2024 Google ML Training Programme (Phase III)

Session 1.3

Introduction to Neural Network and Learning

June 24, 2024

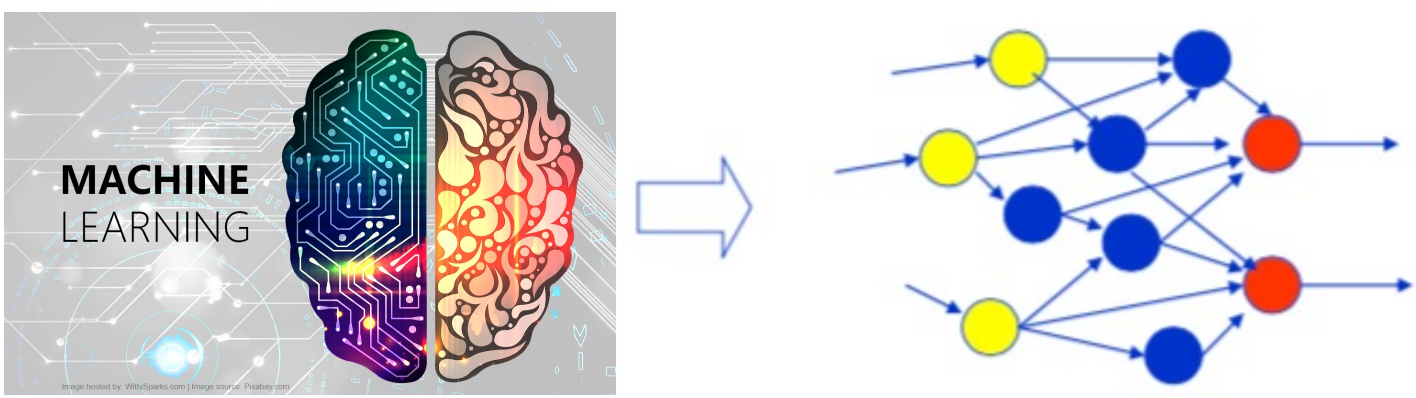
Godfrey A. Mills

Email: gmills@ug.edu.gh

Phone: 020-549-6944

What is a Neural Network

- Neuron is a brain cell in the biological neural network
- Artificial Neuron is a simple processing unit that is considered an approximation of a biological neuron >> may be a physical device or a mathematical construction
- Neural Network is a coordinative system with neurons as the basic elements *connected together in a specific hierarchical structure* (or model or architecture) to perform a *particular task*



What is a Neural Network

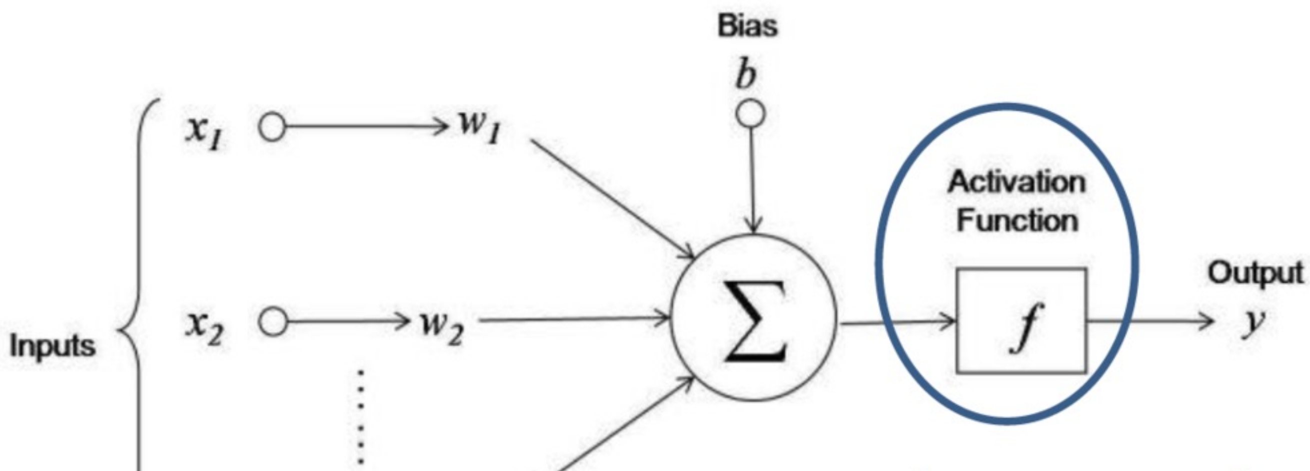
- **Cont'd…..**
- Artificial Neural Network is a *massively parallel distributed processor* system that consists of many *simple processing elements* (neurons) with natural ability to store knowledge from experience and make the knowledge available for use.
- ANNs are *nonlinear statistical data modeling* or *decision tools* used to model *complex relationships between inputs and outputs* or used to find inherent patterns contained in data
- ANN architecture is a layered architecture (single or multiple layers) of neurons that are interconnected in feedforward or feedback loop system >> this gives rise to different models.

