

Unit III – Neural Network

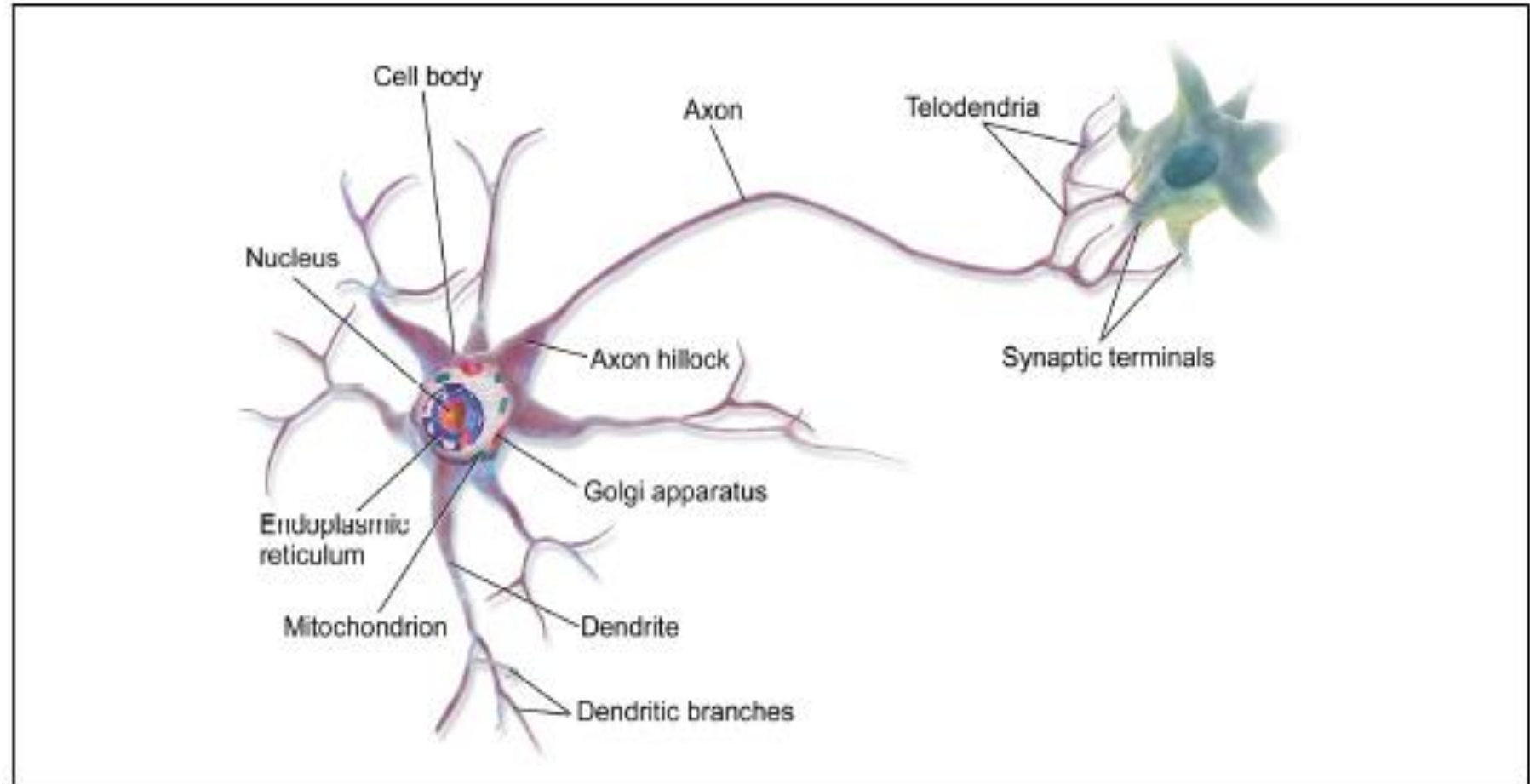
Artificial Intelligence

School of Cyber Security & Digital Forensics

