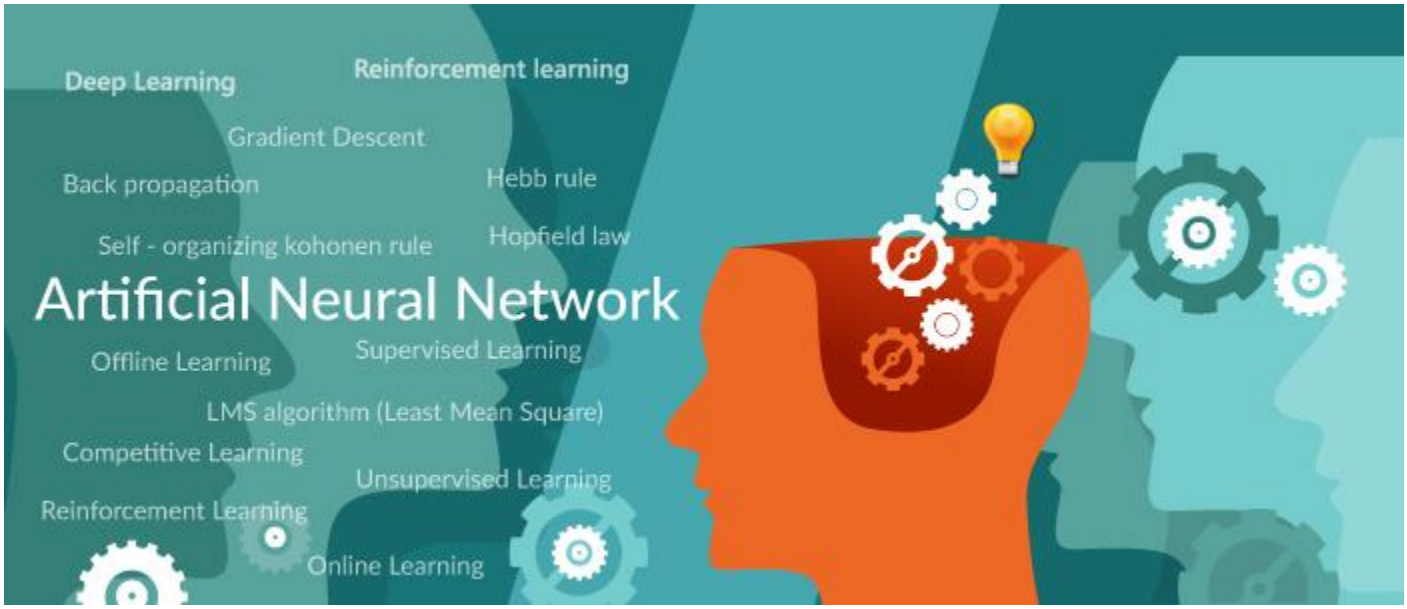
# ARTIFICIAL NEURAL NETWORKS





## Hesham Meneisi 3118

## Helana Ayman 3015

## WHAT ARE ARTIFICIAL NEURAL NETWORKS?

The term 'Neural' is derived from the human nervous system's basic functional unit 'neuron' or nerve cells which are present in the brain and other parts of the human body.

An artificial neural network (ANN) is a computational model based on the structure and functions of biological neural networks. Information that flows through the network affects the structure of the ANN because a neural network changes based on that input and output.

The inventor of the first neurocomputer, Dr. Robert Hecht-Nielsen, defines a neural network as –

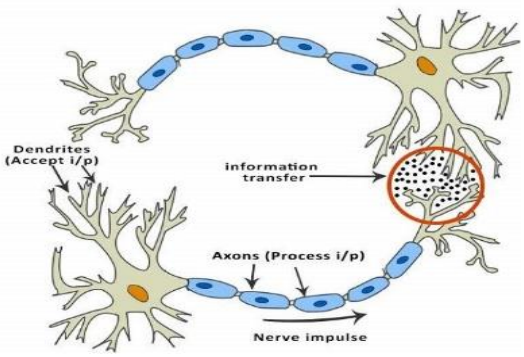
"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

Artificial Neural Networks are the biological brain inspired simulations performed on the computer to perform certain specific tasks like clustering, classification, pattern recognition etc.

