

Unit 4

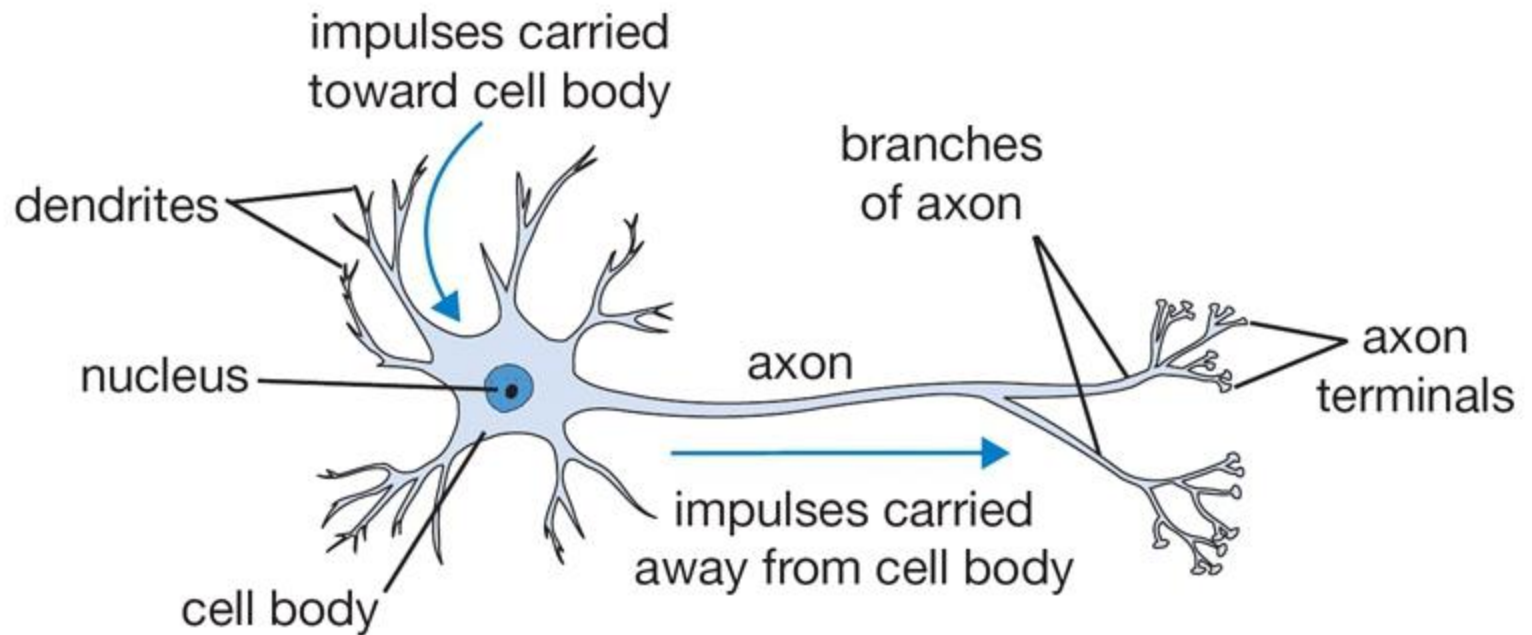
Artificial Neural Networks

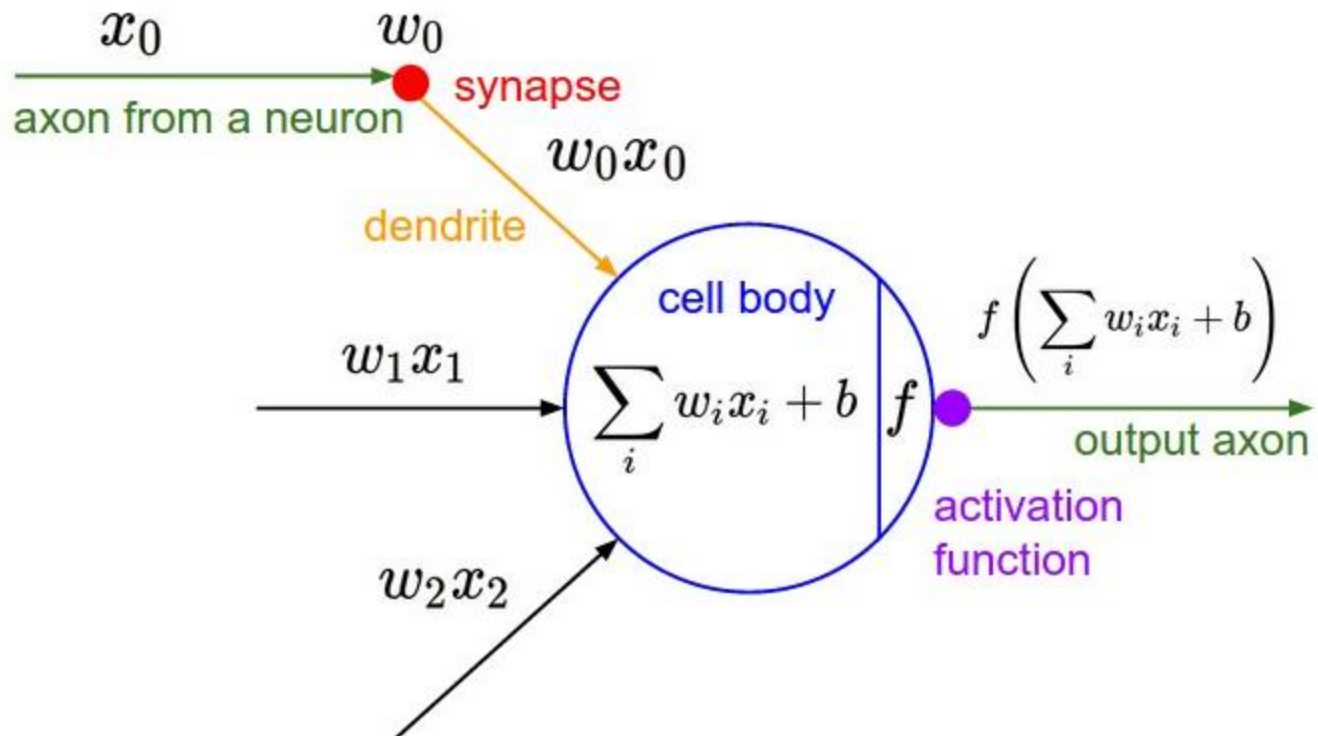
Artificial Neural Networks

- Human Cognitive Learning
- Perceptron
- Sigmoid neuron
- NN Architecture
- Supervised Vs Unsupervised NN
- NN Learning Algorithms
- Feed Forward
- Back Propagation
- Deep Learning

Human Cognitive Learning

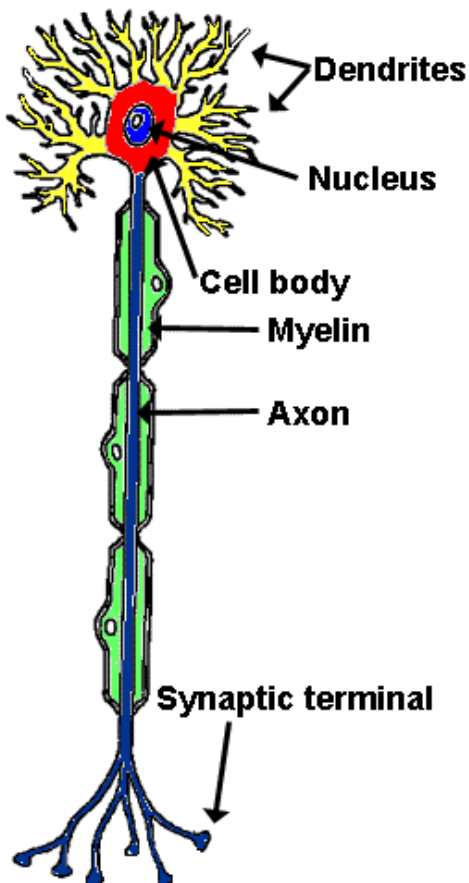
- Cognitive Science – Higher order thinking
- Learning focuses on reasoning and understanding at higher level
- Biological approach to AI
- Developed in 1943
- Understanding the mind
- The networks are represented by interconnected neurons
- They send messages to each other
- Comprised of one or more layers of neurons