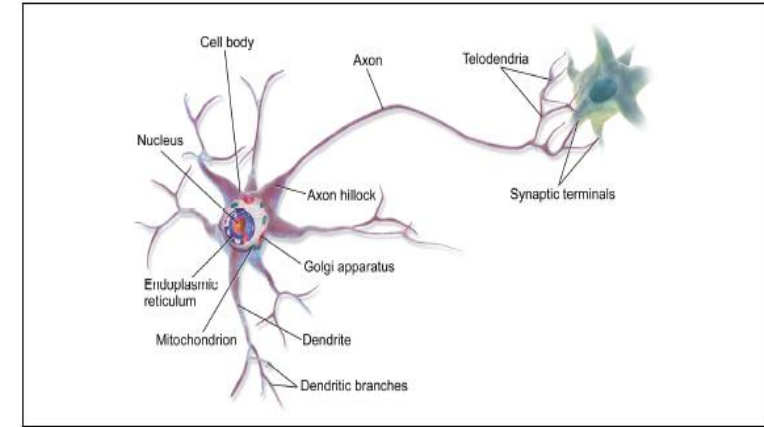
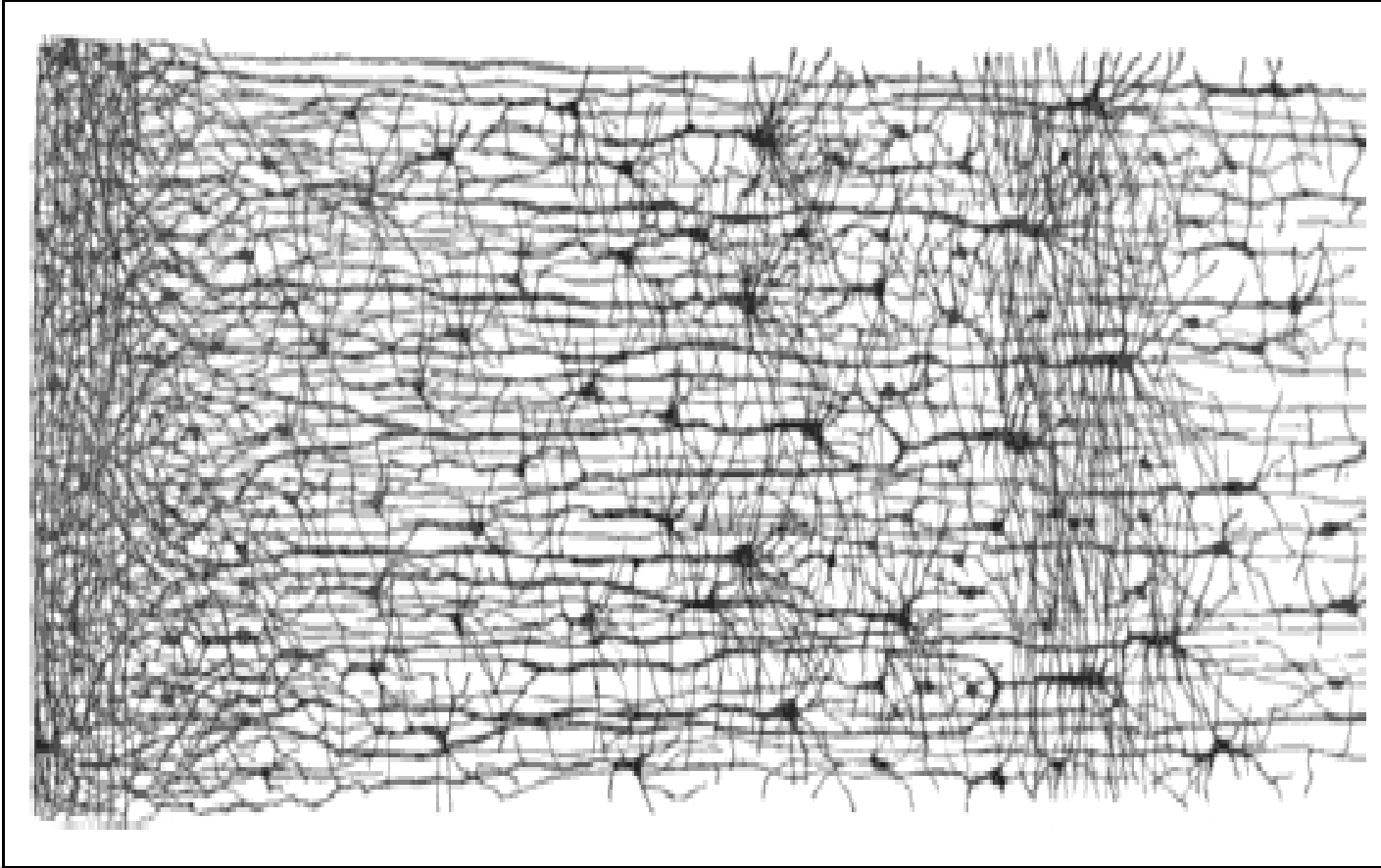
M. Sc. Cyber Security (Semester-I)