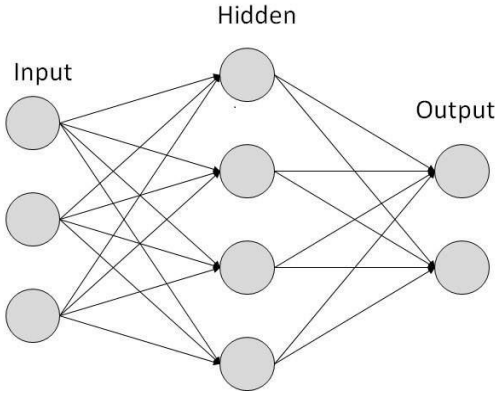
## BASIC STRUCTURE OF ANNS

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites.

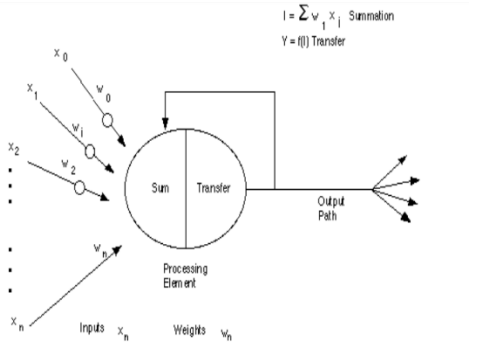
The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.




ANNs are composed of multiple **nodes**, which imitate biological neurons of human brain. The neurons are connected by **links** and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. Each link is associated with **weight**. ANNs are capable of learning, which takes place by altering weight values.



## The artificial neuron :




The artificial neuron simulates four basic functions of a biological neuron.

In fig. , Various inputs to the network are represented by the mathematical symbol, x(n). Each of these inputs is multiplied by a connection weight. The weights are represented by w(n). In the simplest case, these products are summed, fed to a transfer function (activation function) to generate a result, and this result is sent as output. This is also possible with other network structures, which utilize different summing functions as well as different transfer functions.

## Seven major components make up an artificial neuron:

**Component 1. Weighting Factors:** A neuron usually receives many simultaneous inputs. Each input has its own relative weight, which gives the input the impact that it needs on the processing element's summation function. Some inputs are made more important than others to have a greater effect on the processing element as they combine to produce a neural response. They are a measure of an input's connection strength. These strengths can be modified in response to various training sets and according to a network's specific topology or its learning rules.

**Component 2. Summation Function:** The inputs and corresponding weights are vectors which can be represented as (i1, i2 . . . in) and (w1, w2 . . . wn). The total input signal is the dot product of these two vectors. The result; (i1 * w1) + (i2 * w2) +..... + (in * wn) ; is a single number. The summation function can be more complex than just weight sum of products. The input and weighting coefficients can be combined in many different ways before passing on to the transfer function.

**Component 3. Transfer Function:** The result of the summation function is transformed to a working output through an algorithmic process known as the transfer function. In the transfer function the summation can be compared with some threshold to determine the neural output. If the sum is greater than the threshold value, the processing element generates a signal and if it is less than the threshold, no signal (or some inhibitory signal) is generated. Both types of response are significant. The threshold, or transfer function, is generally non-linear. Linear functions are limited because the output is simply proportional to the input.

**Component 4. Scaling and Limiting:** After the transfer function, the result can pass through additional processes, which scale and limit. This scaling simply multiplies a scale factor times the transfer value and then adds an offset. Limiting is the mechanism which insures that the scaled result does not exceed an upper, or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed.

**Component 5. Output Function:** Each processing element is allowed one output signal, which it may give to hundreds of other neurons. Normally, the output is directly equivalent to the transfer function's result. Some network topologies modify the transfer result to incorporate competition among neighboring processing elements.

