize the error.
  - All possible weights
  - Back propagation

Back Propagation

- Back-propagation is an example of supervised learning is used at each layer to minimize the error between the layer's response and the actual data
- The error at each hidden layer is an average of the evaluated error
- Hidden layer networks are trained this way

Back Propagation

- N is a neuron.
- N_w is one of N's inputs weights
- N_{out} is N's output.
- $N_w = N_w + \Delta N_w$
- $\Delta N_w = N_{out} * (1 - N_{out}) * N_{ErrorFactor}$
- $N_{ErrorFactor} = N_{ExpectedOutput} - N_{ActualOutput}$
- This works only for the last layer, as we can know the actual output, and the expected output.

Number of neurons

- Many neurons:
  - Higher accuracy
  - Slower
  - Risk of over-fitting
    - Memorizing, rather than understanding
    - The network will be useless with new problems.
- Few neurons:
  - Lower accuracy
  - Inability to learn at all
- Optimal number.

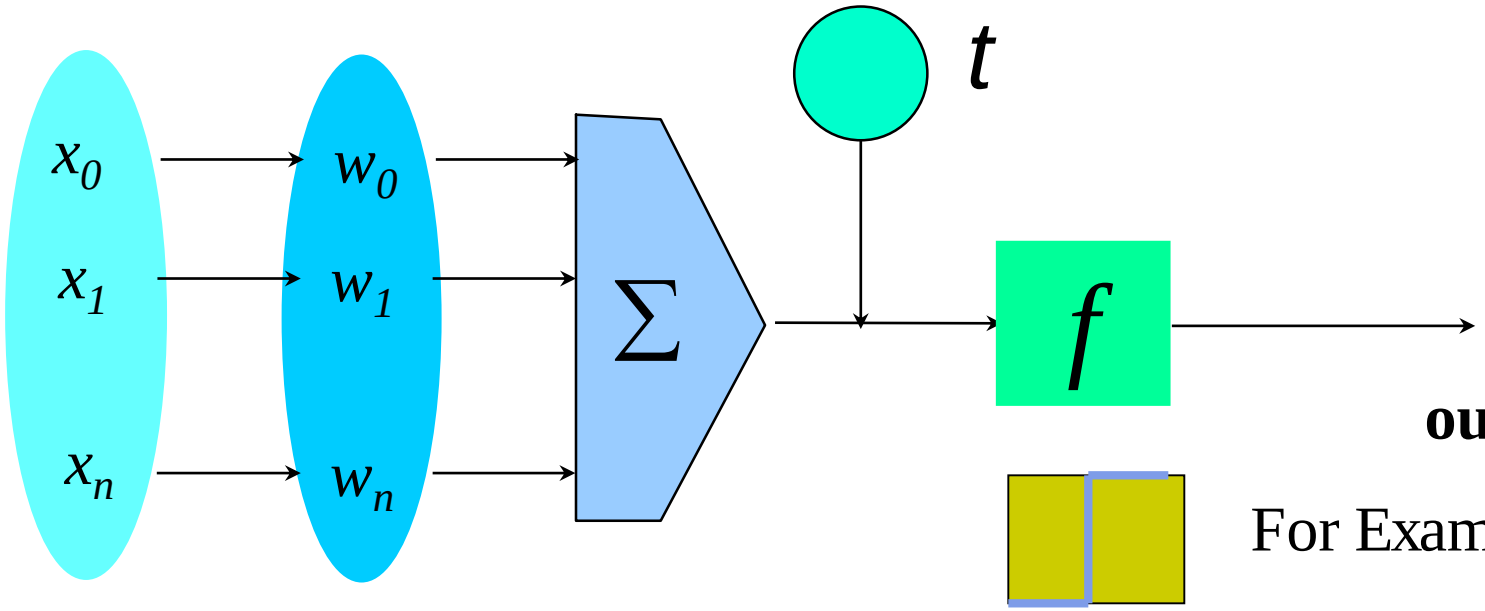
Data representation

- Usually input/output data needs pre-processing
- Pictures
  - Pixel intensity
- Text:
  - A pattern

Size of training set

- No one-fits-all formula
- Over fitting can occur if a “good” training set is not chosen
- What constitutes a “good” training set?
  - Samples must represent the general population.
  - Samples must contain members of each class.
  - Samples in each class must contain a wide range of variations or noise effect.
- The size of the training set is related to the number of hidden neurons

A Neuron (= a perceptron)



Input vector x **weight vector w** **weighted sum** **Activation function**

For Exampl e

$$y = \text{sign}\left(\sum_{i=0}^{n} w_i x_i - t\right)$$

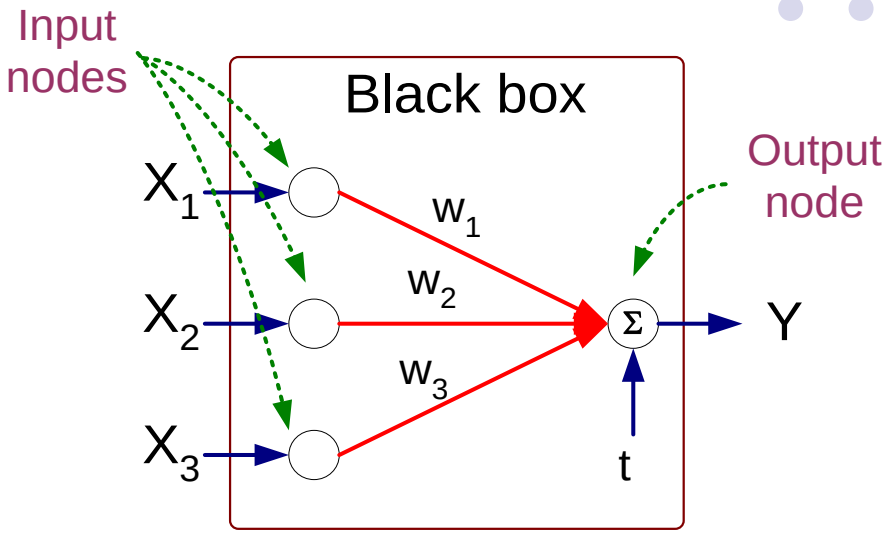
- The *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

Perceptron

- Basic unit in a neural network
- Linear separator
- Parts
  - N inputs, x_1 ... x_n
  - Weights for each input, w_1 ... w_n
  - A bias input x_0 (constant) and associated weight w_0
  - Weighted sum of inputs, $y = w_0x_0 + w_1x_1 + ... + w_nx_n$
  - A threshold function or activation function,
    - i.e 1 if y > t, -1 if y <= t

Artificial Neural Networks (ANN)

- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t



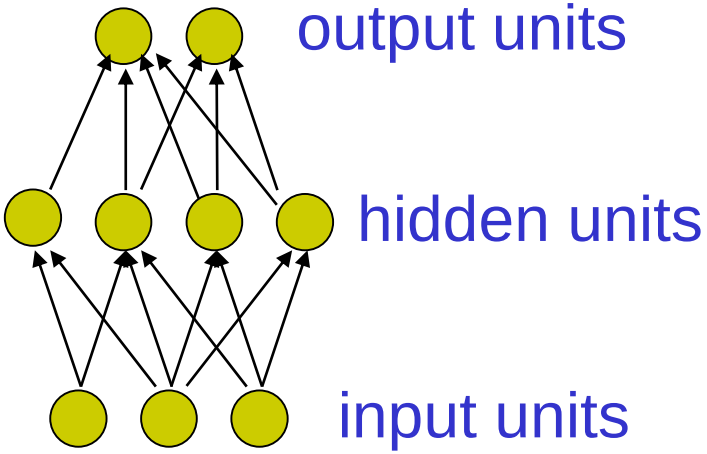
Perceptron Model

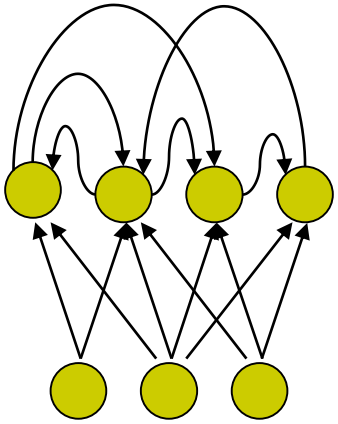
$$Y = I(\sum_i w_i x_i - t)$$ or

$$Y = sign(\sum_i w_i x_i - t)$$

Types of connectivity

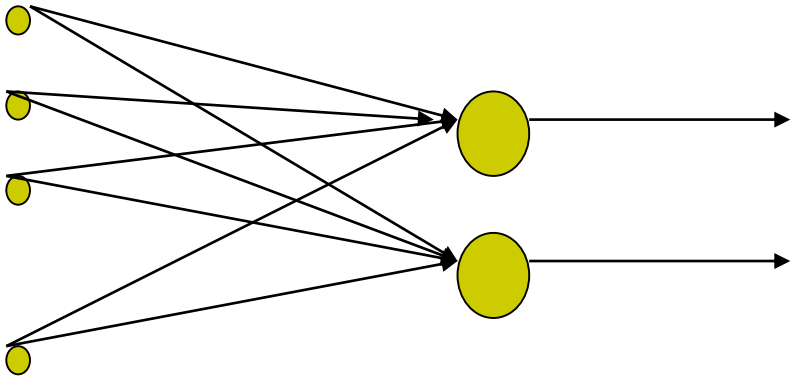
- Feedforward networks
  - These compute a series of transformations
  - Typically, the first layer is the input and the last layer is the output.
- Recurrent networks
  - These have directed cycles in their connection graph. They can have complicated dynamics.
  - More biologically realistic.





Different Network Topologies

- Single layer feed-forward networks
  - Input layer projecting into the output layer



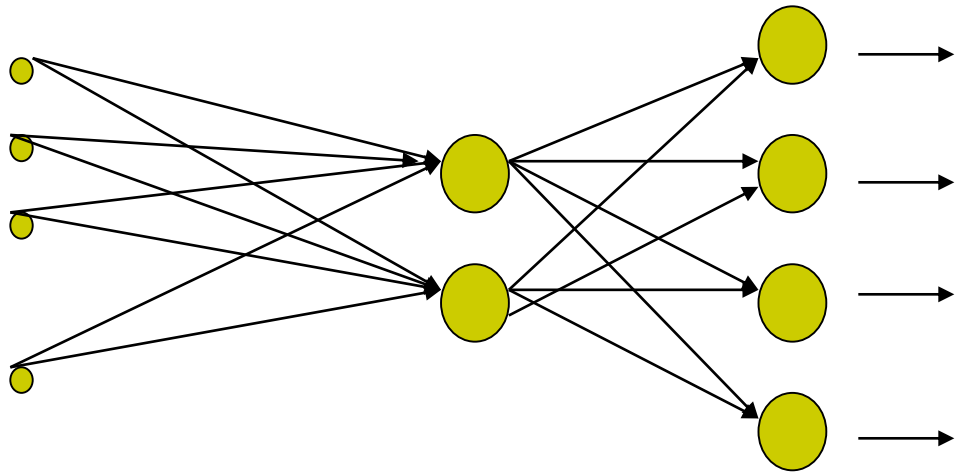
Single layer network

Input layer

Output layer

Different Network Topologies

- Multi-layer feed-forward networks
  - One or more hidden layers. Input projects only from previous layers onto a layer.