Biological Neuron

- **Dendrites**
- **Axon**
- **Telodendria**
- **Synaptic terminals**



Multiple layers in biological Neural Network (NN)



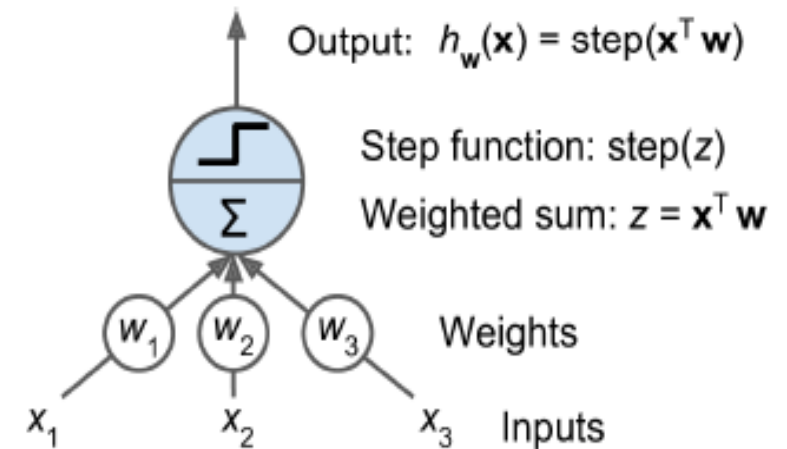
Artificial Neuron

- Simplest neural network is referred to as the perceptron invented in 1957 by Frank Rosenblatt
- Based on artificial neuron called Threshold Logic Unit (TLU)
- Each input associated with weight
- The TLU computes a weighted sum of its inputs

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = \mathbf{x}^T \mathbf{w},$$

then applies a *step function* to that sum and outputs the result:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{step}(z), \text{ where } z = \mathbf{x}^T \mathbf{w}.$$



Step function

- Heaviside step function

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

- Sign step function

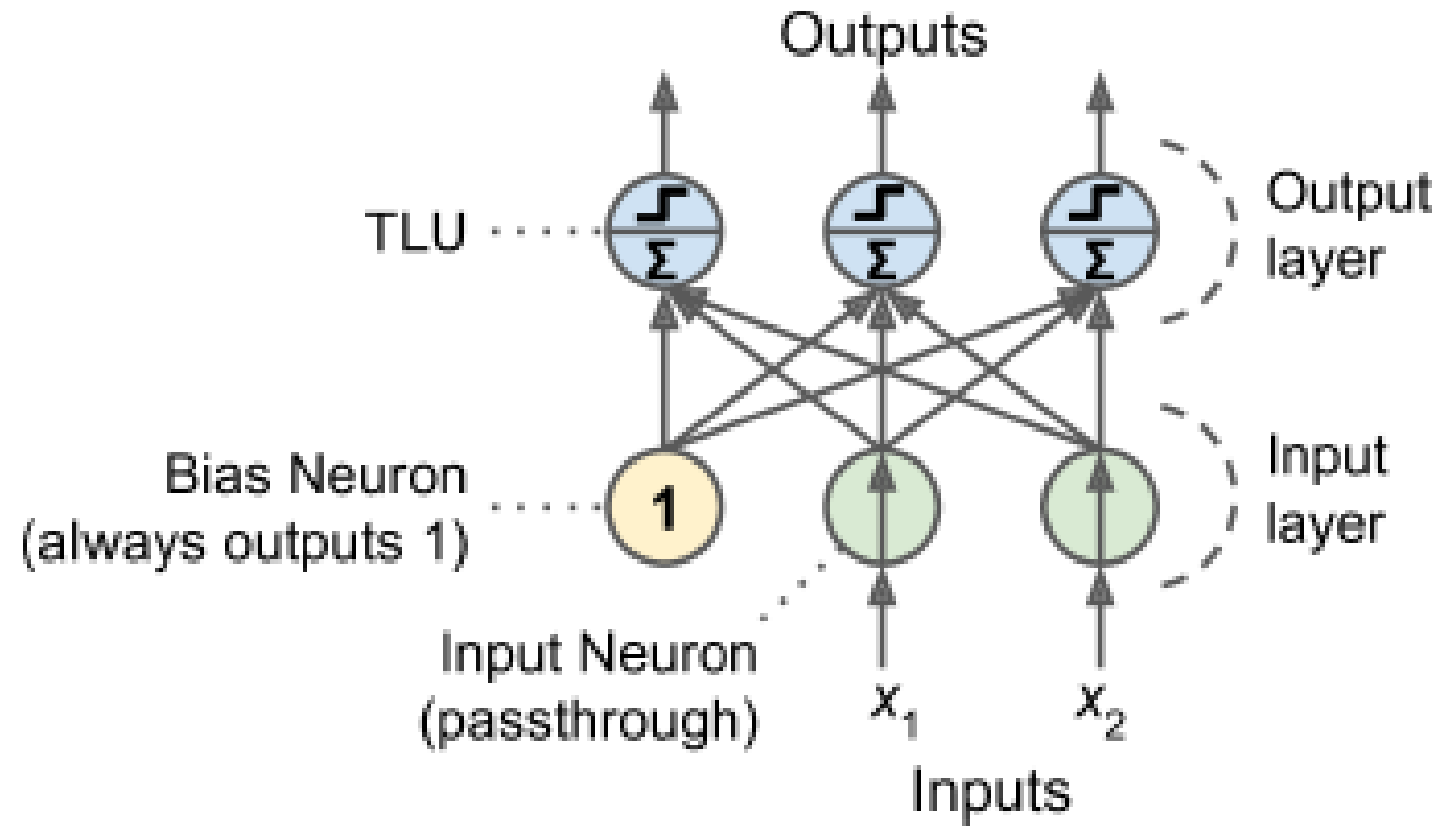
$$\text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