**Component 6. Error Function and Back-Propagated Value:** In most learning networks the difference between the current output and the desired output is calculated as an error which is then transformed by the error function to match a particular network architecture. Most basic architectures use this error directly but some square the error while retaining its sign, some cube the error, other paradigms modify the error to fit their specific purposes. The error is propagated backwards to a previous layer. This back-propagated value can be either the error, the error scaled in some manner or some other desired output depending on the network type.

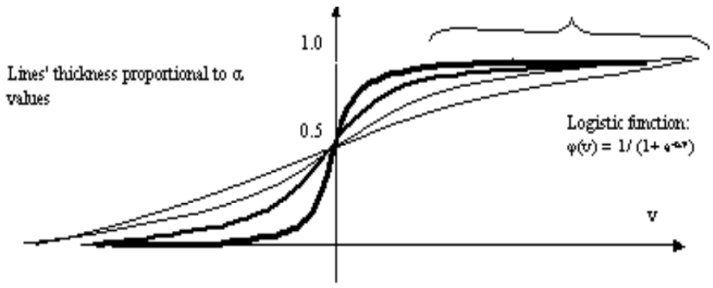
**Component 7. Learning Function:** Its purpose is to modify the weights on the inputs of each processing element according to some neural based algorithm.

## EXAMPLES OF ACTIVATION FUNCTIONS

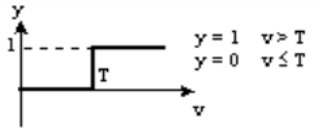
### Transfer (Activation) functions:

The transfer function for neural networks must be differential and therefore continuous to enable correcting error. Derivative of the transfer function is required for computation of local gradient. One such example of a suitable transfer function is the sigmoid function. The sigmoid function is a S-shaped graph. It is one of the most common forms of transfer function used in construction of ANNs. It is defined as a strictly increasing function. Mathematically its derivative is always positive. It exhibits a graceful balance between linear and nonlinear behavior. One example of it is a logistic function represented by the equation: $\varphi(v) = \frac{1}{1+e}$

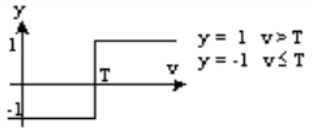
This function has certain characteristic. At extremes of φ(v): φ (v) is flat and φ'(v) is very small. At midrange of φ (v): φ'(v) is maximum .



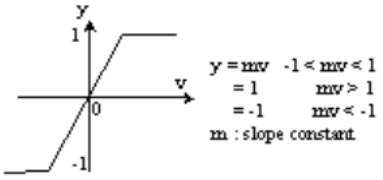

Several other transfer functions can also be employed



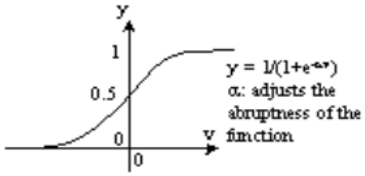
Threshold Function.



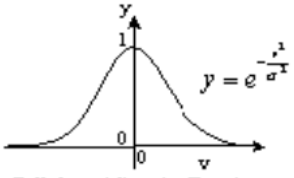
Signum Function



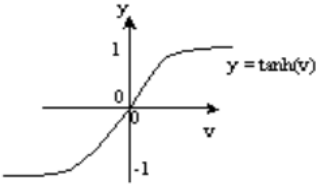
Linear Function with Limiting threshold



Sigmoid Function



Bell shape / Gaussian Function



Hyperbolic Tangent Function.

