ANN - Artificial Neural Network

Introduction

- The term "Artificial Neural Network" is derived from Biological neural networks that develop the structure of a human brain. Similar to the human brain that has neurons interconnected to one another, artificial neural networks also have neurons that are interconnected to one another in various layers of the networks. These neurons are known as nodes.

Figure Biological Neural Network

- Typical diagram of Biological Neural Network

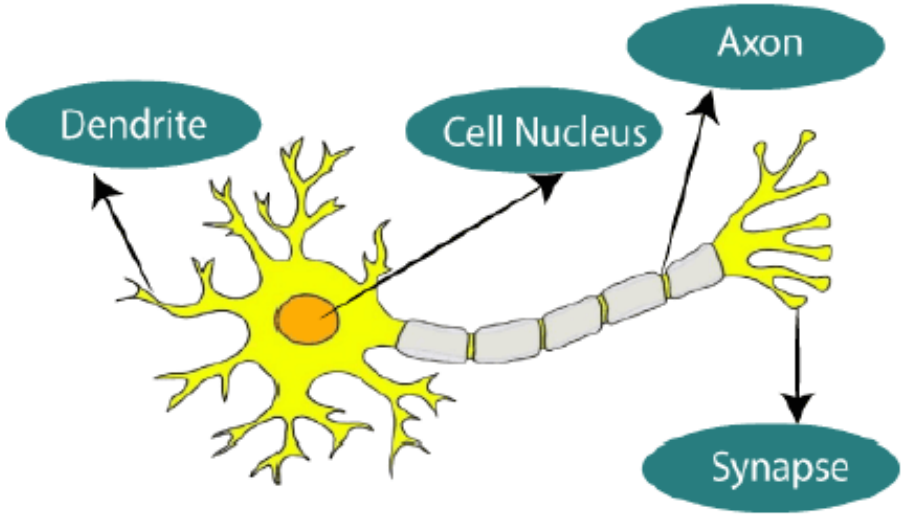
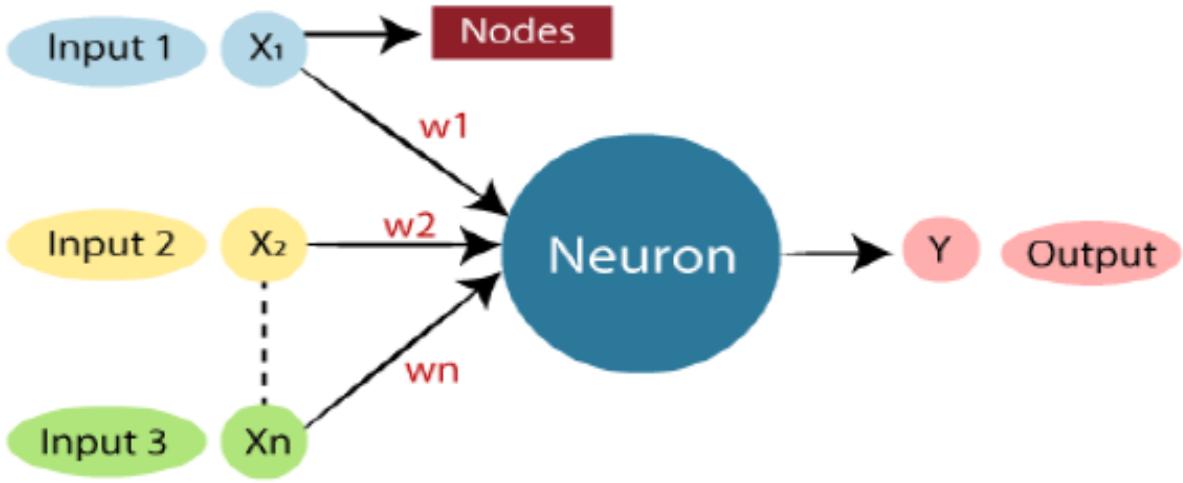


Figure ANN

Typical Artificial Neural Network looks something like the given figure



Dendrites from Biological Neural Network represent inputs in Artificial Neural Networks, cell nucleus represents Nodes, synapse represents Weights, and Axon represents Output.