- Single TLU can be used in simple linear binary classification

Perceptron

- A Perceptron is simply composed of a single layer of TLUs with each TLU connected to all the inputs
- When all the neurons in a layer are connected to every neuron in the previous it is called a **fully connected layer** or a **dense layer**.
- All the input neurons form the input layer: output whatever input they are fed
- Moreover, an extra bias feature is generally added ($x_0 = 1$): it is typically represented using a special type of neuron called a bias neuron, which just outputs 1 all the time.

Perceptron architecture



Computing of a fully connected layer :Linear Algebra

$$h_{\mathbf{W}, \mathbf{b}}(\mathbf{X}) = \phi(\mathbf{XW} + \mathbf{b})$$

- \mathbf{X} represents the matrix of input features. It has one row per instance, one column per feature.
- The weight matrix \mathbf{W} contains all the connection weights except for the ones from the bias neuron. It has one row per input neuron and one column per artificial neuron in the layer.
- The bias vector \mathbf{b} contains all the connection weights between the bias neuron and the artificial neurons. It has one bias term per artificial neuron.
- The function ϕ is called the activation function: when the artificial neurons are TLUs, it is a step function

Training a Perceptron

- Perceptron is fed one training instance at a time, and for each instance it makes its predictions.
- For every output neuron that produced a wrong prediction, it reinforces the connection weights from the inputs that would have contributed to the correct prediction.
- *Perceptron learning rule (weight update)*

Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta (y_j - \hat{y}_j) x_i$$

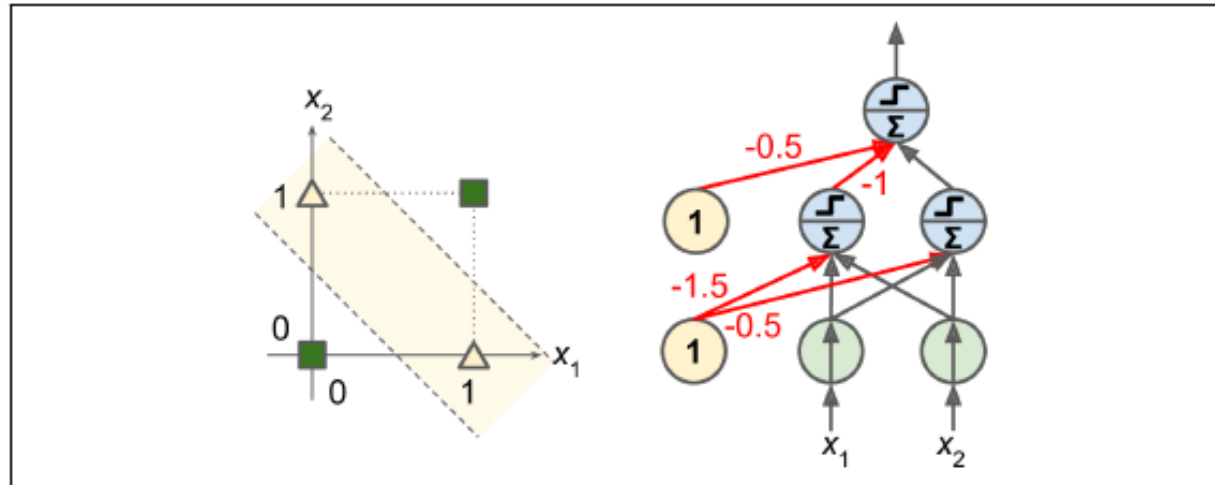
- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- \hat{y}_j is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.
- Example of and gate