## TYPES OF ARTIFICIAL NEURAL NETWORKS

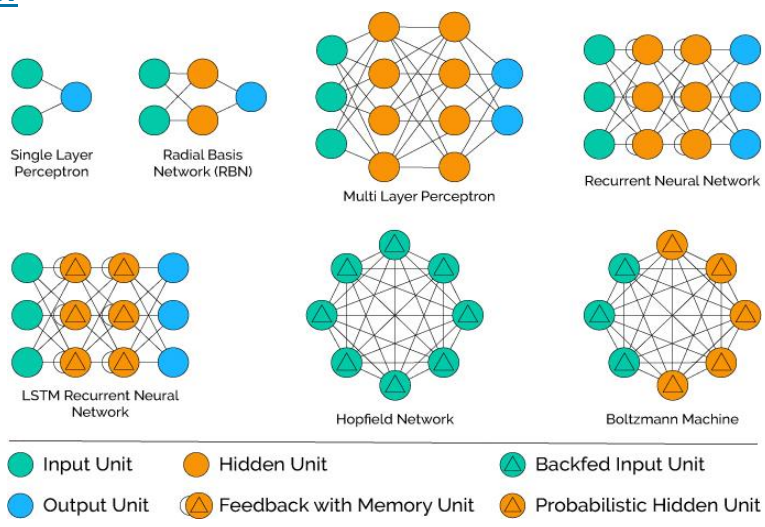
| Parameter | Types | Description |
|---|---|---|
| Based on connection pattern | Feedforward, Recurrent | Feedforward - In which graphs have no loops.<br>Recurrent - Loops occur because of feedback. |
| Based on the number of hidden layer | Single layer, Multi-Layer | Single Layer - Having one hidden layer. E.g. , Single Perceptron<br>Multilayer - Having multiple hidden layers. Multilayer Perceptron |
| Based on nature of weights | Fixed, Adaptive | Fixed - Weights are fixed a priori and not changed at all.<br>Adaptive - Weights are updated and changed during training. |
| Based on Memory unit | Static, Dynamic | Static - Memoryless unit. The current output depends on the current input. E.g. , Feedforward network<br>Dynamic - Memory unit - The output depends upon the current input as well as the current output. E.g. , Recurrent Neural Network |

## Popular Neural Network Architectures:

**Perceptron** - Neural Network is having two input units and one output units with no hidden layers. These are also known as 'single layer perceptrons'.

**Radial Basis Function Network** - These networks are similar to the feed forward Neural Network except radial basis function is used as activation function of these neurons.

**Multilayer Perceptron** - These networks use more than one hidden layer of neurons, unlike single layer perceptron. These are also known as Deep Feedforward Neural Networks.




**Recurrent Neural Network** - Type of Neural Network in which hidden layer neurons has self-connections. It possess memory. At any instance, hidden layer neuron receives activation from the lower layer as well as it previous activation value.

**Long /Short Term Memory Network (LSTM)** - Type of Neural Network in which memory cell is incorporated inside hidden layer neurons.

## BACKPROPAGATION FOR FEED FORWARD NETWORKS

The backpropagation (BP) algorithm is the most commonly used training method for feed forward networks. Consider a multi-layer perceptron with 'k' hidden layers. Together with the layer of input units and the layer of output units this gives k+2 layers of units altogether, which are numbered by 0 , … , k+1. Let the number of input units be K, output units be L and of unis in hidden layer m be $N^m$.

The weight of $j^{th}$ unit in layer m and the $i^{th}$ unit in layer m+1 is denoted by $w_{ij}^m$ . The activation of the $i^{th}$ unit in layer m is $x_i^m$ (for m = 0 this is an input value, for m = k+1 an output value). The training data for a feedforward network training task consist of T input-output (vector-valued) data pairs

$$u(n) = (x_1^0(n), \dots , x_k^0(n))^{\,t}, d(n) = \left(d_1^{k+1}(n), \dots , d_L^{k+1}(n)\right)^t$$

where 'n' denotes training instance. The activation of non-input units is computed according to

$$x_i^{m+1}(n) = f\Big(\sum_{i=1,\dots,N^m} w_{ij}^m \, x_j(n)\Big)$$

Presented with training input u(t), the previous update equation is used to compute activations of units in subsequent hidden layers, until a network response $y(n) = (x_1^{k+1}(n), \dots , x_L^{k+1}(n))^t$ is obtained in the output layer.