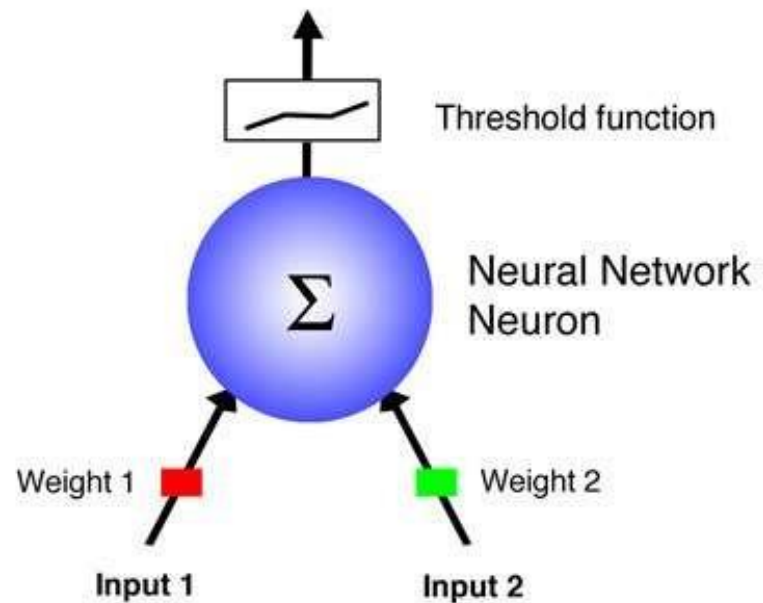


Human Cognitive Learning

Biological



Artificial



Understanding the NN

- Neurons with synapses connecting them
- Input layer, hidden layer, output layer
- N- hidden layers – Deep Learning
- Hidden layers are increased when the system is too complex to understand and learn
- Synapses take the input and multiply by a weight
- Neurons add all the output from all synapses and apply an activation function

Neural Networks

- A NN is a machine learning approach inspired by the way in which the brain performs a particular learning task:
 - Knowledge about the learning task is given in the form of examples.
 - Inter neuron connection strengths (**weights**) are used to **store** the acquired **information (the training examples)**.
 - During the **learning process** the weights are modified in order to model the particular learning task correctly on the training examples.

Types of Neural Networks

Neural Network types can be classified based on following attributes:

- Connection Type

- Static (feedforward)
- Dynamic (feedback)

- Topology

- Single layer
- Multilayer
- Recurrent

- Learning Methods

- Supervised
- Unsupervised
- Reinforcement

Feed forward

- **Feed forward Network** - wherein connections between the nodes do *not* form a cycle.
- Simplest type
- Information moves in one direction
- From input nodes to hidden nodes and then to output
- There is no cycles or loop in the network
- Input is the matrix and the output is the target_vector

Back Propagation

- **Feed backward Network** - Back propagation
- Back propagation is a training algorithm consisting of 2 steps:
 - Feed forward the values
 - Calculate the error and propagate it back to the earlier layers
- Input for back propagation is output_vector, target_output_vector, output is adjusted_weight_vector.

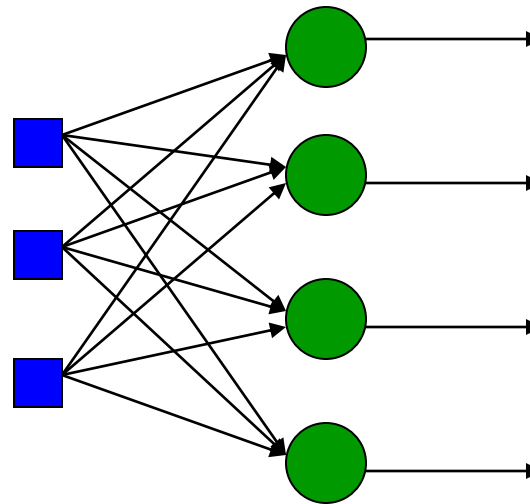
Network architectures

- Three different classes of network architectures
 - single-layer feed-forward
 - multi-layer feed-forward
 - recurrent

} neurons are organized in acyclic layers
- The **architecture** of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