Perceptron

- The decision boundary of each output neuron is linear, so perceptrons are incapable of learning complex patterns
- Contrary to Logistic Regression classifiers, Perceptrons do not output a class probability; rather, they just make predictions based on a hard threshold.
- Perceptrons, are incapable of solving some trivial problems e.g., the Exclusive OR (XOR) classification problem

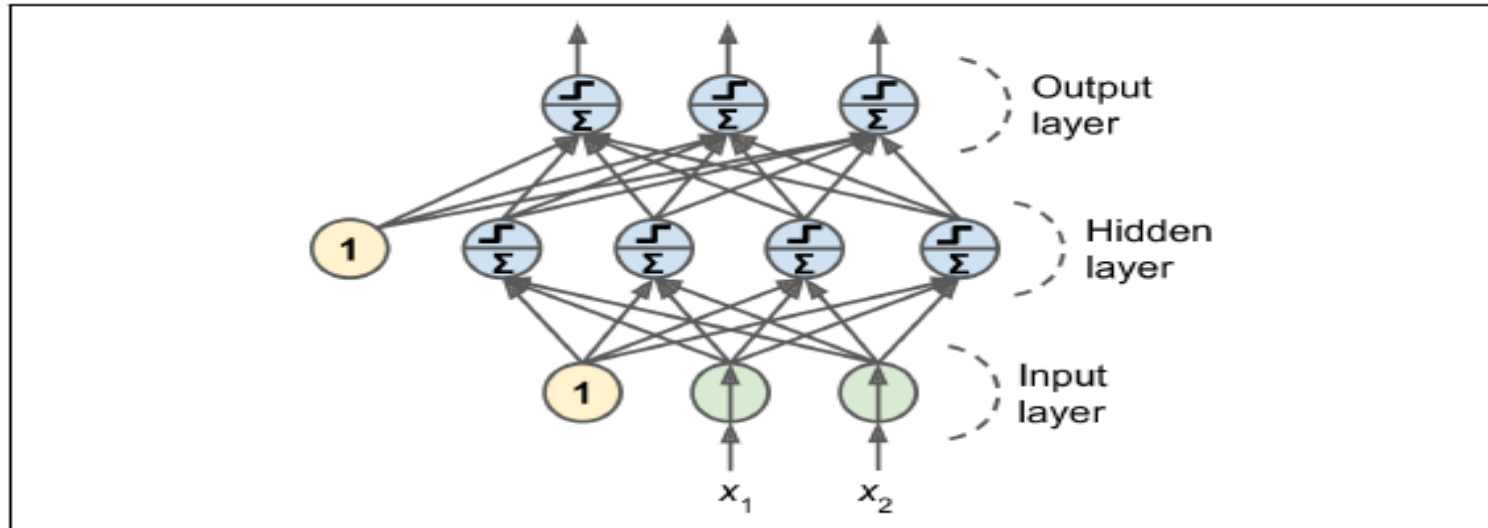
Towards Multi-layer Perceptron

- The limitations of Perceptrons can be eliminated by stacking multiple Perceptrons. The resulting ANN is called a *Multi-Layer Perceptron* (MLP).
- In particular, an MLP can solve the XOR problem



Multi-layer Perceptron

- An MLP is composed of *input layer*, one or more layers of TLUs, called *hidden layers*, and one final layer of TLUs called the *output layer*



- The layers close to the input layer : **lower layers**, and close to the outputs : **upper layers**.
- Every layer except the output layer includes a bias neuron and is fully connected to the next layer.

Backpropagation

- The signal flows only in one direction (from the inputs to the outputs), so this architecture is an example of a *feedforward neural network* (FNN).
- Training of MLPs is done using **backpropagation** algorithm, introduced by David Rumelhart, Geoffrey Hinton and Ronald Williams in 1986
- With two passes through the network (one forward, one backward), the backpropagation algorithm compute the gradient of the network's error with regards to every single model parameter.
- Once it has these gradients, it just performs a regular Gradient Descent step, and the whole process is repeated until the network converges to the solution.