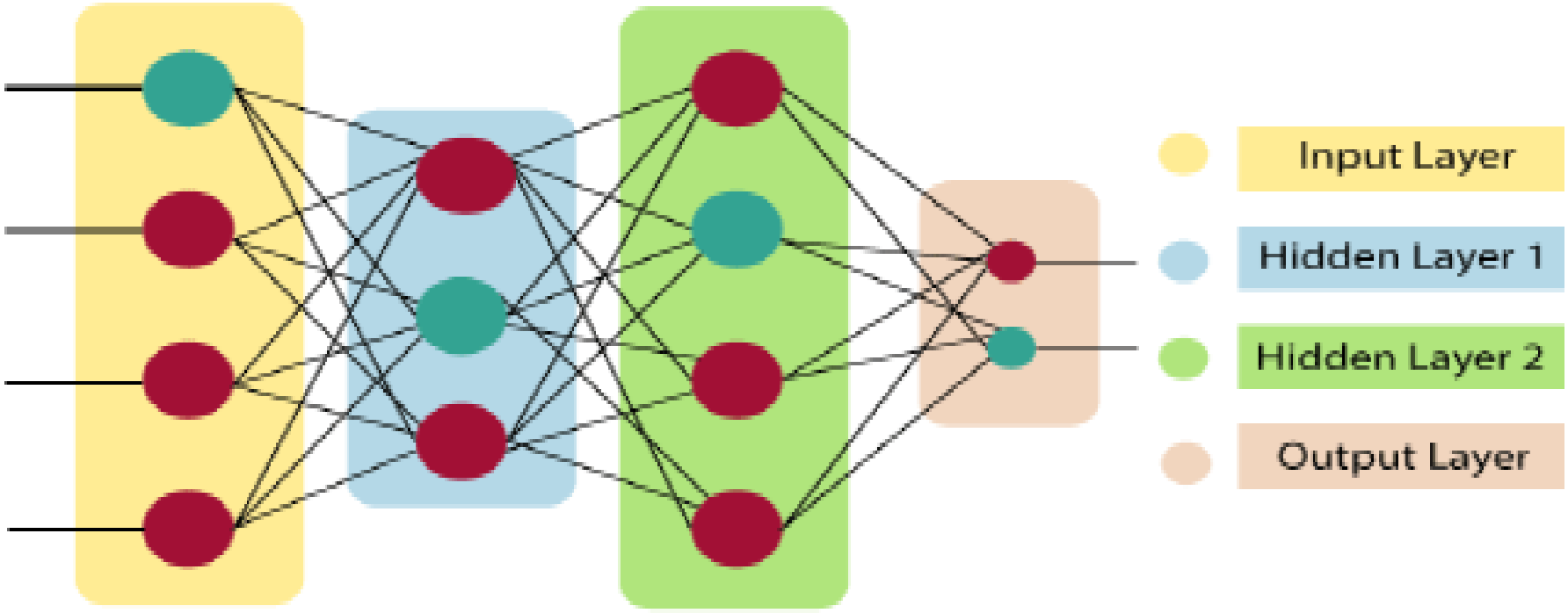
Relationship between biological Neural Network and Artificial neural network

- Relationship between biological Neural Network and Artificial neural network

Biological Neural Network	Artificial Neural Network
Dendrites	Inputs
Cell nucleus	Nodes
Synapse	Weights
Axon	Output

- An **Artificial Neural Network** in the field of **Artificial intelligence** where it attempts to mimic the network of neurons makes up a human brain so that computers will have an option to understand things and make decisions in a human-like manner. The artificial neural network is designed by programming computers to behave simply like interconnected brain cells. There are around 1000 billion neurons in the human brain. Each neuron has an association point somewhere in the range of 1,000 and 100,000. In the human brain, data is stored in such a manner as to be distributed.

Architecture of an artificial neural network



Architecture of an artificial neural network

- **Input Layer:** As the name suggests, it accepts inputs in several different formats provided by the programmer.
- **Hidden Layer:** The hidden layer presents in-between input and output layers. It performs all the calculations to find hidden features and patterns.
- **Output Layer:** The input goes through a series of transformations using the hidden layer, which finally results in output that is conveyed using this layer.