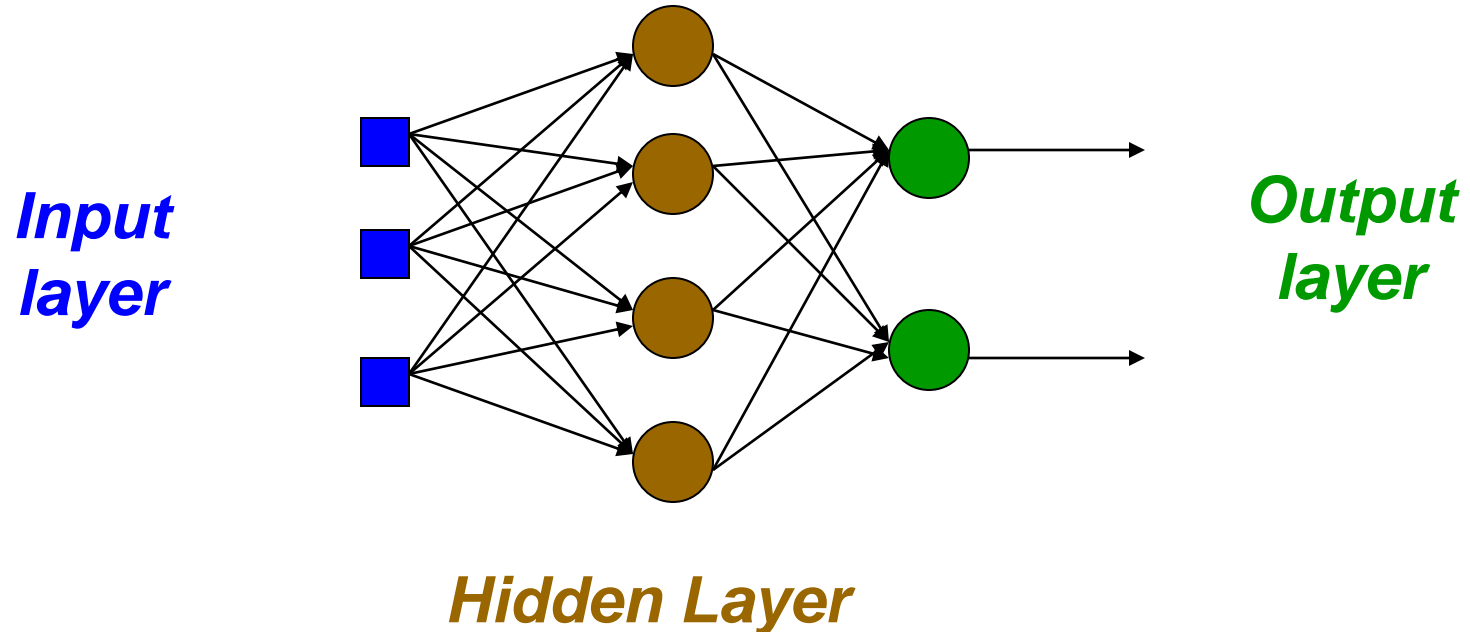
*Input layer
of
source nodes*



*Output layer
of
neurons*

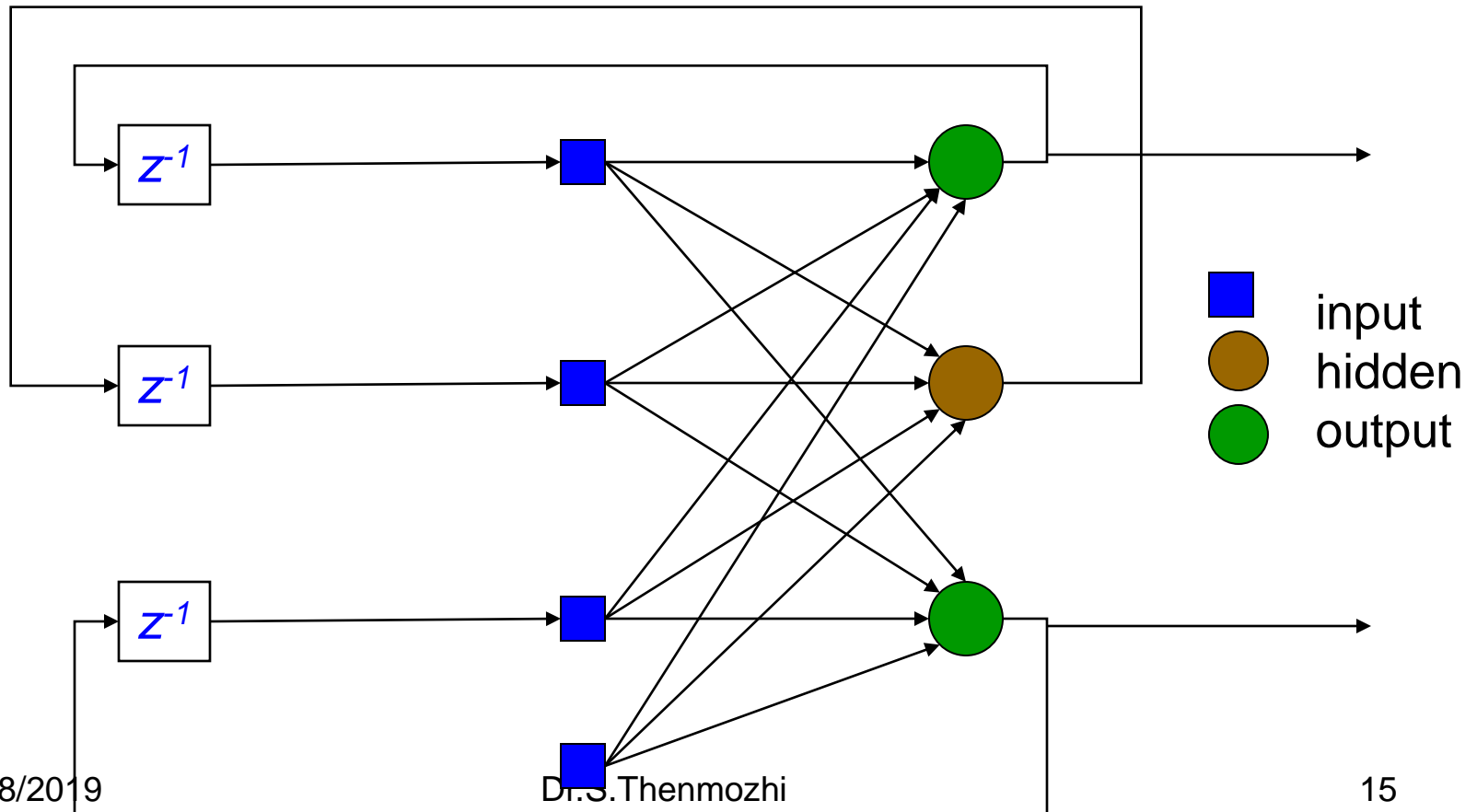
Multi layer feed-forward

3-4-2 Network

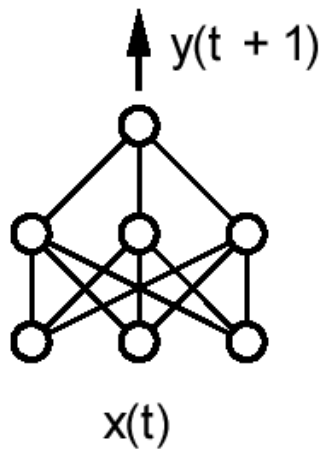


Recurrent network

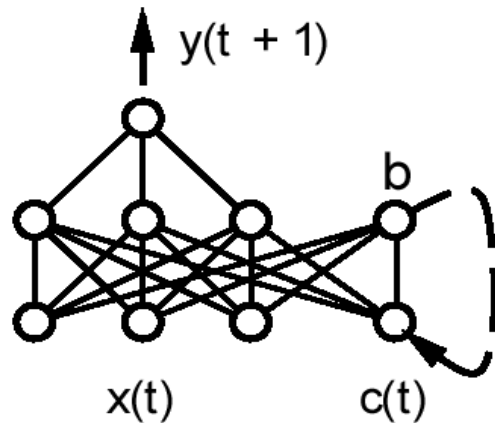
Recurrent Network with *hidden neuron(s)*: unit delay operator z^{-1} implies dynamic system



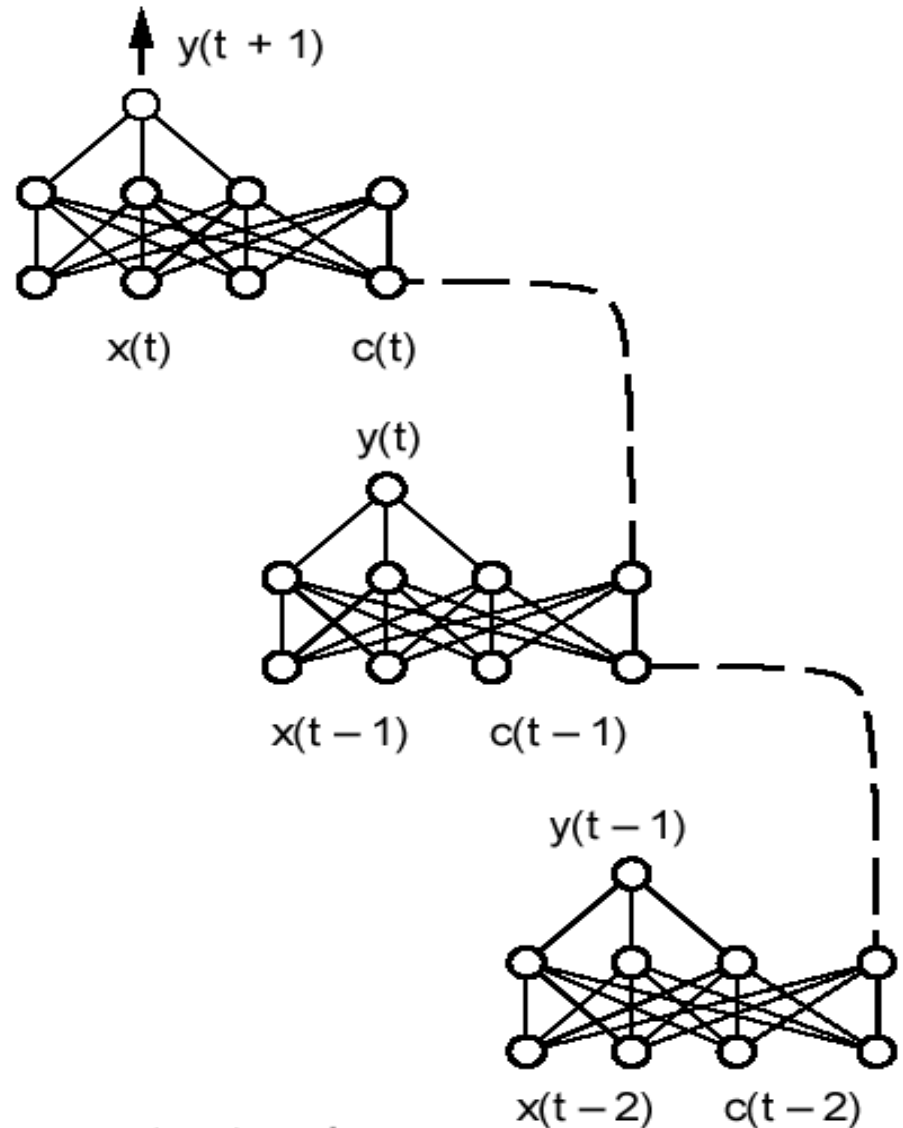
Neural Network Architectures



(a) Feedforward network



(b) Recurrent network



(c) Recurrent network
unfolded in time

Learning Methods

Supervised NN

- Each training pattern: input + desired output
- At each presentation: adapt weights
- After many epochs convergence to a local minimum

UnSupervised NN

- In training data, no information available on the desired output
- *Learning by doing*
- Used to pick out structure in the input:
 - Clustering, Reduction of dimensionality, Compression
- *Example: Kohonen's Learn*

Learning Methods



Reinforcement NN

- Use performance score to shuffle weights randomly
- Relatively slow learning due to randomness

The Neuron

- The neuron is the basic information processing unit of a NN. It consists of:
 - 1 A set of **synapses** or **connecting links**, each link characterized by a **weight**:

$$W_1, W_2, \dots, W_m$$

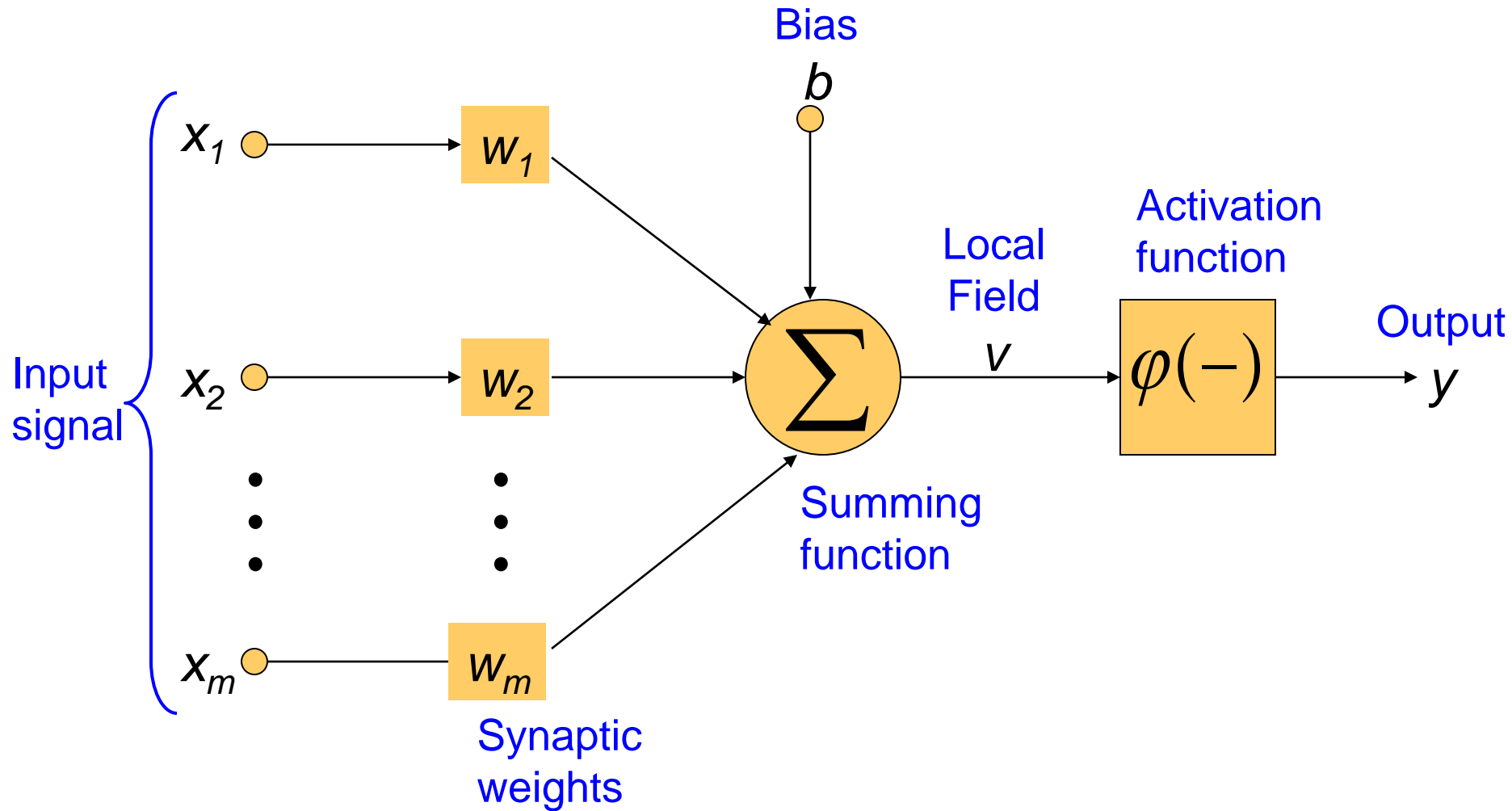
- 2 An **adder** function (linear combiner) which computes the weighted sum of the inputs:

$$u = \sum_{j=1}^m w_j x_j$$

- 3 **Activation function** (squashing function) φ for limiting the amplitude of the output of the neuron.