Backpropagation Algorithm

- It handles one mini-batch at a time and it goes through the full training set multiple times.
Each pass is called an epoch
- Each mini-batch is passed to the network's input layer, which just sends it to the first hidden layer. The algorithm then computes the output of all the neurons in this layer (for every instance in the mini-batch). The result is passed on to the next layer, its output is computed and passed to the next layer, and so on until we get the output of the last layer, the output layer. This is the **forward pass**: it is exactly like making predictions, except all intermediate results are preserved since they are needed for the backward pass.

Backpropagation Algorithm...contd.

- Next, the algorithm measures the network's output error (i.e., it uses a loss function that compares the desired output and the actual output of the network, and returns some measure of the error).
- Then it computes how much each output connection contributed to the error. This is done analytically by simply applying the chain rule
- The algorithm then measures how much of these error contributions came from each connection in the layer below, again using the chain rule—and so on until the algorithm reaches the input layer. This **reverse pass** efficiently measures the error gradient across all the connection weights in the network by **propagating the error gradient backward** through the network
- Finally, the algorithm performs a **Gradient Descent step** to tweak all the connection weights in the network, using the error gradients it just computed

Backpropagation ...key change w.r.t perceptron

- A key change to the MLP's architecture: besides adding hidden layer, they replaced the step function with the logistic function, $\sigma(z) = 1 / (1 + \exp(-z))$.
- This was essential because the step function contains only flat segments, so there is no gradient to work with (Gradient Descent cannot move on a flat surface), while the logistic function has a well-defined nonzero derivative everywhere, allowing Gradient Descent to make some progress at every step.

Regression MLP

- MLPs can be used for regression tasks
- If you want to **predict a single value** then you just **need a single output neuron**: its output is the predicted value. To predict **multiple values at once**, you need **one output neuron per output dimension**.
- In general, when building an MLP for regression, you do not want to use any activation function for the output neurons, so they are free to output any range of values or can use based on requirements.

Regression MLP

Table 10-1. Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Classification MLP

- MLPs can also be used for classification tasks
- For a binary classification problem, you just need a single output neuron using the logistic activation function: the output will be a number between 0 and 1, which you can interpret as the estimated probability of the positive class.
- **Multilabel binary classification** need two output neurons, both using the **logistic activation function**. The output probabilities do not necessarily add up to one.
- **Multiclass classification** need to have one output neuron per class, and use the softmax activation function for the whole output layer that will ensure that all the estimated probabilities are between 0 and 1 and that they add up to one