Input layer

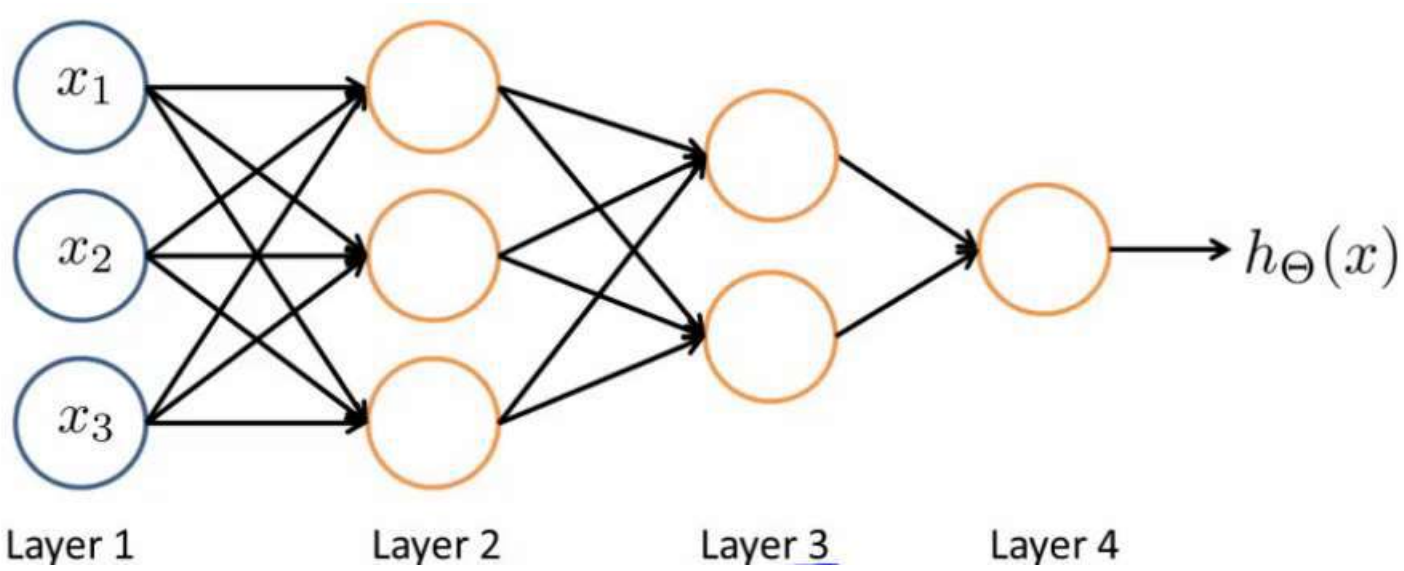
Hidden layer

Output layer

2-layer or 1-hidden layer *fully connected* network

Different Network Topologies

- Multi-layer feed-forward networks



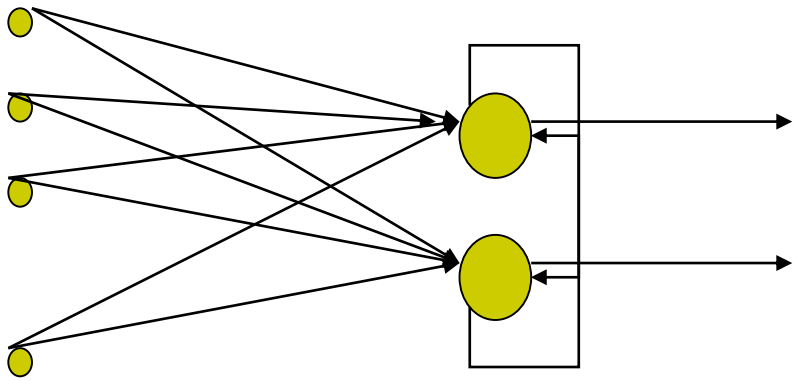
Input layer

Hidden layers

Output layer

Different Network Topologies

- Recurrent networks
  - A network with feedback, where some of its inputs are connected to some of its outputs (discrete time).



Recurrent network

Input layer

Output layer

Algorithm for learning ANN

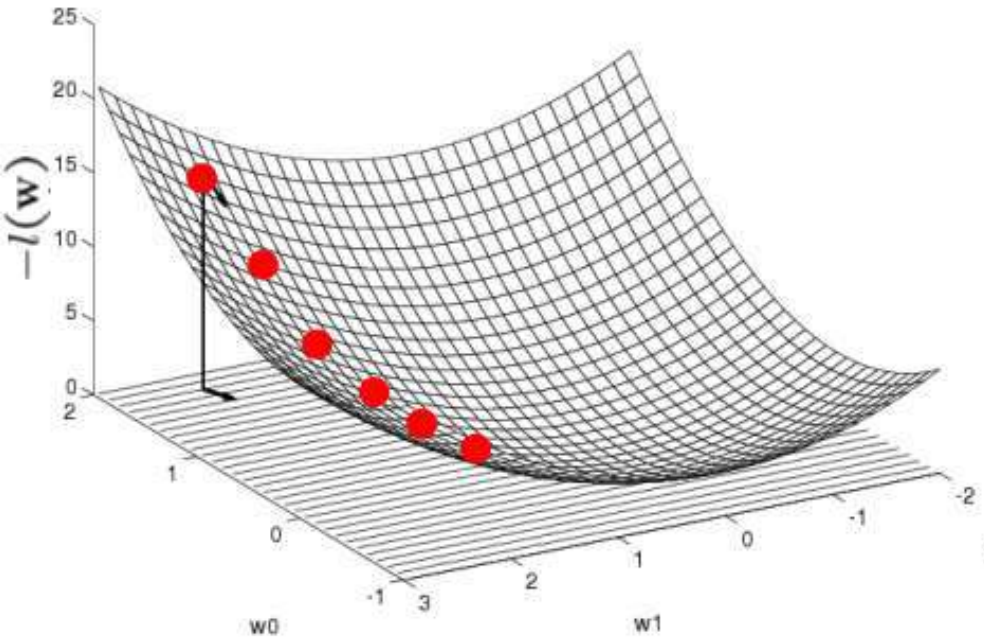
- Initialize the weights (w_0, w_1, …, w_k)

- Adjust the weights in such a way that the output of ANN is consistent with class labels of training examples
  - Error function: $$E = \sum_i [Y_i - f(w_i, X_i)]^2$$
  - Find the weights w_i's that minimize the above error function
    - e.g., gradient descent, backpropagation algorithm

Optimizing concave/convex function

- Maximum of a concave function = minimum of a convex function

Gradient ascent (concave) / Gradient descent (convex)



Gradient:

$$\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_d}]'$$

Update rule: Learning rate, $\eta>0$

$$\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left. \frac{\partial l(\mathbf{w})}{\partial w_i} \right|_t$$

Gradient ascent rule

GRADIENT DESCENT

Suppose we have a scalar function $\mathrm{f}(\mathrm{w}): \Re \rightarrow \Re$

We want to find a local minimum.

Assume our current weight is *w*

GRADIENT DESCENT RULE: $w \leftarrow w - \eta \frac{\partial}{\partial w} \mathrm{f}(w)$

η is called the LEARNING RATE. A small positive number, e.g. η = 0.05

Gradient Descent in "m" Dimensions

Given $\mathrm{f}(\mathbf{w}) : \Re^m \rightarrow \Re$

$$\nabla \mathrm{f}(\mathrm{w}) = \begin{pmatrix} \frac{\partial}{\partial w_1} \mathrm{f}(\mathrm{w}) \\ \vdots \\ \frac{\partial}{\partial w_m} \mathrm{f}(\mathrm{w}) \end{pmatrix}$$ points in direction of steepest ascent.