$$y = \varphi(u + b)$$

The Neuron

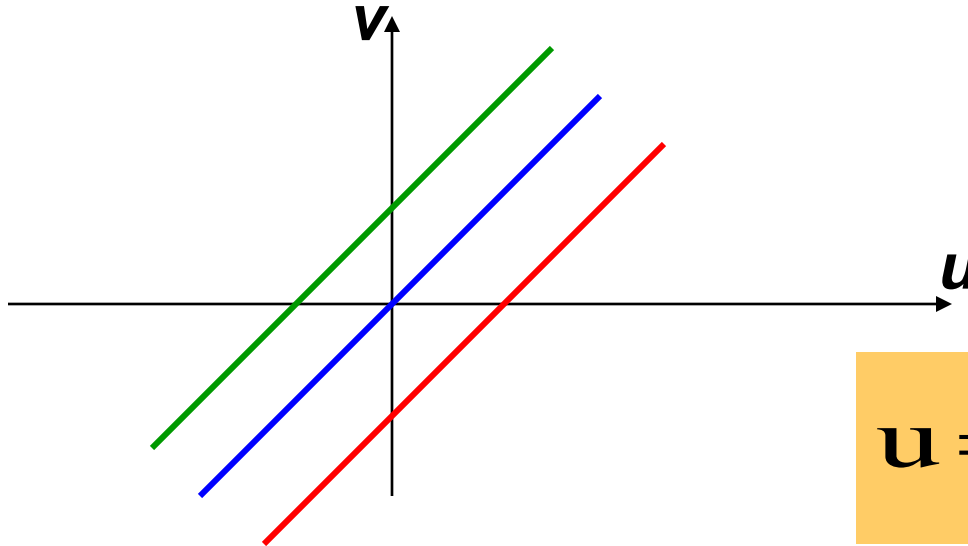


Bias of a Neuron

- Bias b has the effect of applying an affine transformation to u

$$v = u + b$$

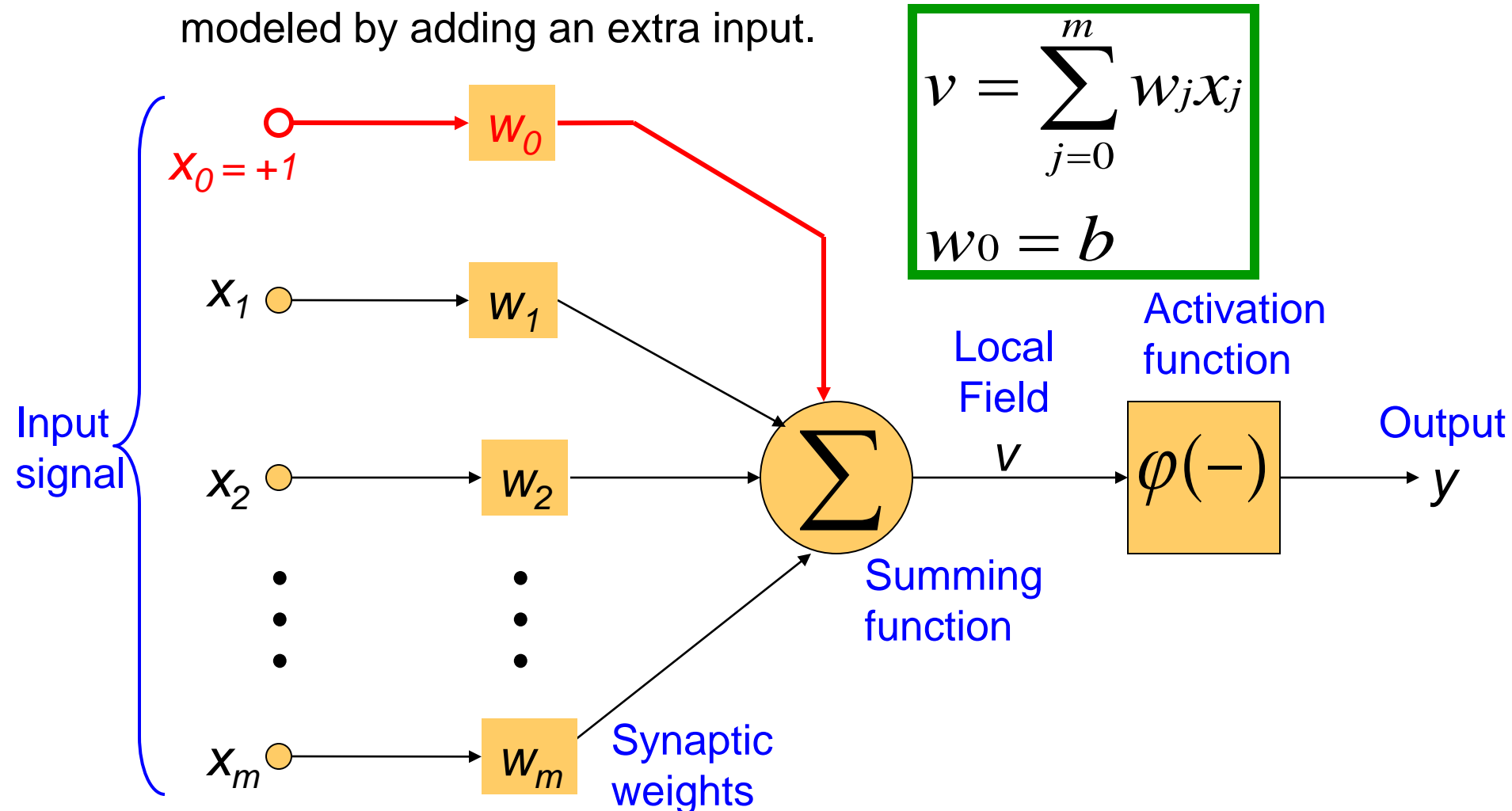
- v is the induced field of the neuron



$$u = \sum_{j=1}^m w_j x_j$$

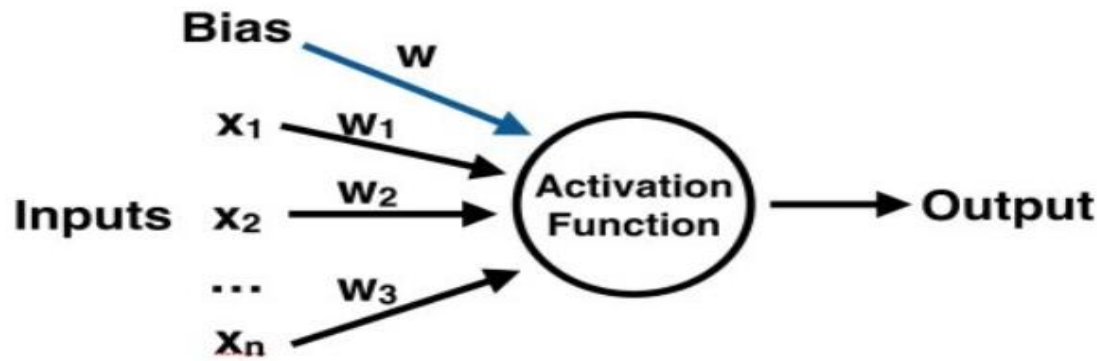
Bias as extra input

- Bias is an external parameter of the neuron. Can be modeled by adding an extra input.

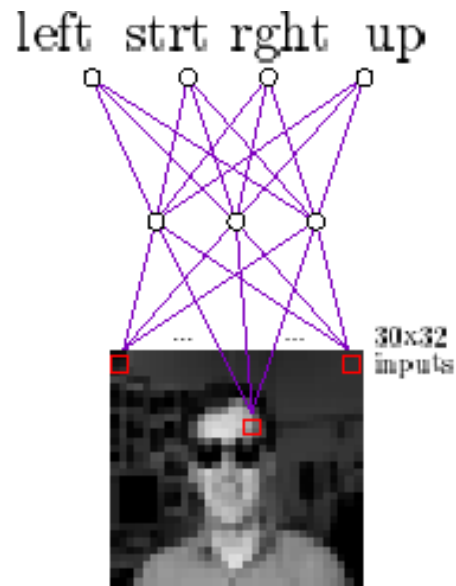


Perceptron

- A perceptron has one or more inputs, a bias, an activation function, and a single output.
- The perceptron receives inputs, multiplies them by some weight, and then passes them into an activation function to produce an output.



Face Recognition



Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Handwritten digit recognition

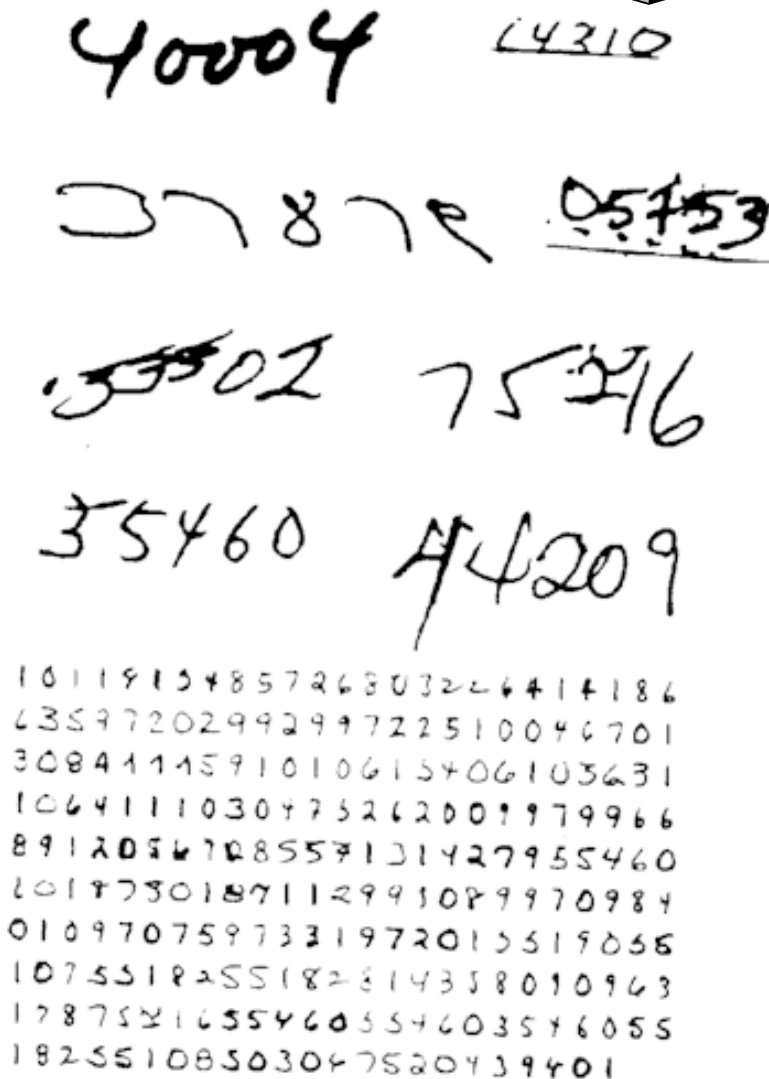


FIGURE 10.8

Examples of ZIP code image, and segmented and normalized numerals from the testing set. (Source: Reprinted with permission from Y. Le Cun, et al., "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, 1:541-551, 1989. ©1989 The MIT Press.)

Advantages



- A neural network can perform tasks that a linear program can not.
- When an element of the neural network fails, it can continue without any problem by their parallel nature

Disadvantages



- Requires high processing time for large neural networks
- The architecture of a neural network is different from the architecture of microprocessors therefore needs to be emulated

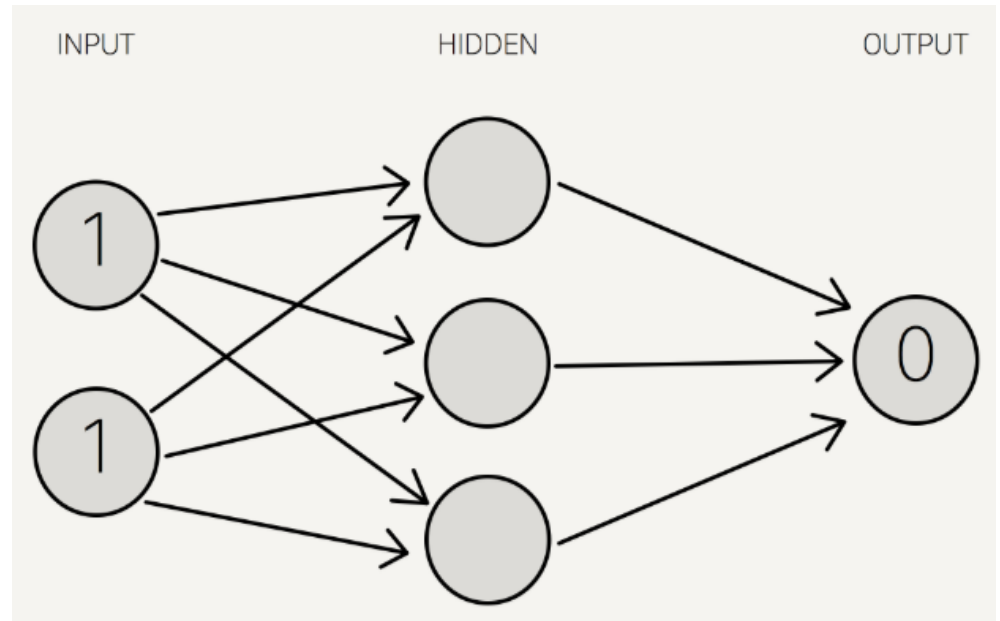
Feed Forward Algorithm

- Initialize the weights and biases randomly.
- Iterate over the data
 - i. Compute the predicted output using the sigmoid function
 - ii. Compute the loss using the square error loss function
 - iii. $W(\text{new}) = W(\text{old}) - \alpha \Delta W$
 - iv. $B(\text{new}) = B(\text{old}) - \alpha \Delta B$
- Repeat until the error is minimal