Properties of Neural Networks

- Unique properties and capabilities of ML networks that make them suitable as information processing tools include:
  - Consist of an assembly of simple elements or processors;
  - Have massive and parallel connectivity;
  - Information is stored in their connections (no memory);
  - Nonlinear ability (activation) of neurons and the network
  - Ability to learn from examples in real time and generalize;
  - Network ability to adapt weights to changes in environment
  - Highly robust, reliable, and fault tolerant and disturbances;
  - Knowledge is represented by same structure and activation
  - Individual dynamics differ from collective behavior;

Engineering Model of Biological Neuron

- A single neuron is characterized by ***n*** inputs, synaptic weights w_{kj} connecting from input (or neuron) k to neuron j, external bias, linear combiner, an activation function, and one output



$$y = f\left(\mathbf{w}^T\mathbf{x} + b\right)$$

Engineering Model of Biological Neuron

- **Cont'd……**
- The activation function $f(x)$ of a neuron defines output of the neuron or the output of the linear combiner system.
- The activation function $f(x)$ limits the strength of the output of the neuron to ***some finite value*** >> different functions
- The bias to the neuron has the effect of lowering or increasing the ***net*** input to the function $f(x)$ based on whether it is +/-
- Final output is the response of each neuron in the network as:

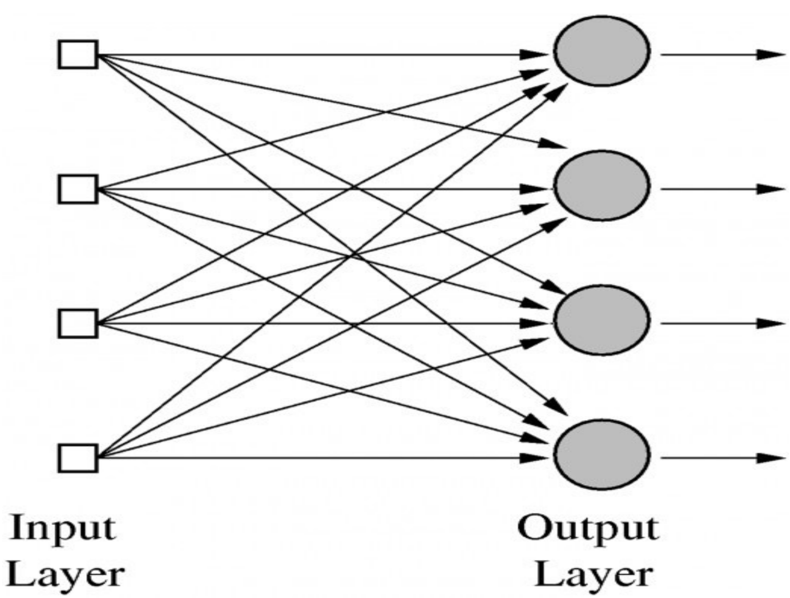
$$y_k = f\left(w_k + u_k = \sum_{i=1}^{m} w_{ki} x_i \right)$$