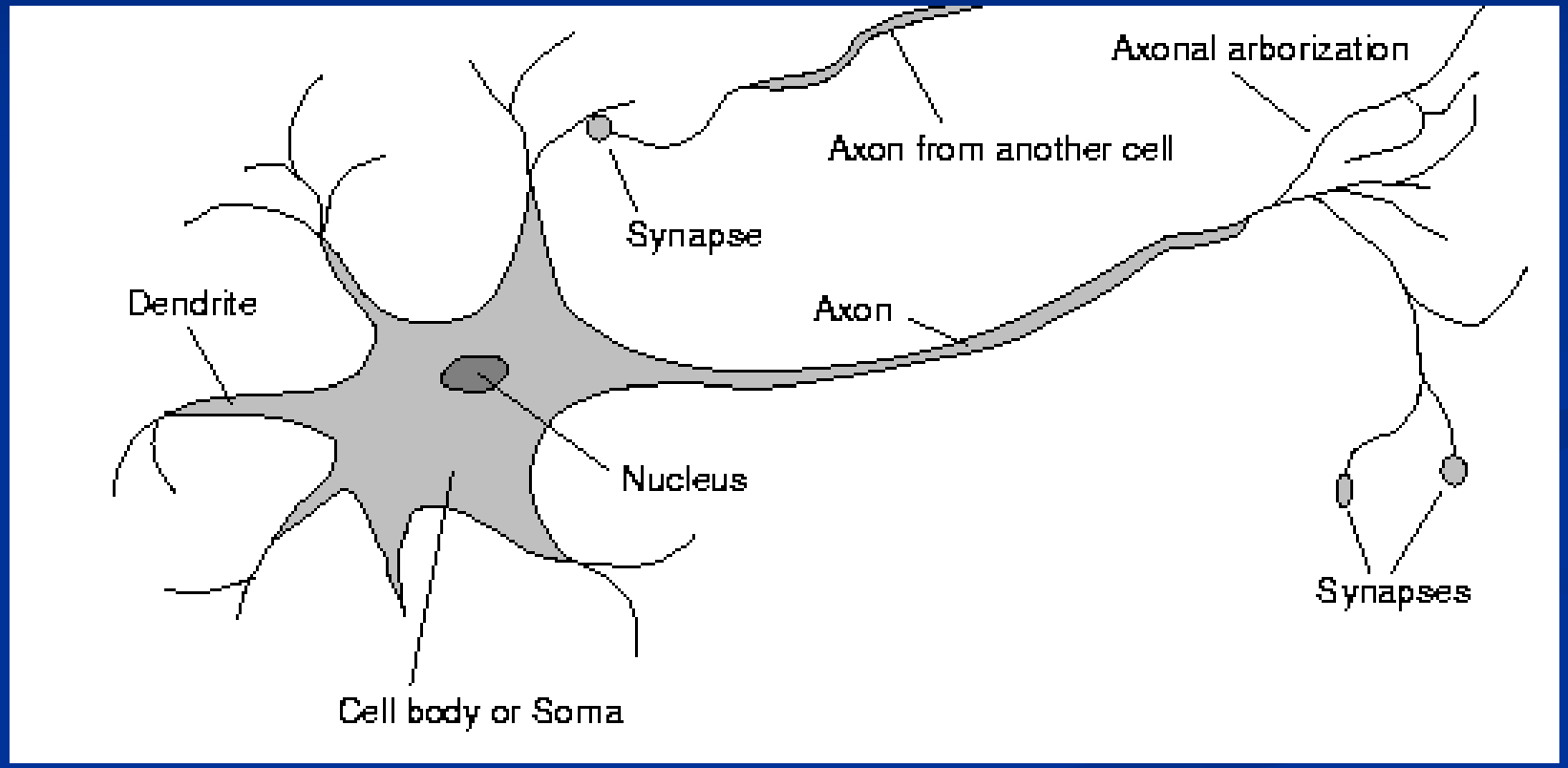
The artificial neural network takes input and computes the weighted sum of the inputs and includes a bias. This computation is represented in the form of a transfer function.

$$\sum_{i=1}^{n} Wi * Xi + b$$

Neural function

- Brain function (thought) occurs as the result of the firing of **neurons**
- Neurons connect to each other through **synapses**, which propagate **action potential** (electrical impulses) by releasing **neurotransmitters**
- Synapses can be **excitatory** (potential-increasing) or **inhibitory** (potential-decreasing), and have varying **activation thresholds**
- Learning occurs as a result of the synapses' **plasticicity**: They exhibit long-term changes in connection strength
- There are about 10^{11} neurons and about 10^{14} synapses in the human brain(!)