$|\nabla \mathrm{f}(\mathrm{w})|$ is the gradient in that direction

GRADIENT DESCENT RULE: $\mathrm{w} \leftarrow \mathrm{w} - \eta \nabla \mathrm{f}(\mathrm{w})$

Equivalently $w_j \leftarrow w_j - \eta \frac{\partial}{\partial w_j} \mathrm{f}(\mathrm{w})$ ....where w_j is the jth weight

"just like a linear feedback system"

Linear Perceptron Training Rule

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent tells us we should update **w** thusly if we wish to minimize *E:*

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

So what's $\frac{\partial E}{\partial w_j}$?

$$\frac{\partial E}{\partial w_j} = \sum_{k=1}^{R} \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

$$= \sum_{k=1}^{R} 2(y_k - \mathbf{w}^T \mathbf{x}_k) \frac{\partial}{\partial w_j} (y_k - \mathbf{w}^T \mathbf{x}_k)$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}_k$$

...where...

$$\delta_k = y_k - \mathbf{w}^T \mathbf{x}_k$$

$$= -2 \sum_{k=1}^{R} \delta_k \frac{\partial}{\partial w_j} \sum_{i=1}^{m} w_i x_{ki}$$

$$= -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

Linear Perceptron Training Rule

$$E = \sum_{k=1}^{R} (y_k - \mathbf{w}^T \mathbf{x}_k)^2$$

Gradient descent tells us we should update **w** thusly if we wish to minimize *E:*

$$w_j \leftarrow w_j - \eta \frac{\partial E}{\partial w_j}$$

...where...

$$\frac{\partial E}{\partial w_j} = -2 \sum_{k=1}^{R} \delta_k x_{kj}$$

$$w_j \leftarrow w_j + 2\eta \sum_{k=1}^{R} \delta_k x_{kj}$$

We frequently neglect the 2 (meaning we halve the learning rate)

The "Batch" perceptron algorithm

1) Randomly initialize weights w_1 w_2 ... w_m

2) Get your dataset (append 1's to the inputs if you don't want to go through the origin).

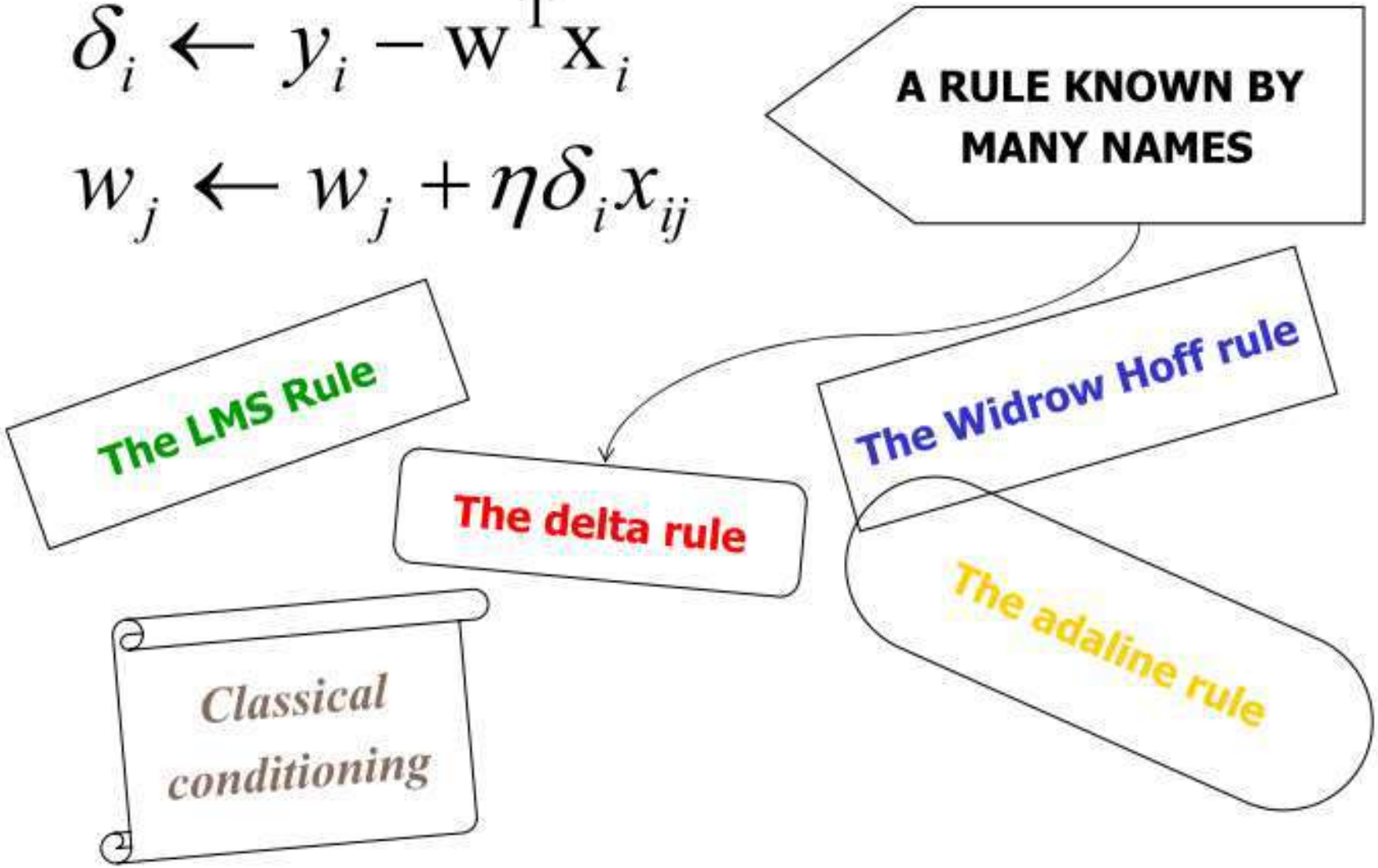
3) for $i = 1$ to R $\quad \delta_i := y_i - \mathbf{w}^T \mathbf{x}_i$

4) for $j = 1$ to m $\quad w_j \leftarrow w_j + \eta \sum_{i=1}^{R} \delta_i x_{ij}$

5) if $\sum \delta_i^2$ stops improving then stop. Else loop back to 3.