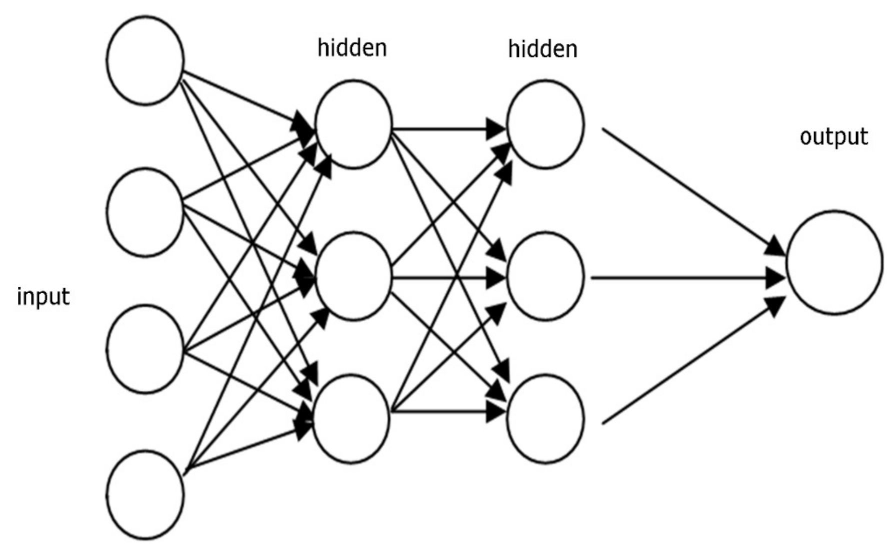
Engineering Model of Biological Neuron

- **Cont'd……**
- The nature of the output of a neuron is based on the activation function $f(x)$ that is used by the neuron for its operation
- **Discrete or Binary Output**
  - Output of $f(x)$ = ***sign***(.) >> neuron produces output of "1" if and only if the ***net*** value input ≥ a certain a threshold θ:
    - >> $f(x) = 1$ if $x \geq \theta$ and $f(x) = -1$ if $x < \theta$
  - The neuron gets triggered only when the weighted inputs reaches that defined threshold value >> similar to a diode
- **Continuous Output**
  - Output of $f(x)$ = ***S*** >> output conforms to the shape $f(x)$

Engineering model of biological neuron

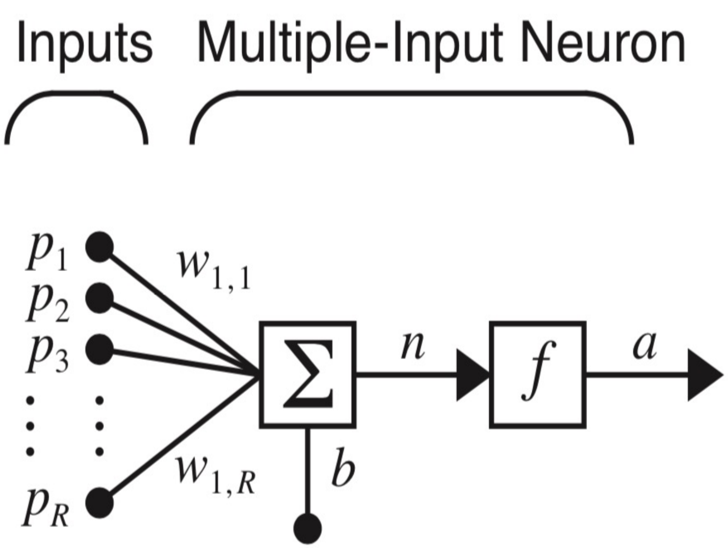
- **Cont’d……**
- Perceptron networks are of two major types:
  - ***single layer perceptron*** and ***multilayer perceptron***
  - multi-input single neuron network, multi-input multi-neuron single layer network, multi-input multi-layer,