Classification MLP

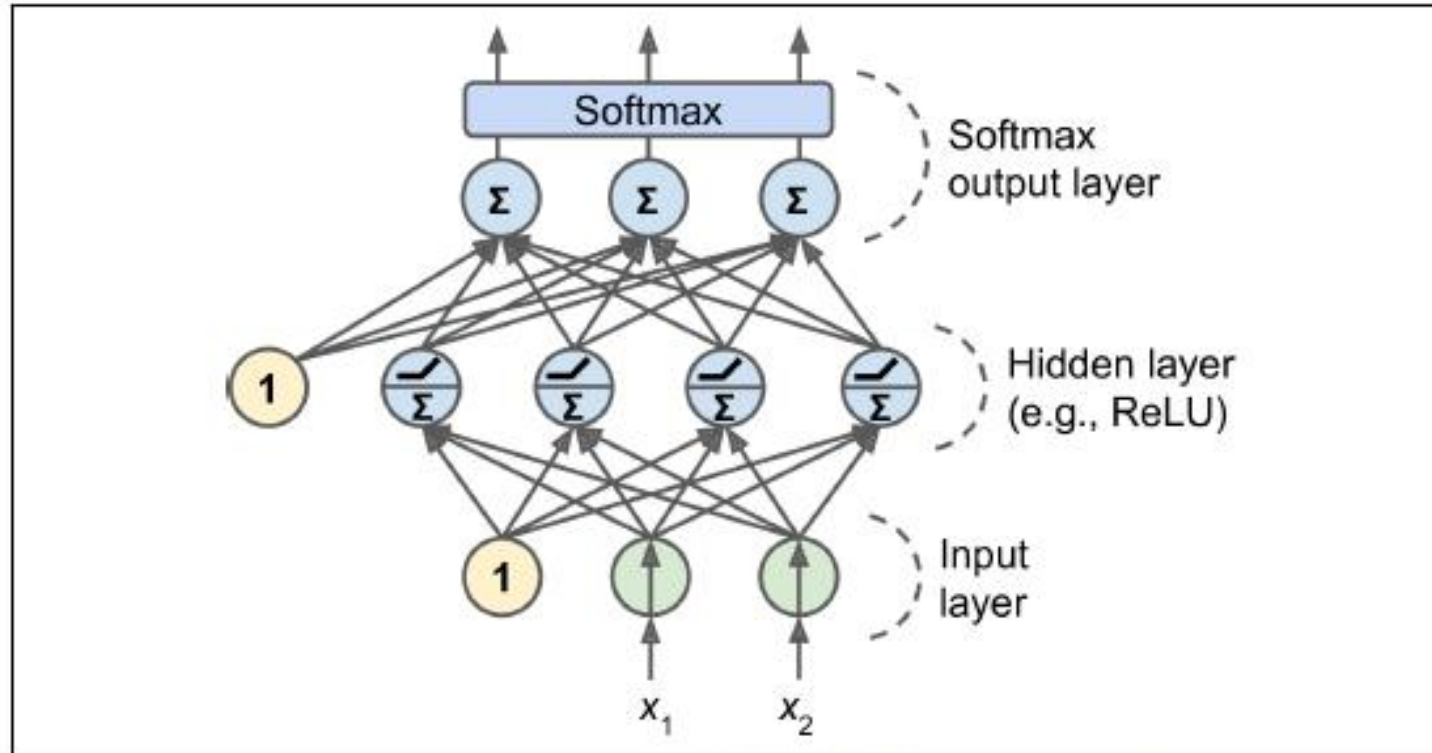


Figure 10-9. A modern MLP (including ReLU and softmax) for classification

Classification MLP

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

Few terminologies

- Batch size - The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

Batch Gradient Descent Batch Size = Size of Training Set

Stochastic Gradient Descent Batch Size = 1

Mini-Batch Gradient Descent $1 < \text{Batch Size} < \text{Size of Training Set}$.

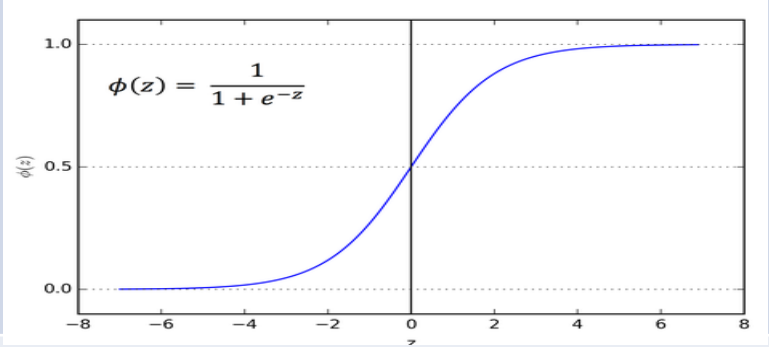
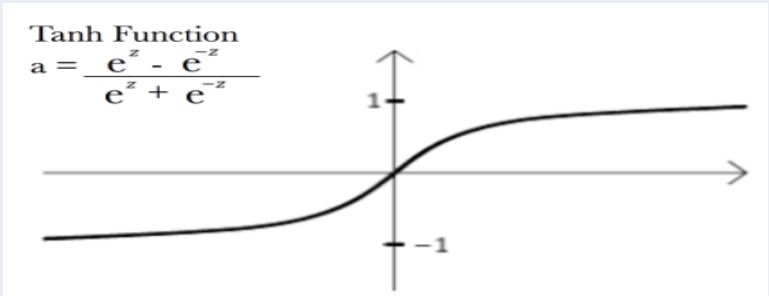
- Epoch – One pass through the entire training set.

The number of epochs is the number of complete passes through the training dataset.