The objective of training is to find a set of network weights such that the summed squared error

$E = \sum_{n=1,\dots,T} \|d(n) - y(n)\|^2 = \sum_{n=1,\dots,T} E(n)$ is minimized. This is done by incrementally changing the weights along the direction of the error gradient with respect to weights $\frac{\partial E}{\partial w_{ij}^m} = \sum_{t=1,\dots,T} \frac{\partial E(n)}{\partial w_{ij}^m}$ using a (small) learning rate γ:

new $w_{ij}^m = w_{ij}^m - \gamma \frac{\partial E}{\partial w_{ij}^m}$ .

This is the formula used in batch learning mode, where new weights are computed after presenting all training samples. One such pass through all samples is called an epoch. Before the first epoch, weights are initialized, typically to small random numbers. A variant is incremental learning, where weights are changed after presentation of individual training samples: new $w_{ij}^m = w_{ij}^m - \gamma \frac{\partial E(n)}{\partial w_{ij}^m}$ . The subtask in this method is the computation of the error gradients $\frac{\partial E(n)}{\partial w_{ij}^m}$.

The backpropagation algorithm is a scheme to perform these computations. The procedure for one epoch of batch processing is given below. Input: current weights $w_{ij}^m$ , training samples.

Output: new weights.

**Computation steps:**

1. For each sample n, compute activations of internal and output units (forward pass).

2. Compute, by proceeding backward through m = k+1, k, …, 1 , for each unit $x_i^m$ the error propagation term $\delta_i^m(n)$

$$\delta_i^{k+1}(n) = (d_i(n) - y_i(n)) \frac{\partial f(u)}{\partial u}\Big|_{u=z_i^{k+1}}$$

for the output layer $\delta_i^m(n) = \sum_{i=1}^{N^{m+1}} \delta_i^{m+1} \; w_{ij}^m \; \frac{\partial f(u)}{\partial u}\Big|_{u=z_j^m}$

and for the hidden layers, where $z_i^m(n) = \sum_{j=1}^{N^{m-1}} x_j^{m-1}(n) \; w_{ij}^{m-1}$

is the internal state (or potential) of unit $x_i^m$ . This is the error backpropagation pass.

Mathematically, the error propagation term $\delta_i^m(n)$ represents the error gradient w.r.t. the potential of the unit $x_i^m$ .
$\frac{\partial E}{\partial u}\Big|_{u=z_j^m}$

3. Adjust the connection weights according to

new $w_{ij}^{m-1} = w_{ij}^{m-1} + \gamma \sum_{t=1}^{T} \delta_i^m(n)\, x_j^{m-1}(n)$

After every such epoch, compute the error. Stop when the error falls below a predetermined threshold or when the change in error falls below another predetermined threshold or when the number of epochs exceeds a predetermined maximal number of epochs. Many (order of thousands in nontrivial tasks) such epochs may be required until a sufficiently small error is achieved. One epoch requires O(T M) multiplications and additions, where M is the total number of network connections.

The basic gradient descent approach (and its backpropagation algorithm implementation) is notorious for slow convergence, because the learning rate γ must be typically chosen small to avoid instability. Another approach to achieve faster convergence is to use second-order gradient descent techniques, which exploit curvature of the gradient but have epoch complexity O(T $M^2$ ).

## TRAINING OF ARTIFICIAL NEURAL NETWORKS

Once a network has been structured for a particular application, it is ready for training. At the beginning, the initial weights are chosen randomly and then the training or learning begins. There are two approaches to training; supervised and unsupervised.

### Supervised learning:

The learning algorithm would fall under this category if the desired output for the network is also provided with the input while training the network. By providing the neural network with both an input and output pair it is possible to calculate an error based on it's target output and actual output. It can then use that error to make corrections to the network by updating it's weights.

### Unsupervised Learning:

In this paradigm the neural network is only given a set of inputs and it's the neural network's responsibility to find some kind of pattern within the inputs provided without any external aid. This type of learning paradigm is often used in data mining and is also used by many recommendation algorithms due to their ability to predict a user's preferences based on the preferences of other similar users it has grouped together.