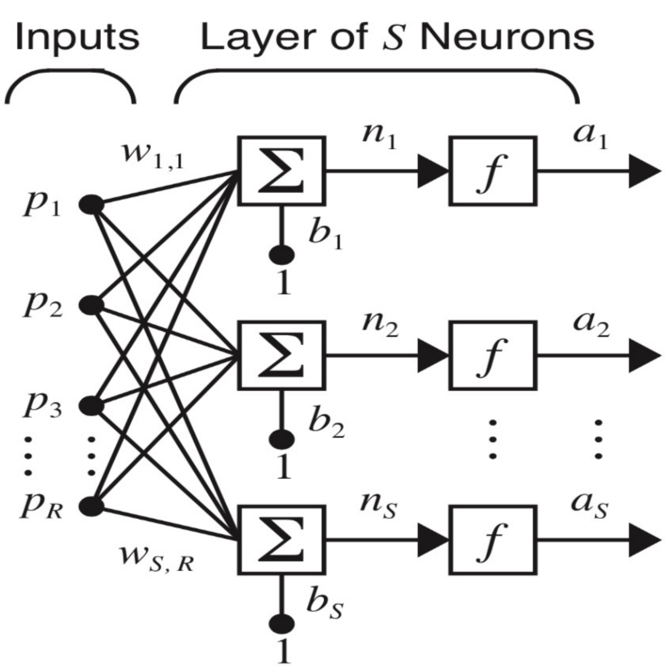




Single-layer Network Model

- Suppose we have a set of inputs ***p*** to a single layer network with multiple neurons in the layer (single neuron network is called a perceptron) and each neuron has a bias ***b*** and weights ***w***, then the output of each neuron will be :
  - >>$\mathbf{z_j = \sum_k [w_{kj} p_k + b_j]}$ and $a_j = f(\mathbf{z_j})$





Single-layer Network Model

- **Cont'd…..**
- Example, for 3-input, 2-neuron single layer network, we have:
  - >> $a_1 = f(\sum_k[\mathbf{w_{11}x_1} + \mathbf{w_{21}x_2} + \mathbf{w_{31}x_3} + \mathbf{b_1}])$
  - >> $a_2 = f(\sum_k[\mathbf{w_{12}x_1} + \mathbf{w_{22}x_2} + \mathbf{w_{31}x_3} + \mathbf{b_2}])$
- The network parameters for computations can be defined for case of single layer multiple neurons and inputs in the form as