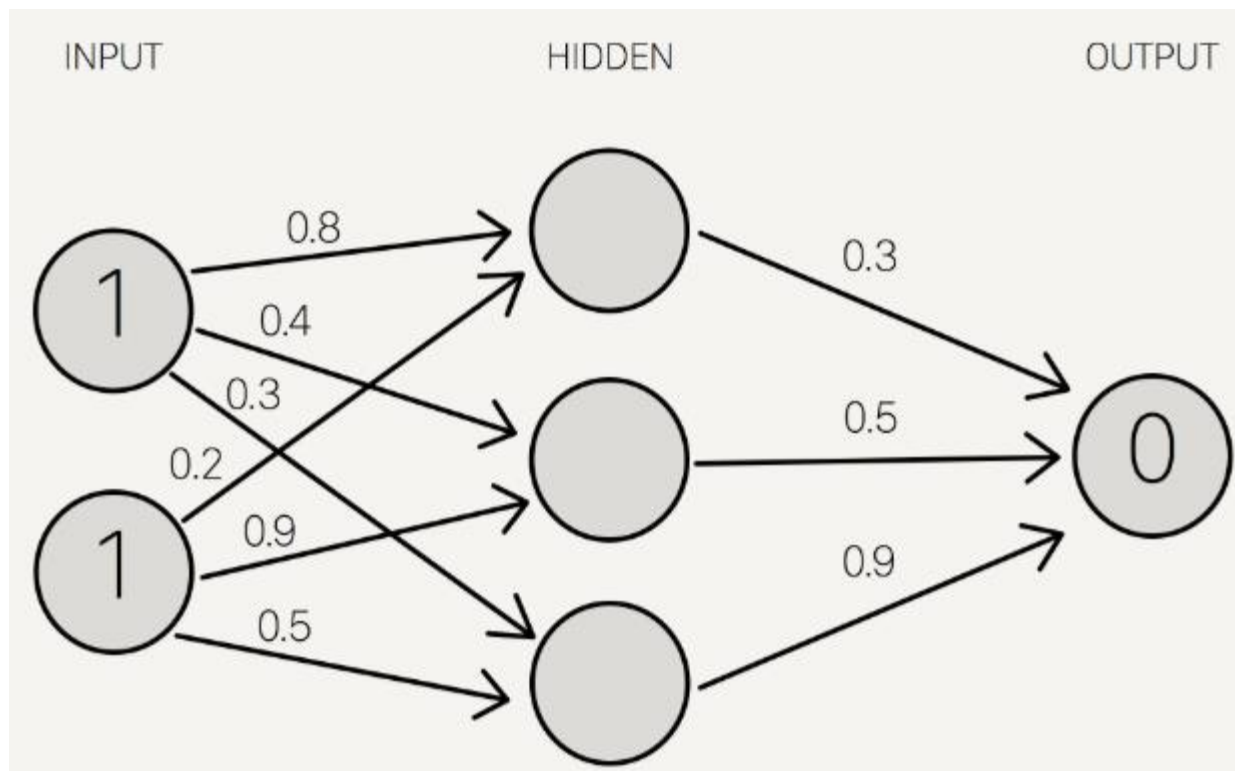
Feed forward - Illustration

$(1, 1) \Rightarrow 0$

input	output
0, 0	0
0, 1	1
1, 0	1
1, 1	0

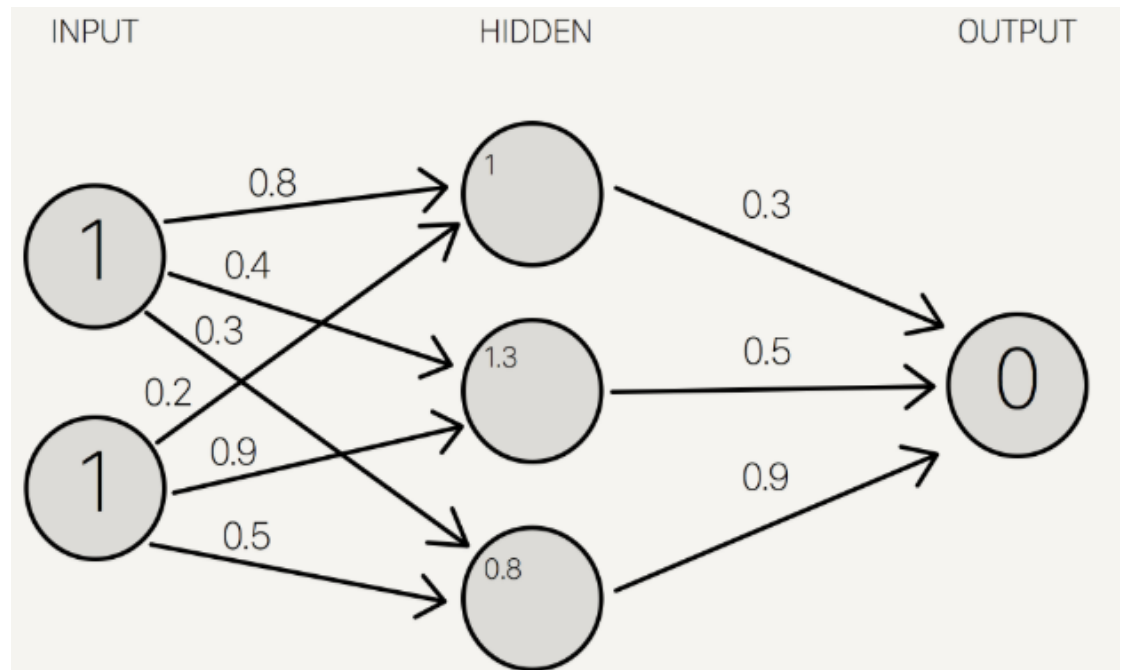


Assign weight to all the synapses






Find Z

$$\begin{aligned}1 * 0.8 + 1 * 0.2 &= 1 \\1 * 0.4 + 1 * 0.9 &= 1.3 \\1 * 0.3 + 1 * 0.5 &= 0.8\end{aligned}$$



Activation Function

	Linear	$g(z) = z$
	Logistic (sigmoid)	$g(z) = 1 / (1 + \exp(-z))$
	Hyperbolic tangent (sigmoid)	$g(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$

ReLU Rectified linear units represented by $R(x)$

$$R(x) = \max(0, x)$$

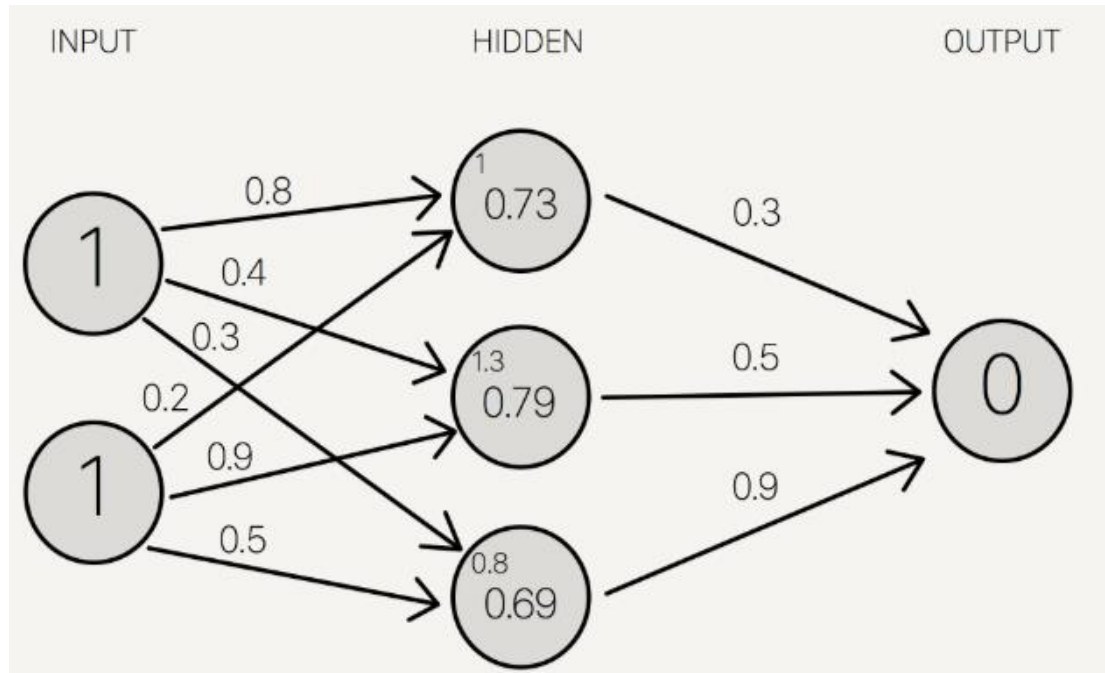
Partial Derivative of ReLU

if $x < 0$, $R(x) = 0$

and if $x \geq 0$, $R(x) = x$.