### Reinforcement Learning:

Reinforcement learning is similar to supervised learning in that some feedback is given, however instead of providing a target output a reward is given based on how well the system performed. The aim of reinforcement learning is to maximize the reward the system receives through trial-and-error. This paradigm relates strongly with how learning works in nature, for example an animal might remember the actions it's previously taken which helped it to find food (the reward).

## The Universal Approximation Theorem:

The universal approximation theorem claims that the standard multilayer feed-forward networks with a single hidden layer that contains finite number of hidden neurons, and with arbitrary activation function(φ is an activation function if and only if φ is bounded and $\lim_{x\to+\infty} \phi(x) = a$ , $\lim_{x\to-\infty} \phi(x) = b$ , $a \neq b$ are universal approximators in C($R^m$). The multilayer feedforward architecture gives neural networks the potential of being universal approximators. The output units are always assumed to be linear. For notational convenience we shall explicitly formulate our results only for the case where there is only one output unit. (The general case can easily be deduced from the simple case.) The theorem in mathematical terms:

***Theorem:***

Let φ( ) be an arbitrary activation function. Let X ⊆ $R^m$. and X is compact. The space of continuous functions on X is denoted by C(X). Then $\forall f \in C(X), \forall \varepsilon > 0$: $\exists n \in N, a_{ij}, b_i, w_i \in R, i \in \{1 \ldots n\}, j \in \{1 \ldots m\}$ :

$$(A_n f)(x_1, .., x_m) = \sum_{i=1}^{n} w_i \ \phi\left(\sum_{j=1}^{m} a_{ij} x_j + b_i\right)$$

as an approximation of the function $f( )$; that is $\|f - A_n f\| < \varepsilon$

(In the notation $A_n f$, n shows the number of hidden neurons) How to measure the accuracy of approximation depends on how to measure closeness between functions, which in turn varies significantly with the specific problem to be dealt with. In many application, it is necessary to have the network simultaneously well on all input samples. In this case, closeness measured by the uniform distance between functions, that is:

$$\|f - A_n f\|_\infty < sup_{\underline{x}\in X} \left|f(\underline{x}) - (A_n f)(\underline{x})\right|$$

In other applications, we think of inputs as random variables and are interested in the average performance:

$$\|f - A_n f\|_p = \sqrt[p]{\int_X |f(\underline{x}) - (A\_n f)(\underline{x})|^p \, d\mu(\underline{x})}$$

1≤ p < ∞ , the most popular choice being p = 2, corresponding to the mean square error. Of course, there are many more ways of measuring closeness of functions. In particular, in many applications, it is also necessary that the derivates of the approximating function implemented by the network closely resemble those of the function to be approximated, up to some order.

## Classification Using ANN :

The ANN is an important and successful application because of the characteristics of its information-processing mechanism , and it has been successfully applied to broad spectrum of data-intensive applications. ANNs have gained popularity in various fields due to its ability to solve highly complex problems such as; classification, detection, and prediction with good accuracy . NNs have found a profound success in the area of classification. By repeatedly showing a neural network inputs classification into groups, the network can be trained to discern the criteria used to classify, and it can do so in a generalized manner allowing successful classification of new inputs not having been used during training Over the years, significant researches have been performed for different applications of classifications. One of these researches , proposed the classification system for fingerprint images using NN with good performance and high accuracy. In 2010, they tried to classify the bacteria. The results proved that ANN is feasible and efficient. Han, Cheng & Meng (2002), proposed the Classification of Aerial Photograph Using Neural Network. The results demonstrated that the neural network suits the classification of remotely sensed data and is superior to the maximum likelihood classifier in the accuracy of the classification and has an overall effect. Spoken letters has been classified by using neural network. The results indicate a classification accuracy of 100% (training) and 93% (testing). The capability of neural network is proved in the field of classification and its high performance.