$$\delta_i \leftarrow y_i - \mathrm{w}^{\mathrm{T}}\mathrm{x}_i$$

$$w_j \leftarrow w_j + \eta\delta_i x_{ij}$$



Perceptrons and Boolean Functions

- Can learn any disjunction of literals

  e.g. $x_1 \wedge \sim x_2 \wedge \sim x_3 \wedge x_4 \wedge x_5$

- Can learn majority function

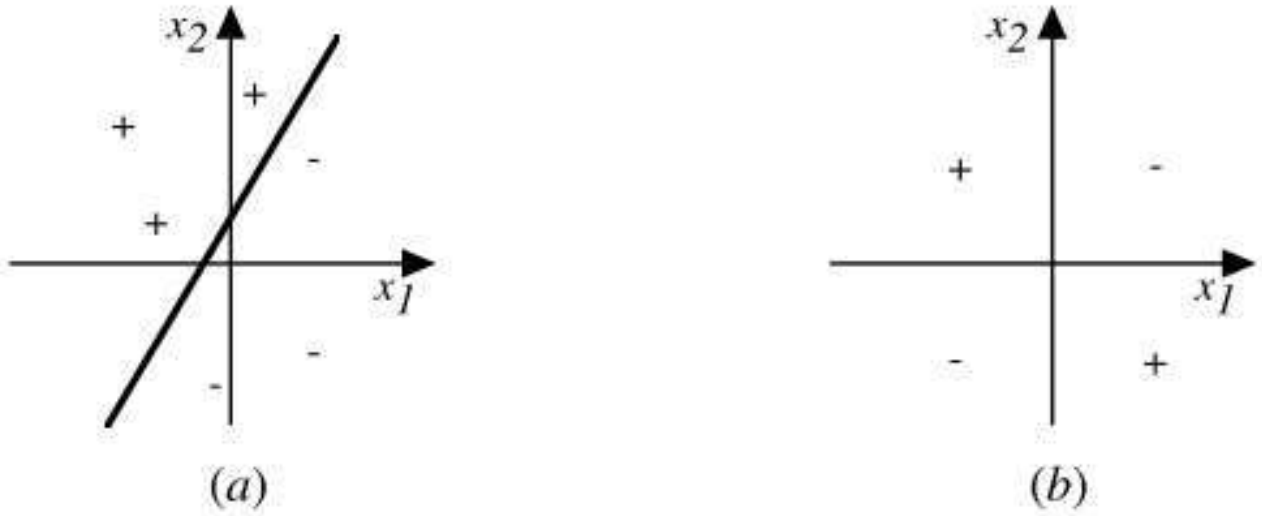
  $f(x_1, x_2 \ldots x_n) = \begin{cases} 1 \text{ if } n/2 \ x_i\text{'s or more are} = 1 \\ 0 \text{ if less than } n/2 \ x_i\text{'s are} = 1 \end{cases}$

- What about the exclusive or function?

  $f(x_1, x_2) = x_1 \ \forall \ x_2 =$

  $(x_1 \wedge \sim x_2) \vee (\sim x_1 \wedge x_2)$

Decision surface of a perceptron

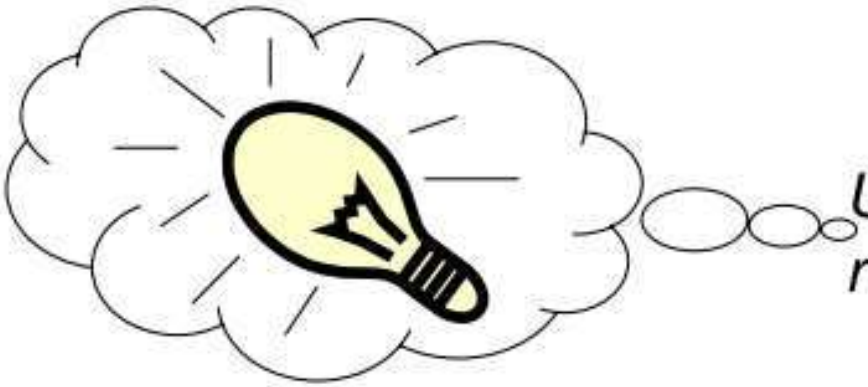


- Decision surface is a hyperplane
  - Can capture linearly separable classes
- Non-linearly separable
  - Use a network of them

Multilayer Networks

The class of functions representable by perceptrons is limited

$$\text{Out}(\mathbf{x}) = g\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) = g\left(\sum_j w_j x_j\right)$$



Use a wider representation !

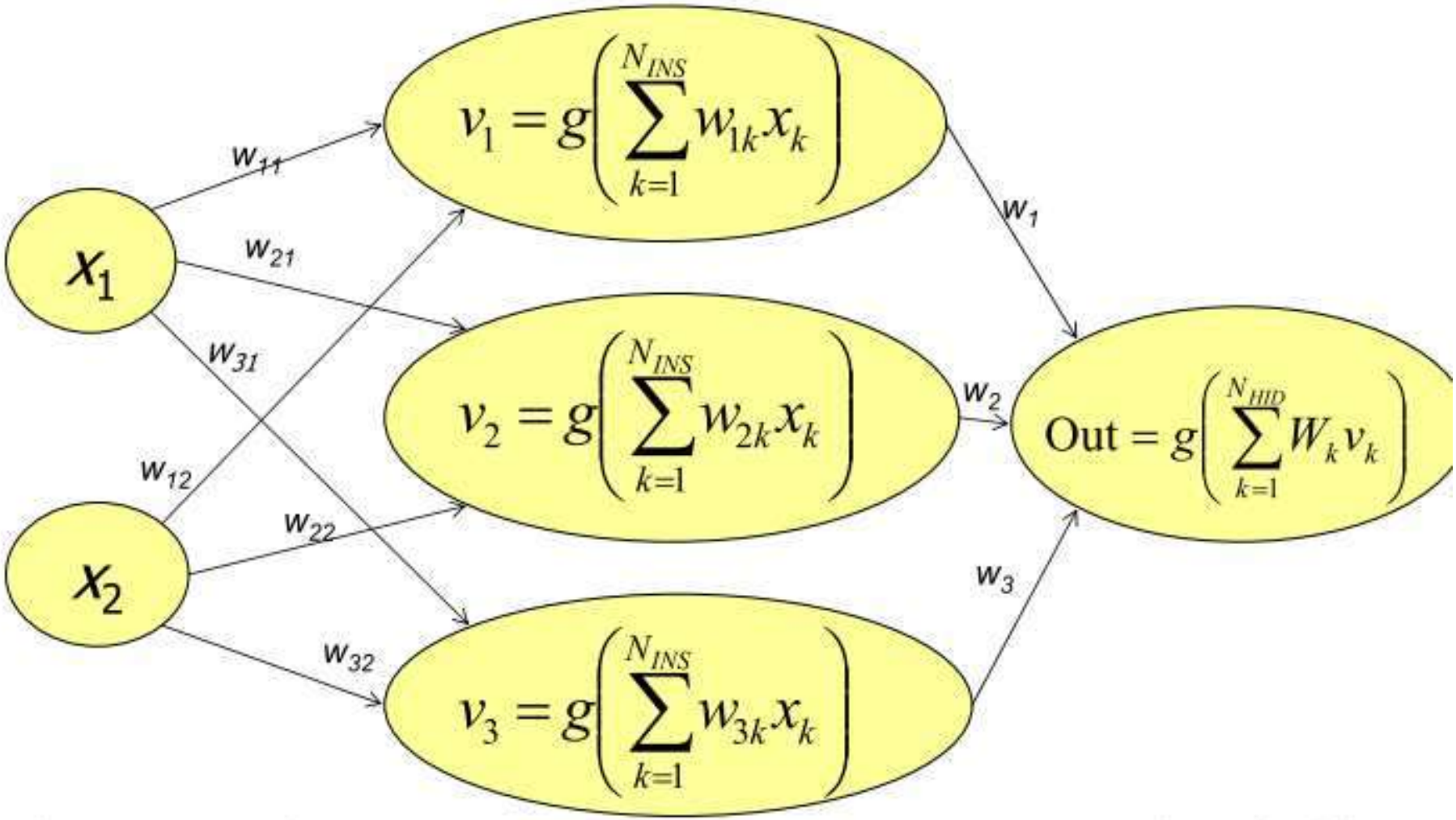
$$\text{Out}(\mathbf{x}) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_{jk}\right)\right)$$