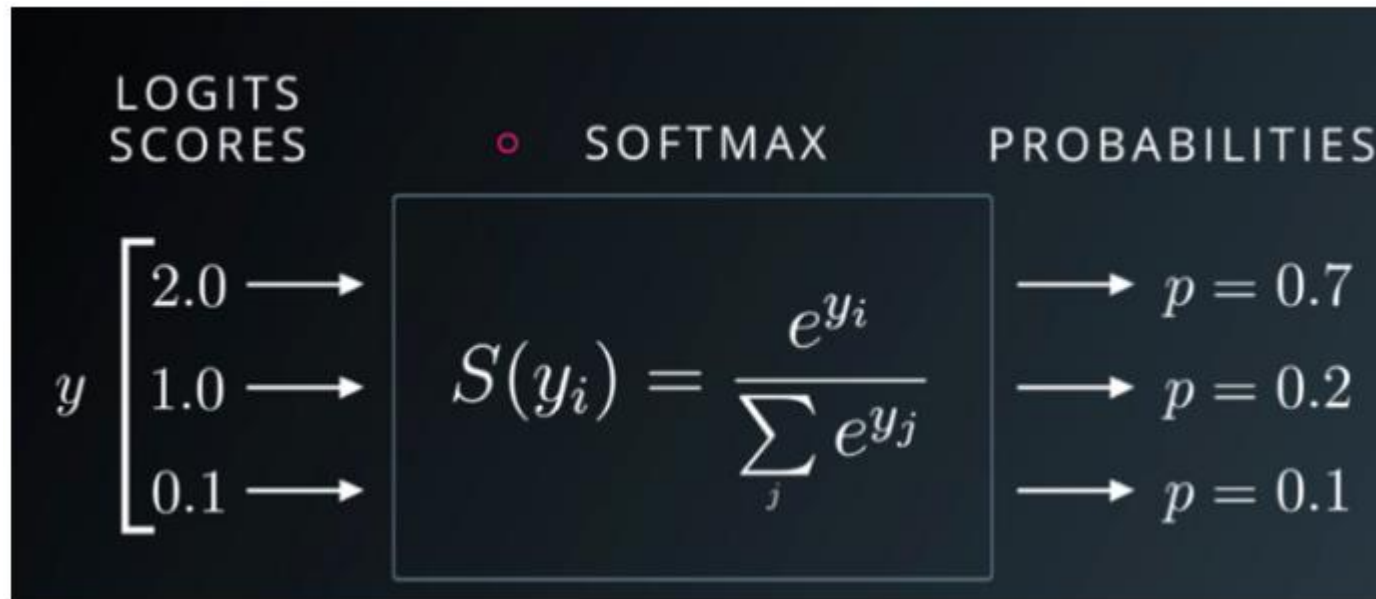
Activation Function

- Activation functions map a neuron's inputs to that of the next layer.
- **Need for an Activation function**
 1. **Restricting value:** The activation function keeps the values from the node restricted within a certain range, because they can become infinitesimally small or huge depending on the multiplication or other operations they go through in various layers (i.e the [vanishing](#) and [exploding](#) gradient problem).
 2. **Add non-linearity:** In the absence of an activation function, the operations done by various functions can be considered as stacked over one another, which ultimately means a linear combination of operations performed on the input. Thus, a neural network without an activation function is essentially a linear regression model.

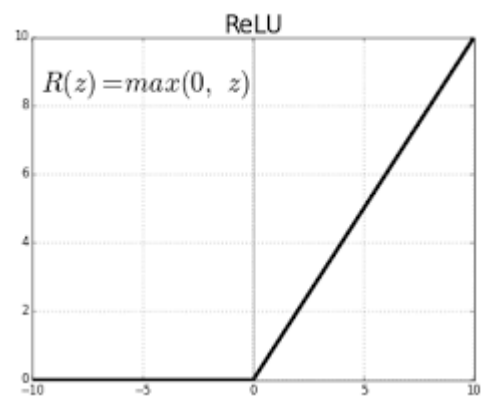
Activation Function

Name	Formula	Range	Plot
Sigmoid	$\frac{1}{1 + e^{-z}}$	(0,1)	
Tanh Function	$\frac{e^z - e^{-z}}{e^z + e^{-z}}$	(-1,1)	
Softmax (not to be used in hidden as neurons should be independent)	$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$ <p>K – no of classes J – variable o/p neuron</p>	(0,1)	NA

Activation Function: Softmax Function



Activation Function

Name	Formula	Range	Plot
ReLU (rectified linear activation function)	$\max(0, x)$	$(0, \infty)$	

Loss Function

- With neural networks, we seek to minimize the error (difference between actual and predicted value) which is calculated by the loss function.

Name	Formula	Task
Mean Squared Error	$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	Regression
Mean Absolute Error	$\frac{1}{n} \sum_{i=1}^n y_i - \hat{y}_i $	Regression
Binary Cross Entropy	$-\frac{1}{n} \sum_{i=1}^n y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$	Classification
Cross Entropy	$-\frac{1}{n} \sum_{i=1}^n y_i \log(\hat{y}_i)$	Classification