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

Multi-layer Network Model

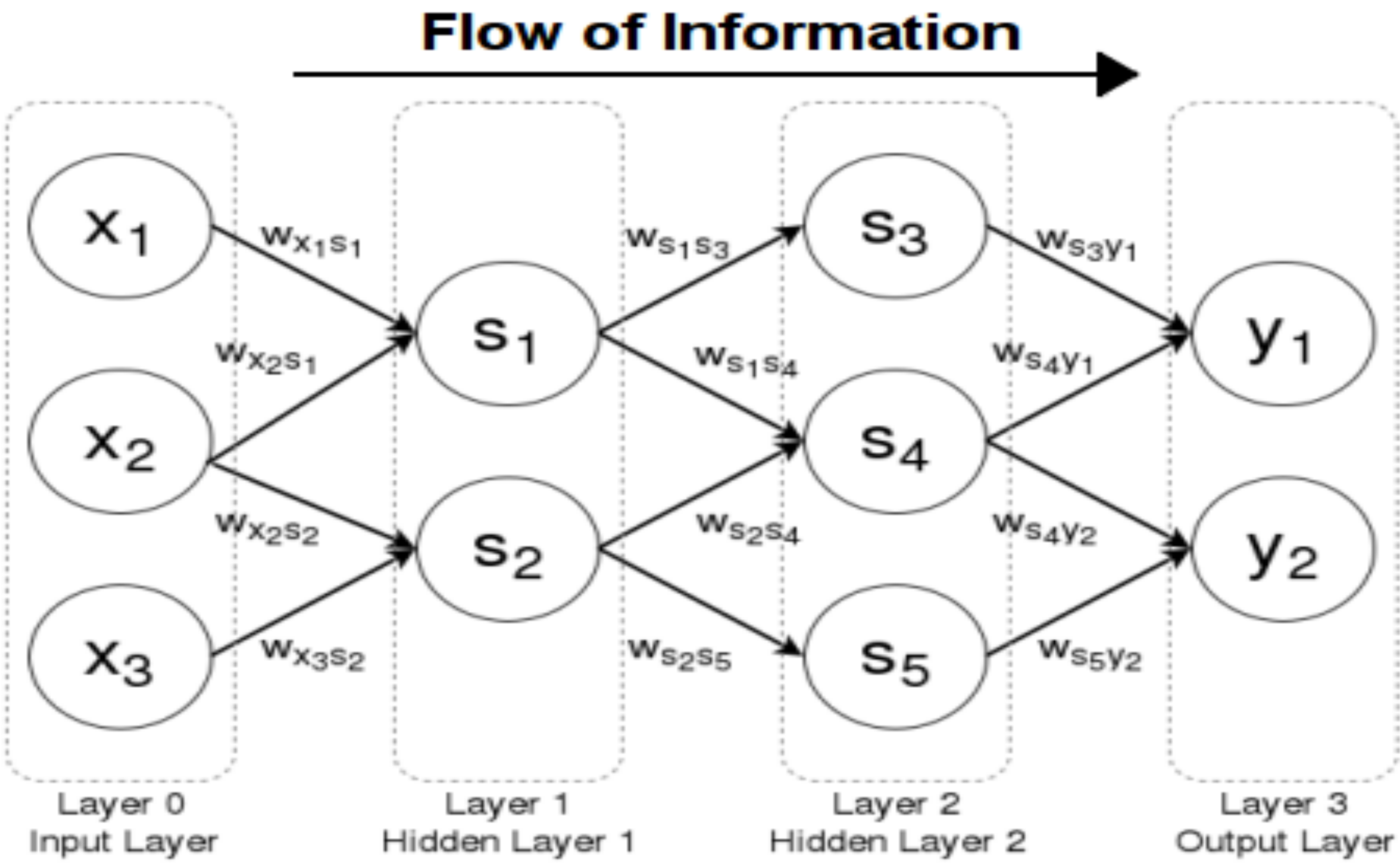
- Multilayer network model is a generalized ***m***-layer feed-forward perceptron network (multilayer perceptron MLP)
- This network is characterized by multiple layers with neurons often numbered by the layers instead of the global numbering
- Most common multilayer network consists of **three** layers:
  - Layer of "**input**" neurons connected to a layer of "**hidden**" neurons which is connected to a layer of "**output**" neurons
  - Input layer neurons do not perform any processing but only receive the input data
  - The first processing neurons are the first hidden layer neurons which which performs operations on the data

Multi-layer Network Model

- **Cont'd…..**
- Simplest feed forward network has zero or multiple hidden layers and varied neurons between the input and output layer >> number of layers depends on the complexity of the task
  - ***Shallow network*** >> neural network limited to only one hidden layer with neurons and an output layer with one or more neurons depending on the classification task at hand
  - ***Deep network*** >> neural network with multiple hidden layers more than one with neurons in layers and an output layer with one or more neurons depending on task at hand