Biology of a neuron



Brain structure

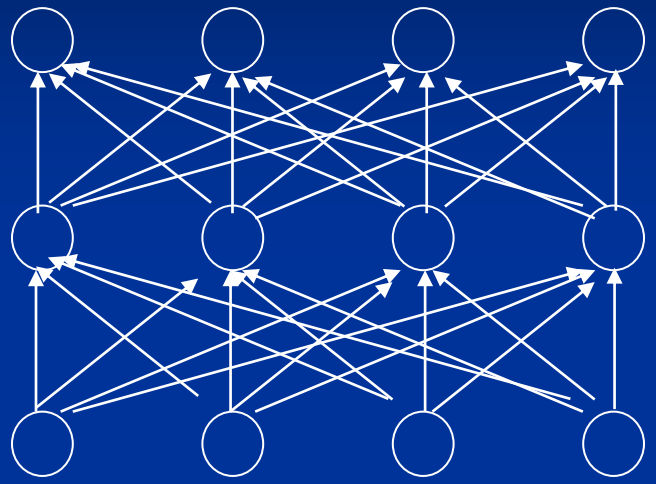
- Different areas of the brain have different functions
  - Some areas seem to have the same function in all humans (e.g., Broca's region for motor speech); the overall layout is generally consistent
  - Some areas are more plastic, and vary in their function; also, the lower-level structure and function vary greatly
- We don't know how different functions are "assigned" or acquired
  - Partly the result of the physical layout / connection to inputs (sensors) and outputs (effectors)
  - Partly the result of experience (learning)
- We *really* don't understand how this neural structure leads to what we perceive as "consciousness" or "thought"
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

Comparison of computing power

INFORMATION CIRCA 1995	Computer	Human Brain
Computation Units	1 CPU, 10^5 Gates	10^{11} Neurons
Storage Units	10^4 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec	10^{-3} sec
Bandwidth	10^4 bits/sec	10^{14} bits/sec
Updates / sec	10^5	10^{14}

- Computers are way faster than neurons…
- But there are a lot more neurons than we can reasonably model in modern digital computers, and they all fire in parallel
- Neural networks are designed to be massively parallel
- The brain is effectively a billion times faster

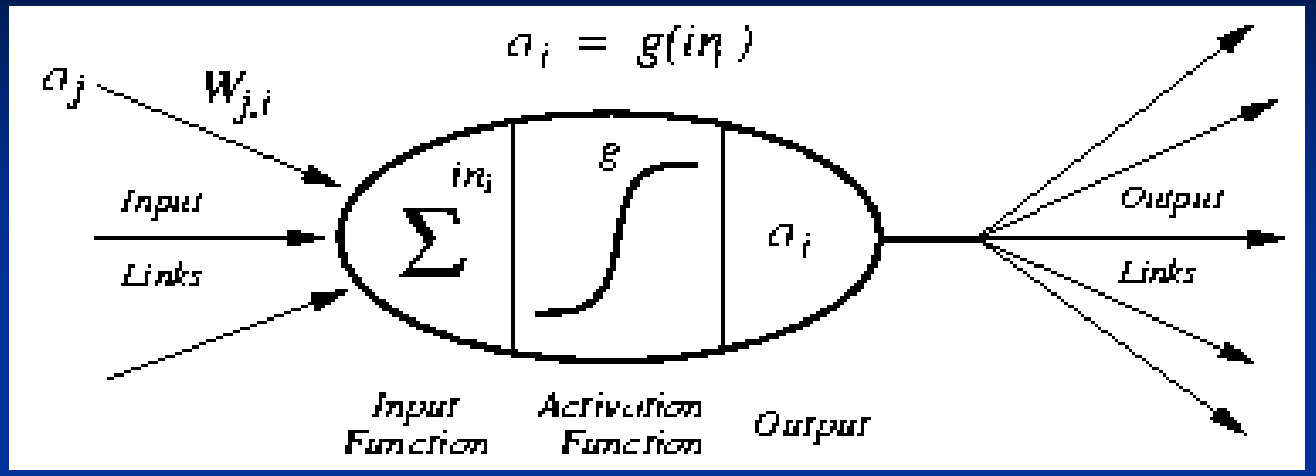
Neural networks



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Model of a neuron



- Neuron modeled as a unit i
- weights on input unit j to i, w_{ji}
- net input to unit i is:

$$in_i = \sum_j w_{ij} \cdot o_j$$

- Activation function g() determines the neuron's output
  - g() is typically a sigmoid
  - output is either 0 or 1 (no partial activation)

"Executing" neural networks

- Input units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level
- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node
- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Learning rules

- Rosenblatt (1959) suggested that if a target output value is provided for a single neuron with fixed inputs, can incrementally change weights to learn to produce these outputs using the *perceptron learning rule*
  - assumes binary valued input/outputs
  - assumes a single linear threshold unit