## GENERAL APPLICATIONS:

Many of the networks being designed presently are statistically quite accurate (upto 85% to 90% accuracy). Currently, neural networks are not the user interface, which translates spoken words into instructions for a machine but some day it will be achieved. VCRs, home security systems, CD players, and word processors will simply be activated by voice. Touch screen and voice editing will replace the word processors of today while bringing spreadsheets and data bases to a level of usability. Neural network design is progressing in other more promising application areas.

**-Language Processing:** These applications include text-to-speech conversion, auditory input for machines, automatic language translation, secure voice keyed locks, automatic transcription, aids for the deaf, aids for the physically disabled which respond to voice commands and natural language processing.

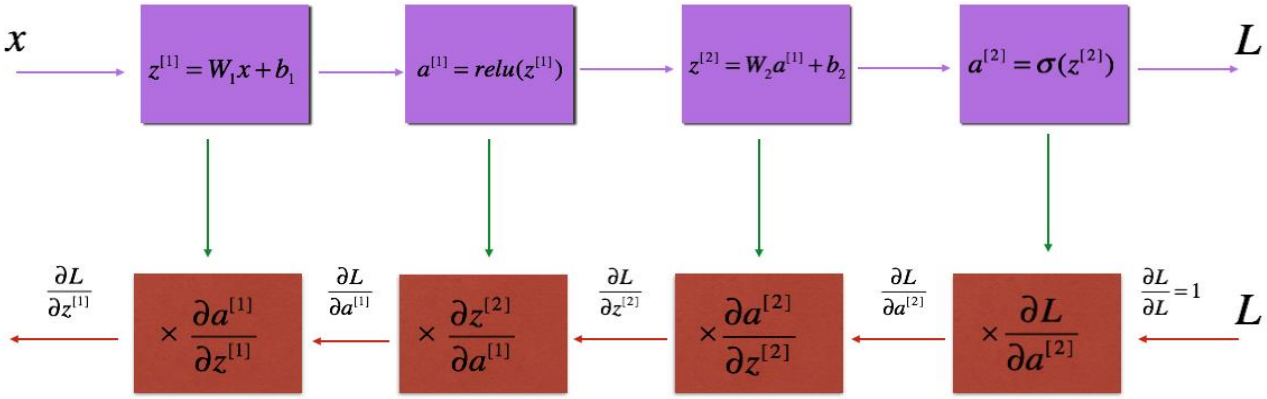
**-Character Recognition:** Neural network based products are available which can recognize hand printed characters through a scanner. It is 98% accurate for numbers, a little less for alphabetical characters. Quantum Neural Network software package is available for recognizing characters, including cursive.

**-Image (data) Compression:** Neural networks can do real-time compression and decompression of data. These networks can reduce eight bits of data to three and then reverse that process upon restructuring to eight bits again. 94 (iv) Pattern Recognition: Many pattern recognition applications are in use like, a system that can detect bombs in luggage at airports by identifying from small variances and patterns from within specialized sensor's outputs, a back-propagation neural network which can discriminate between a true and a false heart attack, a network which can scan and also read the PAP smears etc. Many automated quality control applications are now in use, which are based on pattern recognition. (v) Signal Processing: Neural networks have proven capable of filtering out electronic noise. Another application is a system that can detect engine misfire simply from the engine sound.

## CODING TASK 1

### ANN Implementation

We used the following model:




$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

$$A^{[l]} = g(Z^{[l]}) = g(W^{[l]} A^{[l-1]} + b^{[l]})$$

$$dW^{[l]} = \frac{\partial \mathscr{L}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T}$$