Multi-layer Network Model

- Cont'd……



Neuron Activation or Transfer Functions

- Activation function is the mechanism that a neuron processes incoming information and passes it throughout the network.
- Each neuron in the layers compute linear function (linear combination) of input signals applied to the neuron via the connection weights as (compare with the regression) as :
  - $\mathbf{z} = \sum_{\mathbf{k=1}} \mathbf{w_k x_k} + \mathbf{x_0 w_0}$ where $\mathbf{x_0}$ is the neuron bias.
- Variety of transformational functions used in neural networks:
  - **Threshold function:**
    - $y = f(\mathbf{z}) = \mathbf{1}$, $z \geq 0$ or $y = \mathbf{-1}$, $z < 0$
  - **Unit step function:**
    - $y = f(\mathbf{z}) = \mathbf{1}$, $z \geq 0$ or $y = \mathbf{0}$, $z < 0$

Neuron Activation or Transfer Function

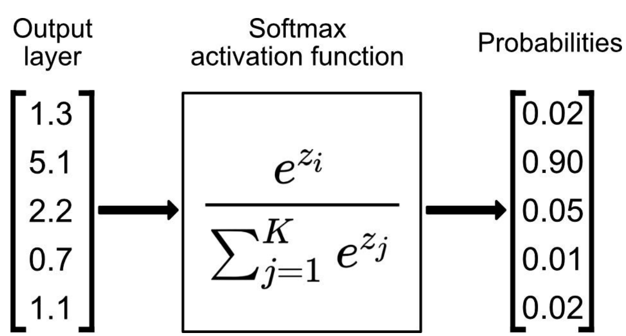
- **Cont'd…..**
  - **Pure linear function:**
    - $y = f(z) = \mathbf{z}$, $z \geq 0$ or $y = \mathbf{-z}$, $z < 0$
  - **Rectified linear unit (ReLu) function:**
    - $y = f(z) = \mathbf{z}$, $z_{min} \leq z \leq z_{max}$;
    - $y = \mathbf{0}$, $z < z_{min}$ and $y = \mathbf{0}$, $z > z_{max}$
  - **Logistic sigmoid function:**
    - $y = f(z) = 1/(1 + e^{-z})$
  - **Hyperbolic tanh function:**
    - $y = f(z) = (e^{z} - e^{-z})/(e^{z} + e^{-z})$

Neuron Activation or Transfer Function

- **Cont'd…..**
  - **Gaussian function:**
    - $y = f(z) = g^* \exp[-(z - z^*)^2 / 2\sigma^2], z^* = \text{mean}(z)$
  - **Softmax function:**
    - $y = f(z) = e^z / \sum e^z$
- Softmax function shares similar operations on the neuron ***net*** signal like the sigmoid function but with more capability.
- Sigmoid function takes single ***net*** input variable z and assigns a number (probability) from 0 to 1 while softmax function can take multiple input ***net*** variable z_j and assign probability for each ***net*** input z_j (j=1, 2,…N).

Neuron Activation or Transfer Function

- **Cont'd…..**
- Softmax function therefore performs role of multiple sigmoid functions >> softmax function is thus used in output layer for models that require multiclass classification tasks.
- Suppose the neurons in the output layer of a network has ***net*** values and softmax function is applied on the variable inputs, the layer will produce output vector whose sum of probability for the classes is 100% >> highest is selected for input class.



Designing Neural Network Model

- Designing an MLP network for any given application involves:
  - input data and transformation to format;
  - number of input layer neurons;
  - number of output layer neurons;
  - network architecture and size or number of layers;
  - activation function for each layer;
  - network learning rule and parameters
  - learning rate;
  - learning algorithm;
  - data learning or training and testing;
  - network performance evaluation;

Learning in Neural Networks

- Learning (weight adjustment) in networks can be viewed as searching through given weight space in a systematic way to determine a weight vector that leads to an optimum (maximum or minimum) value of an objective function.
- Search depends on criterion used for learning >> several criteria used in network learning include:
  - Minimization of mean squared error, relative entropy, maximum likelihood, *gradient descent*.
  - We also have several laws learning laws and new ones are being proposed to suit given application and architecture
- Learning in networks can be ***supervised*** or ***unsupervised***.