Apply activation function to z

$S(1.0) = 0.73105857863$
 $S(1.3) = 0.78583498304$
 $S(0.8) = 0.68997448112$



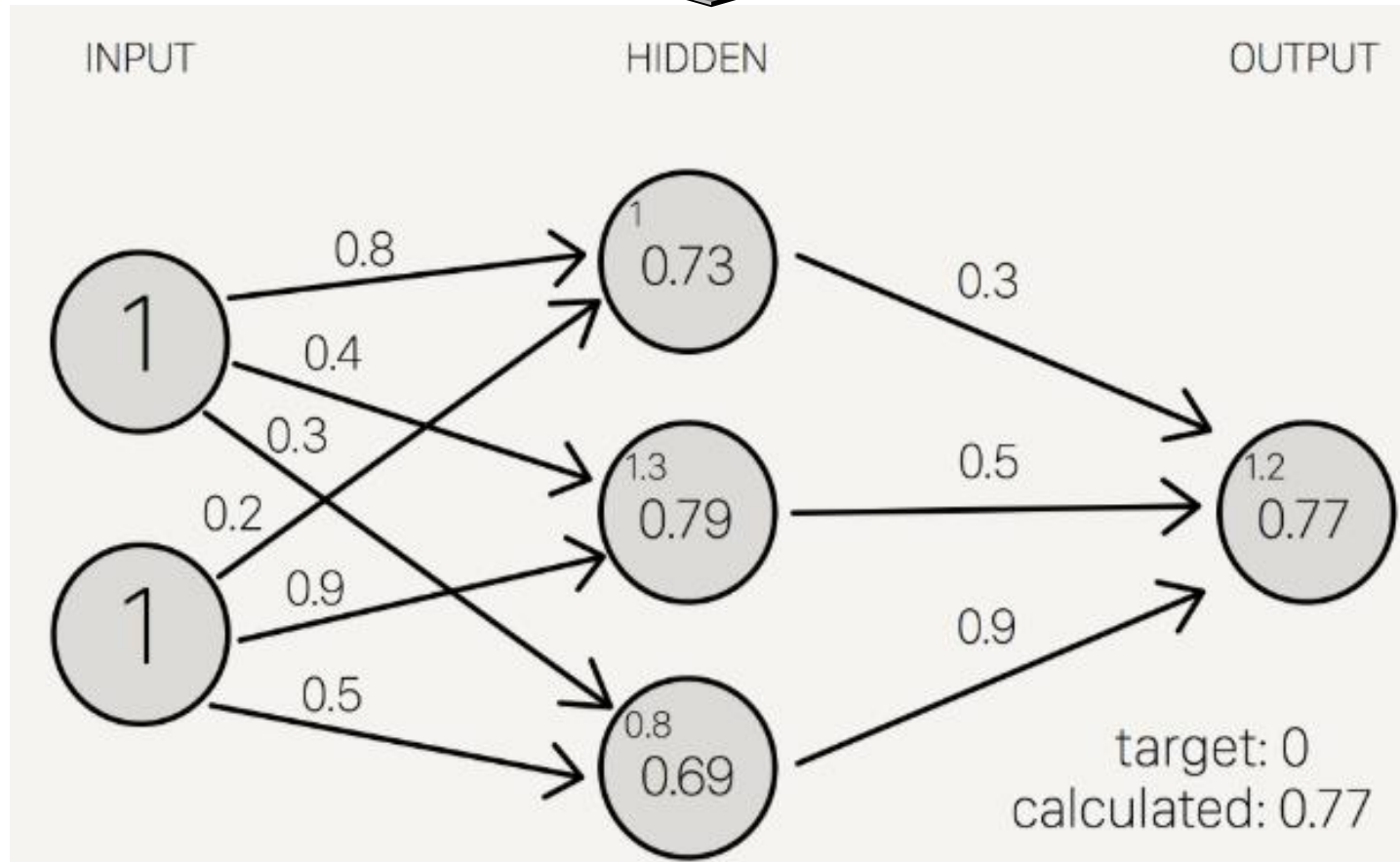
- Sum the product of the hidden layer results with the second set of weights

$$0.73 * 0.3 + 0.79 * 0.5 + 0.69 * 0.9 = 1.235$$

- apply the activation function to get the final output result.

$$S(1.235) = 0.7746924929149283$$

Result of Feedforward



Back Propagation

- To improve our model, we first have to quantify just how wrong our predictions are. Then, we adjust the weights accordingly so that the margin of errors are decreased.
- Output sum margin of error= target-calc

$$0-0.77=-0.77$$

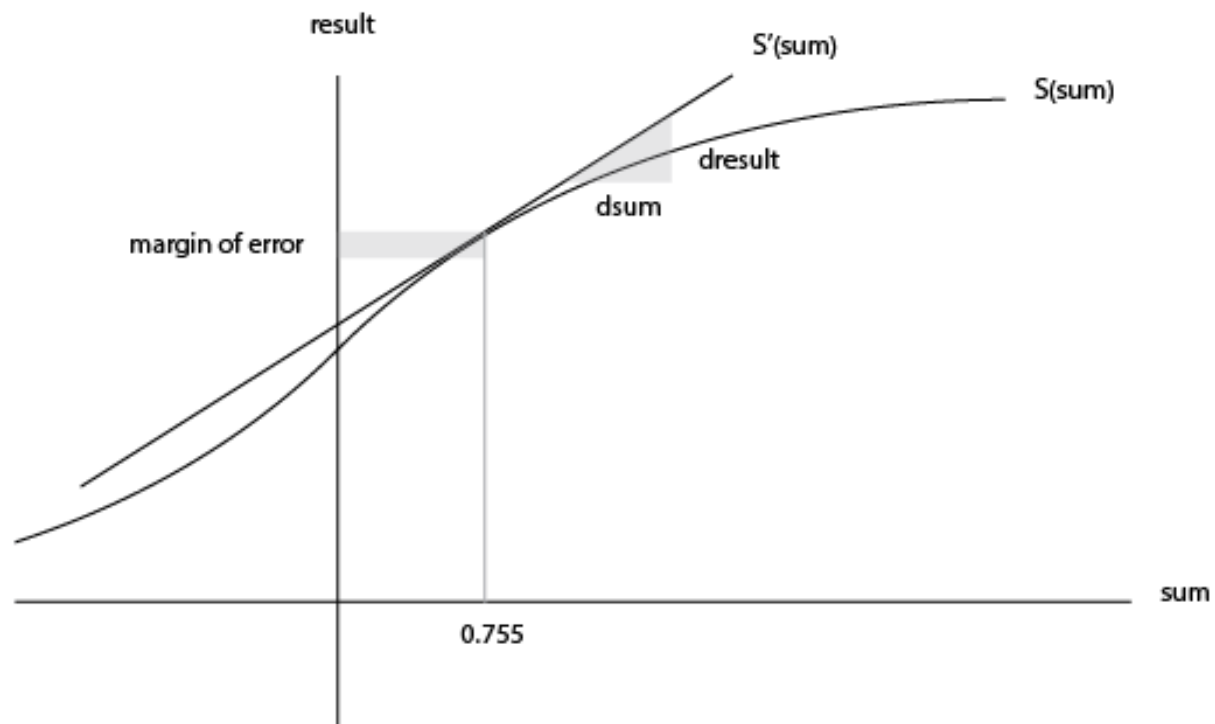
- Take the derivative of the activation function and apply it to the output sum
- The activation function we have taken in the example is sigmoid function
- The derivative of the sigmoid function is ds or $s' = \frac{\exp(-x)}{(1+\exp(-x))^2}$

Find delta output sum

Delta output sum = $S'(\text{sum}) * (\text{output sum margin of error})$

Delta output sum = $S'(1.235) * (-0.77)$

Delta output sum = -0.13439890643886018



- Now we have the proposed change in the output layer is -0.13

$$H_{result} \times w_{h \rightarrow o} = O_{sum}$$

$$\frac{dO_{sum}}{dw_{h \rightarrow o}} = H_{results}$$

$$dw_{h \rightarrow o} = \frac{dO_{sum}}{H_{results}}$$

Find Delta weights using delta output sum

```
hidden result 1 = 0.73105857863  
hidden result 2 = 0.78583498304  
hidden result 3 = 0.68997448112
```

```
Delta weights = delta output sum / hidden layer results  
Delta weights = -0.1344 / [0.73105, 0.78583, 0.69997]  
Delta weights = [-0.1838, -0.1710, -0.1920]
```

```
old w7 = 0.3  
old w8 = 0.5  
old w9 = 0.9
```

```
new w7 = 0.1162  
new w8 = 0.329  
new w9 = 0.708
```