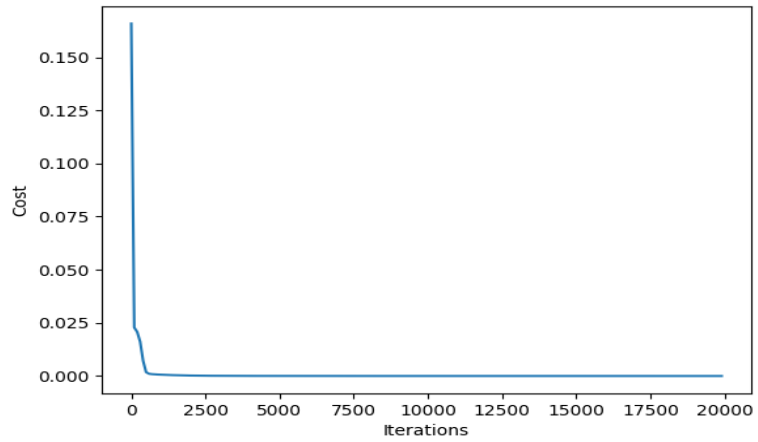
$$db^{[l]} = \frac{\partial \mathscr{L}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)}$$

$$dA^{[l-1]} = \frac{\partial \mathscr{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}$$

This code structure was part of the Deep Learning Engineering specialization course administered by deeplearning.ai instructors on coursera.com

The comments at the beginning of each function describe what it should be implemented to do. The code was delivered to an online judge and passed all tests but has been modified to support custom cost and activation.

We trained a model to try fitting the given function. The model consists of three layers, activated using relu, relu then a linear output. The cost function is MSE




```
X = np.random.rand(2, ex_count)

Y = np.exp(-(np.sum(X**2, axis=0, keepdims=True)))
```

```
    Cost after iteration 19900: 0.000008
    Mean Error =  0.00301537839401
```

```
activation = ['relu', 'relu', 'lin']

parameters, cost_log = L_layer_model(X, Y, [2, 50, 10, 1], activation, 'mse', 0.15, 20000, report_interval)
```

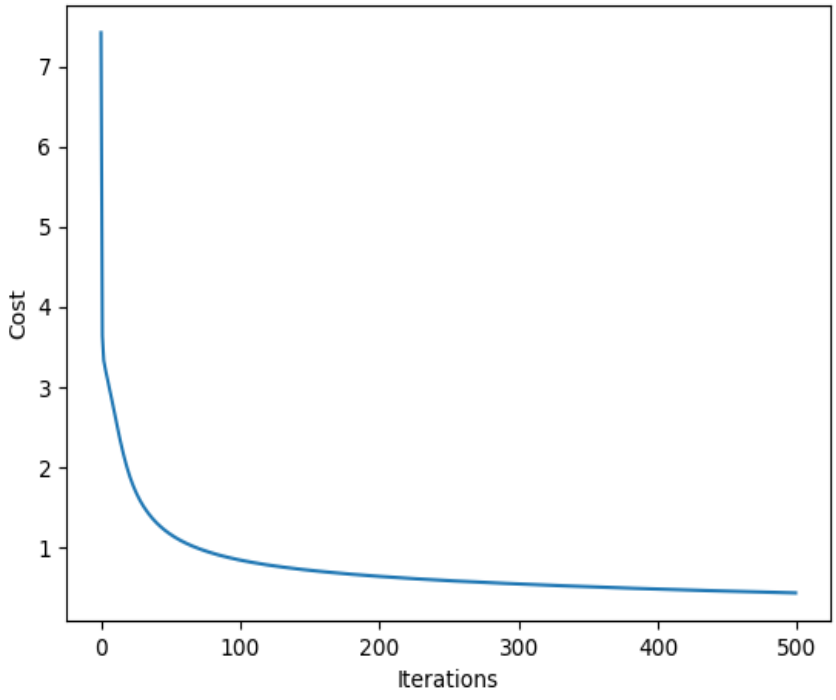
## CODING TASK 2

We used another model to classify the famous MNIST dataset.

The model consists of 2 layers activated using relu then a sigmoid function. The cost function for this model is the log function.

```
activation = ['relu', 'sigmoid']

parameters, cost_log = L_layer_model(X_train, Y_train, [n_features, 30, 10], activation, 'log', 0.22, 500, report_interval)
```




```
Cost after iteration 499: 0.437707
(10000,)
5.89 %
```