Perceptron learning rule

- If the target output for unit i is t_i

$$w_{ji} = w_{ji} + \eta(t_i - o_i)o_j$$

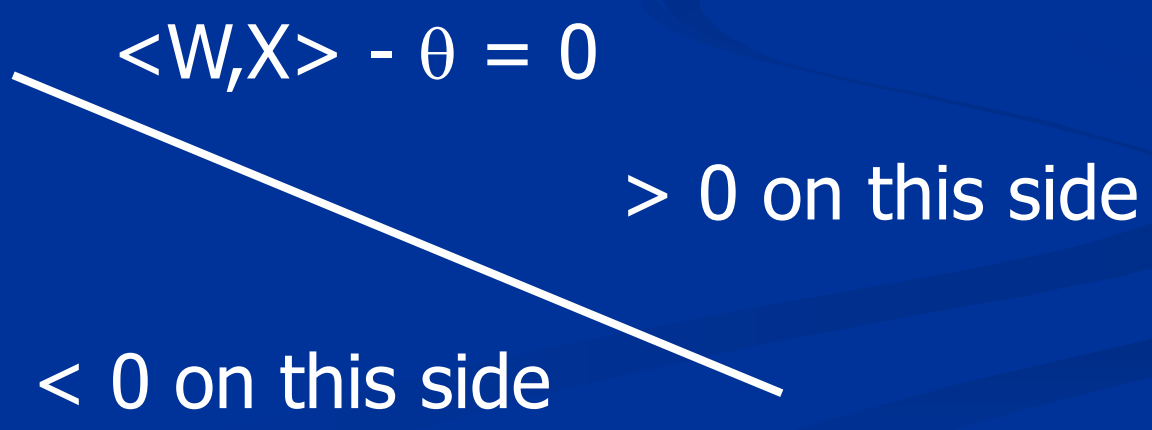
- Equivalent to the intuitive rules:
  - If output is correct, don't change the weights
  - If output is low (o_i=0, t_i=1), increment weights for all the inputs which are 1
  - If output is high (o_i=1, t_i=0), decrement weights for all inputs which are 1
  - Must also adjust threshold. Or equivalently assume there is a weight w_{0i} for an extra input unit that has an output of 1.

Perceptron learning algorithm

- Repeatedly iterate through examples adjusting weights according to the perceptron learning rule until all outputs are correct
  - Initialize the weights to all zero (or random)
  - Until outputs for all training examples are correct
    - for each training example e do
      - compute the current output o_j
      - compare it to the target t_j and update weights
- Each execution of outer loop is called an ***epoch***
- For multiple category problems, learn a separate perceptron for each category and assign to the class whose perceptron most exceeds its threshold

Representation limitations of a perceptron

- Perceptrons can only represent linear threshold functions and can therefore only learn functions which linearly separate the data.
  - i.e., the positive and negative examples are separable by a hyperplane in n-dimensional space



Perceptron learnability

- **Perceptron Convergence Theorem:** If there is a set of weights that is consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge (Minicksy & Papert, 1969)
- Unfortunately, many functions (like parity) cannot be represented by LTU

Learning: Backpropagation

- Similar to perceptron learning algorithm, we cycle through our examples
  - if the output of the network is correct, no changes are made
  - if there is an error, the weights are adjusted to reduce the error
- The trick is to assess the blame for the error and divide it among the contributing weights

Output layer

- As in perceptron learning algorithm, we want to minimize difference between target output and the output actually computed

$$W_{ji} = W_{ji} + \alpha \times a_j \times Err_i \times g'(in_i)$$

activation of hidden unit j

$(T_i - O_i)$

derivative of activation function

$$\Delta_i = Err_i \times g'(in_i)$$

$$W_{ji} = W_{ji} + \alpha \times a_j \times \Delta_i$$

Hidden layers

- Need to define error; we do error backpropagation.
- Intuition: Each hidden node j is "responsible" for some fraction of the error Δ_I in each of the output nodes to which it connects.
- Δ_I divided according to the strength of the connection between hidden node and the output node and propagated back to provide the Δ_j values for the hidden layer:

$$\Delta_j = g'(in_j)\sum_j W_{ji}\Delta_i$$

update rule: $W_{kj} = W_{kj} + \alpha \times I_k \times \Delta_j$

Backprogation algorithm

- Compute the Δ values for the output units using the observed error
- Starting with output layer, repeat the following for each layer in the network, until earliest hidden layer is reached:
  - propagate the Δ values back to the previous layer
  - update the weights between the two layers

Backprop issues

- "Backprop is the cockroach of machine learning. It's ugly, and annoying, but you just can't get rid of it."

  –Geoff Hinton

- Problems:
  - black box
  - local minima