$$H_{result} \times w_{h \rightarrow o} = O_{sum}$$

$$\frac{dH_{result}}{dO_{sum}} = \frac{1}{w_{h \rightarrow o}}$$

$$dH_{result} = \frac{dO_{sum}}{w_{h \rightarrow o}}$$

$$S'(H_{sum}) = \frac{dH_{sum}}{dH_{result}}$$

$$\delta H_{result} \times \frac{dH_{sum}}{dH_{result}} = \frac{dO_{sum}}{w_{h \rightarrow o}} \times \frac{dH_{sum}}{dH_{result}}$$

$$\delta H_{sum} = \frac{dO_{sum}}{w_{h \rightarrow o}} \times S'(H_{sum})$$

```
Delta hidden sum = delta output sum / hidden-to-outer weights * S'(hidden sum)
Delta hidden sum = -0.1344 / [0.3, 0.5, 0.9] * S'([1, 1.3, 0.8])
Delta hidden sum = [-0.448, -0.2688, -0.1493] * [0.1966, 0.1683, 0.2139]
Delta hidden sum = [-0.088, -0.0452, -0.0319]
```

$$I \times w_{i \rightarrow h} = H_{sum}$$

$$\frac{dH_{sum}}{dw_{i \rightarrow h}} = I$$

$$dw_{i \rightarrow h} = \frac{dH_{sum}}{I}$$

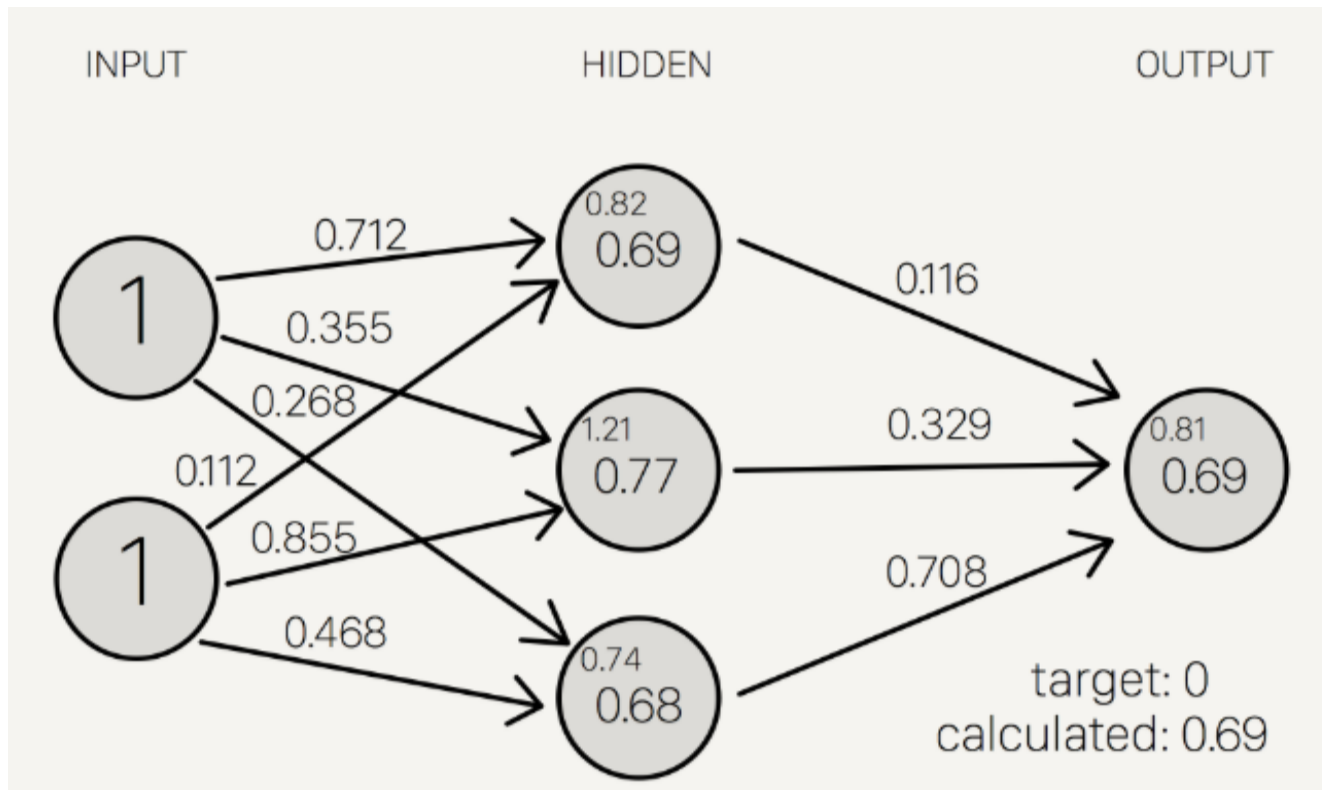
```
input 1 = 1
input 2 = 1

Delta weights = delta hidden sum / input data
Delta weights = [-0.088, -0.0452, -0.0319] / [1, 1]
Delta weights = [-0.088, -0.0452, -0.0319, -0.088, -0.0452, -0.0319]

old w1 = 0.8
old w2 = 0.4
old w3 = 0.3
old w4 = 0.2
old w5 = 0.9
old w6 = 0.5

new w1 = 0.712
new w2 = 0.3548
new w3 = 0.2681
new w4 = 0.112
new w5 = 0.8548
new w6 = 0.4681
```

Do again the feed forward

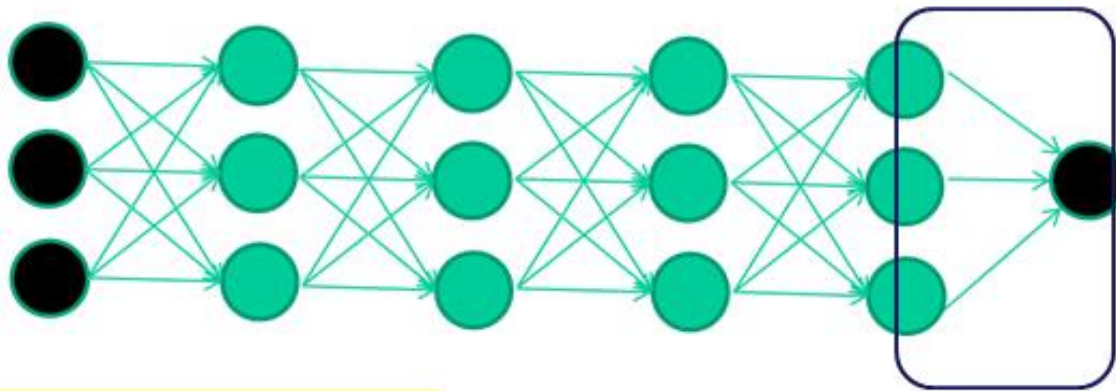


Gradient Descent

- Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost)
- Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.
- Takes longer time

Deep Learning

‘Deep Learning’ means using a neural network with several layers of nodes between input and output



Tuning parameters

- **hidden_layer_sizes** : *tuple, length = n_layers - 2, default (100,)* The *i*th element represents the number of neurons in the *i*th hidden layer.
- **activation** : *{'identity', 'logistic', 'tanh', 'relu'}, default 'relu'*
Activation function for the hidden layer.
 - 'identity', no-op activation, $f(x) = x$
 - 'logistic', the logistic sigmoid function, returns $f(x) = 1 / (1 + \exp(-x))$.
 - 'tanh', the hyperbolic tan function, returns $f(x) = \tanh(x)$.
 - 'relu', the rectified linear unit function, returns $f(x) = \max(0, x)$
- **solver** : *{'lbfgs', 'sgd', 'adam'}, default 'adam'* The solver for weight optimization.
 - 'lbfgs' is an optimizer in the family of quasi-Newton methods.
 - 'sgd' refers to stochastic gradient descent.
 - 'adam' refers to a stochastic gradient-based optimizer

Hint: Larger datasets – use adam, Smaller datasets – lbfgs

- **alpha** : *float, optional, default 0.0001*
 - L2 penalty (regularization term) parameter.
 - **Can be given in GridSearchCV to find best alpha – can be in range of 10 like 10,1,0.1,0.01,0.0001**
- **learning_rate** : *{‘constant’, ‘invscaling’, ‘adaptive’}, default ‘constant’* Learning rate schedule for weight updates.
 - ‘constant’ is a constant learning rate given by ‘learning_rate_init’.
 - ‘invscaling’ gradually decreases the learning rate at each time step ‘t’ using an inverse scaling exponent of ‘power_t’.

$$\text{effective_learning_rate} = \text{learning_rate_init} / \text{pow}(t, \text{power_t})$$
 - ‘adaptive’ keeps the learning rate constant to ‘learning_rate_init’ as long as training loss keeps decreasing.
- **learning_rate_init** : *double, optional, default 0.001*
- **power_t** : *double, optional, default 0.5*
- **max_iter** : *int, optional, default 200*