This is a nonlinear function
Of a linear combination
Of non linear functions
Of linear combinations of inputs

A 1-HIDDEN LAYER NET

$N_{INPUTS} = 2$

$N_{HIDDEN} = 3$

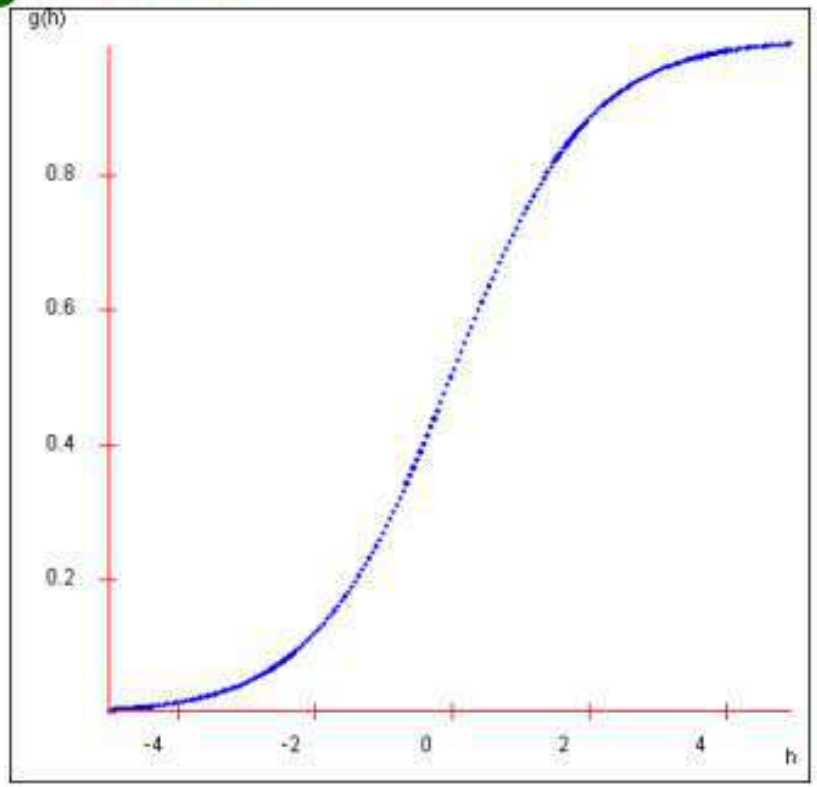


Multi-layer Networks

- Linear units inappropriate
  - No more expressive than a single layer
- Introduce non-linearity
  - Threshold not differentiable
- Use sigmoid function

The Sigmoid

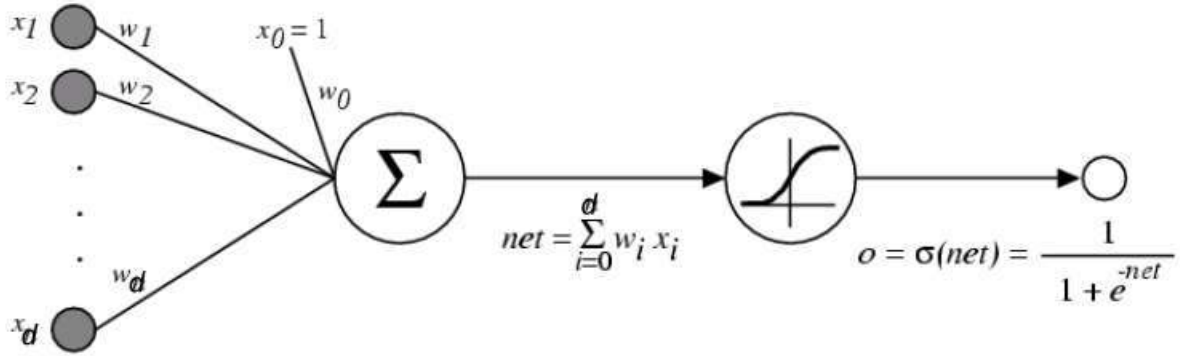
$$g(h) = \frac{1}{1+\exp(-h)}$$



Now we choose ***w*** to minimize

$$\sum_{i=1}^{R}\left[y_i - \mathrm{Out}(\mathrm{x}_i)\right]^2 = \sum_{i=1}^{R}\left[y_i - g(\mathrm{w}^{\mathrm{T}}\mathrm{x}_i)\right]^2$$

Sigmoid Unit



$\sigma(x)$ is the sigmoid function/activation function (also linear, threshold)

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

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

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units → Backpropagation

Backpropagation

$$\text{Out}(\text{x}) = g\left(\sum_j W_j g\left(\sum_k w_{jk} x_k\right)\right)$$

Find a set of weights $\{W_j\}, \{w_{jk}\}$

to minimize

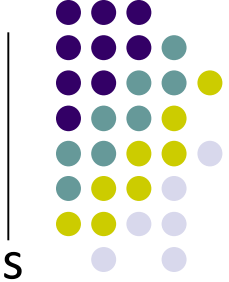
$$\sum_i \left(y_i - \text{Out}(\text{x}_i)\right)^2$$

by gradient descent.

That's it!

That's the backpropagation algorithm.

Backpropagation

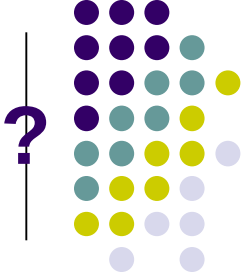


- Iteratively process a set of training tuples & compare the network's prediction with the actual known target value
- For each training tuple, the weights are modified to **minimize the mean squared error** between the network's prediction and the actual target value
- Modifications are made in the "**backwards**" direction: from the output layer, through each hidden layer down to the first hidden layer, hence "**backpropagation**"
- Steps
  - Initialize weights (to small random #s) and biases in the network
  - Propagate the inputs forward (by applying activation function)
  - Backpropagate the error (by updating weights and biases)
  - Terminating condition (when error is very small, etc.)

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

How A Multi-Layer Neural Network Works?



- The **inputs** to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the **input layer**
- They are then weighted and fed simultaneously to a **hidden layer**
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the **output layer**, which emits the network's prediction
- The network is **feed-forward** in that none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform **nonlinear regression**: Given enough hidden units and enough training samples, they can closely approximate any function

Defining a Network Topology

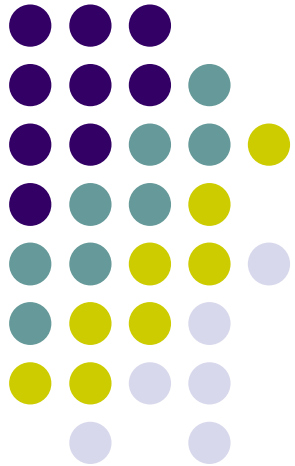
- First decide the **network topology:** # of units in the *input layer*, # of *hidden layers* (if > 1), # of units in *each hidden layer*, and # of units in the *output layer*
- Normalizing the input values for each attribute measured in the training tuples to [0.0—1.0]
- One **input** unit per domain value, each initialized to 0
- **Output**, if for classification and more than two classes, one output unit per class is used
- Once a network has been trained and its accuracy is **unacceptable**, repeat the training process with a *different network topology* or a *different set of initial weights*

Backpropagation and Interpretability

- Efficiency of backpropagation: Each **epoch** (one interation through the training set) takes O(|D| * *w*), with |D| tuples and *w* weights, but # of epochs can be exponential to n, the number of inputs, in the worst case
- **Rule extraction from networks:** network pruning
  - Simplify the network structure by removing weighted links that have the least effect on the trained network
  - Then perform link, unit, or activation value clustering
  - The set of input and activation values are studied to derive rules describing the relationship between the input and hidden unit layers
- **Sensitivity analysis:** assess the impact that a given input variable has on a network output. The knowledge gained from this analysis can be represented in rules

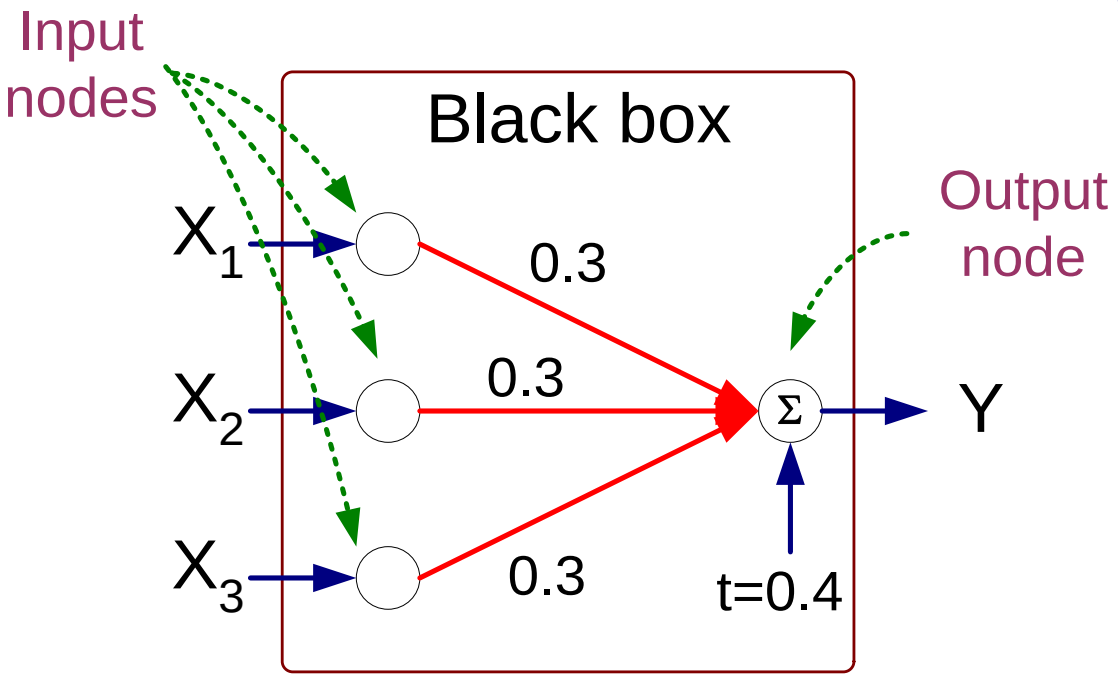
Neural Network as a Classifier

- Weakness
  - Long training time
  - Require a number of parameters typically best determined empirically, e.g., the network topology or “structure.”
  - Poor interpretability: Difficult to interpret the symbolic meaning behind the learned weights and of “hidden units” in the network
- Strength
  - High tolerance to noisy data
  - Ability to classify untrained patterns
  - Well-suited for continuous-valued inputs and outputs
  - Successful on a wide array of real-world data
  - Algorithms are inherently parallel
  - Techniques have recently been developed for the extraction of rules from trained neural networks



Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	0
0	1	0	0
0	1	1	1
0	0	0	0



$$Y = I(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4 > 0)$$

$$\text{where } I(z) = \begin{cases} 1 & \text{if } z \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

Linear Perceptrons

They are multivariate linear models:

$$\text{Out}(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$$

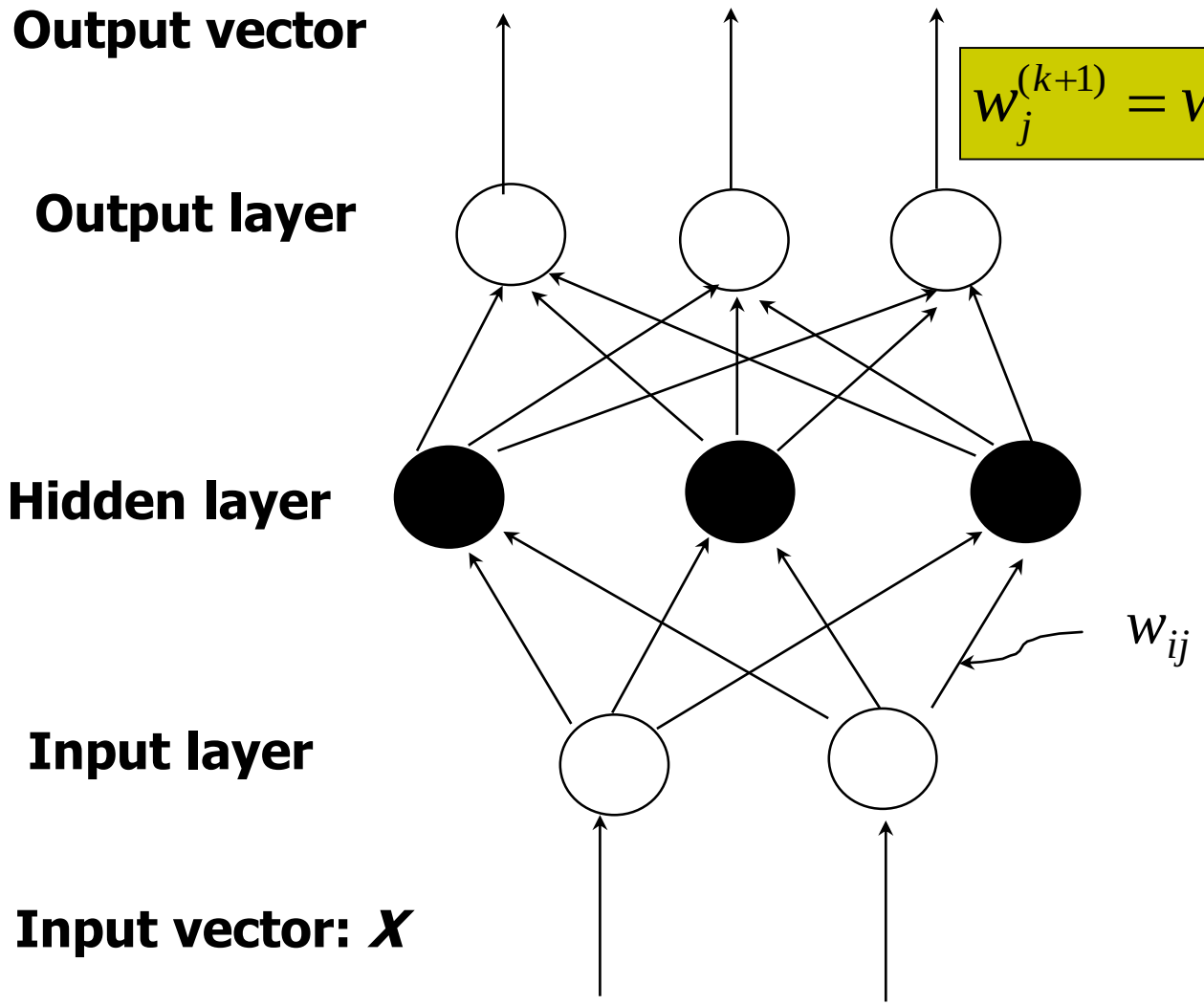
And "training" consists of minimizing sum-of-squared residuals by gradient descent.

$$E = \sum_k (\text{Out}(x_k) - y_k)^2$$

$$= \sum_k (w^T x_k - y_k)^2$$

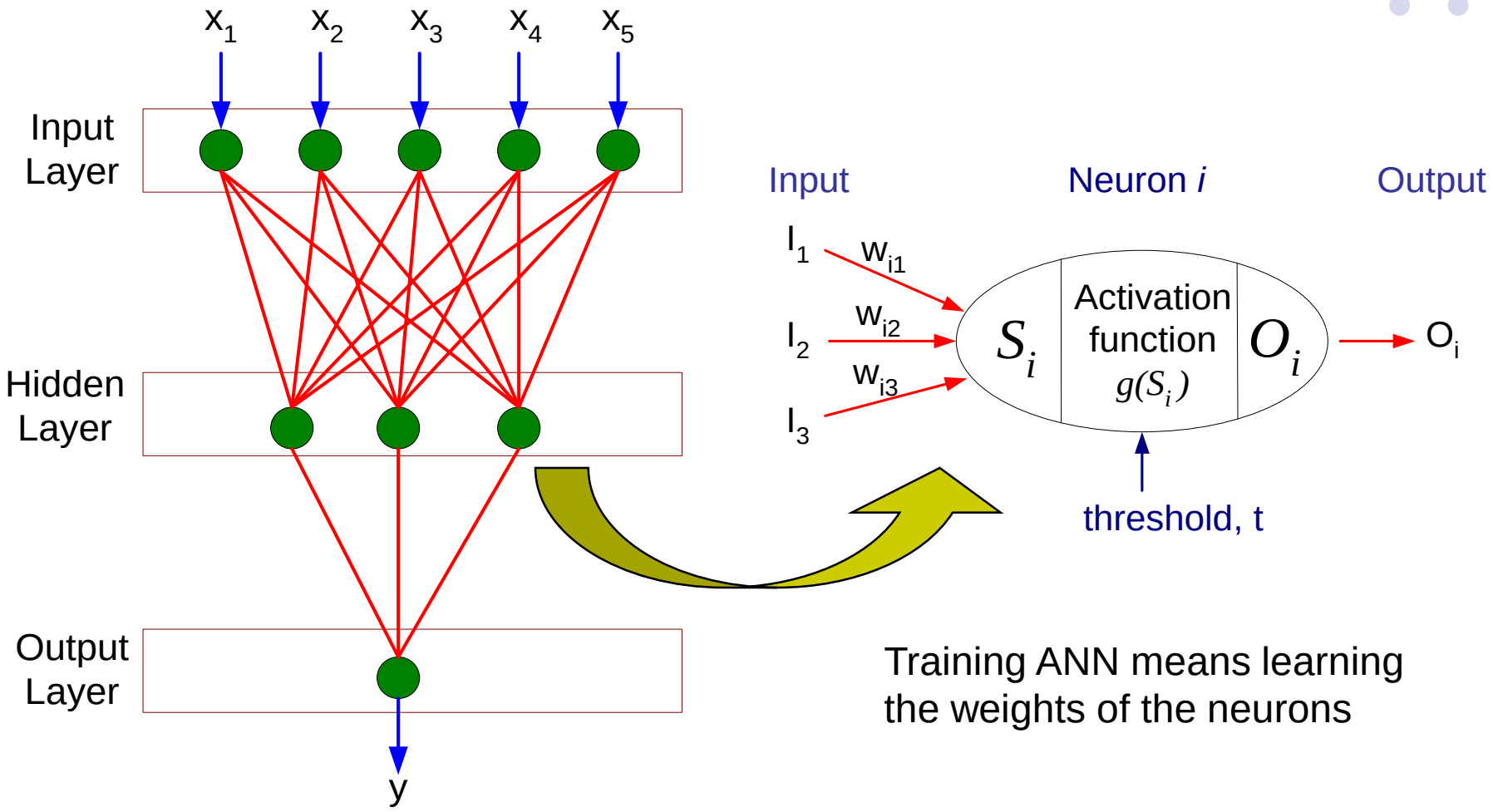
QUESTION: Derive the perceptron training rule.

A Multi-Layer Feed-Forward Neural Network



$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$

General Structure of ANN



Training ANN means learning the weights of the neurons