Learning in Neural Networks

- **Cont'd….**
- We have two modes of training in neural networks :
  - ***sequential*** learning and ***batch*** learning
- In sequential mode of training, the network parameters are adjusted after each training pattern or sample.
- In batch mode of training, the network parameters are adjusted at end of each epoch (all training patterns are passed)
- While batch mode of training is slower compared with the sequential, it ends up providing slightly better results.
- Limitation of the sequential mode of training is the problem of sensitivity to the ordering in training patterns.

Learning Rules in Neural Networks

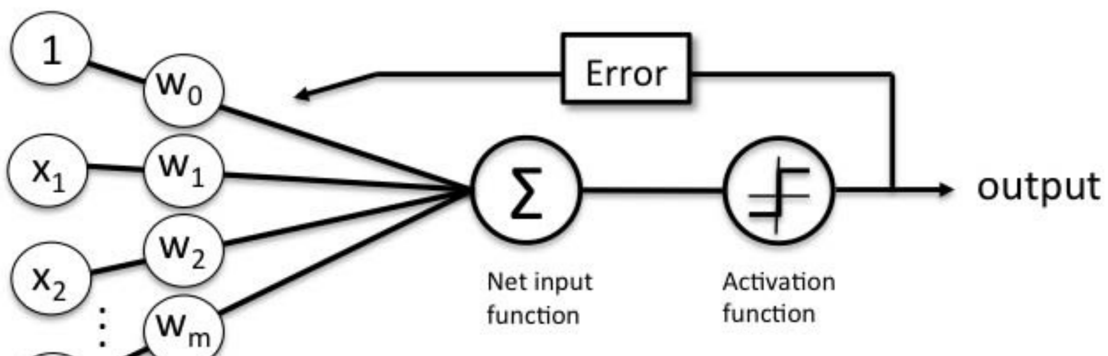
- Learning laws in neural networks basically refers to specific manner the learning equations are implemented.
- General learning law for weights adjustments in network is :
  - >> $\mathbf{W_{k+1} = W_k + \Delta W_k}$ where
  - ΔW is the change in weight, **W** is the weight vector, **E** is the learning function (error function), $\boldsymbol{\eta}$ is a learning constant called learning rate, and X is the input vector >> $\boldsymbol{\eta} = [0,1]$
- Similarly, bias adjustment for the neurons is defined as:
  - >> $\mathbf{b_{k+1} = b_k + \Delta b_k}$ where
  - Δb_k the change in bias and the bias input $\mathbf{X_0 = +1, -1}$

Learning Rules in Neural Networks

- **Cont'd….**
- **Perceptron learning law** : >> discrete learning rule
  - Learning law is only applicable to bipolar output function or neurons that perform binary activation >> learning law is also called ***discrete learning rule***.
  - Change in weight vector is based on desired target output for each input >> law represents supervised learning
  - Weights are adjusted according to the relationship :
    - $\Delta W_k = \eta[S - f(u)]X$

Learning Rules in Neural Networks

- **Cont'd….**
- Algorithm for perceptron learning rule can be described as:
  - Given a set of input vectors, the neuron predicts an output
  - The predicted output is compared with the known output
  - If the output does not match, error is propagated backward to allow weight adjustment to occur in the network.



Perceptron rule.

Learning Rules in Neural Networks

- **Cont'd….**
  - **Change in network weight** >> $\Delta W_k = \eta[S - f(u)]X$
  - where S is the given desired output, $u = sgn(\sum w^T{}_k X + b)$, where function $f(u) = +1$ for $u \geq 0$ and $f(u) = -1$ for $u < 0$, where 0 is the threshold value, and learning function is the loss or error $E = S - f(u)$ and w_0 is ***random value***
- With this law, a neuron receives input signals and if the sum of the input signals exceeds a certain threshold, it determines whether it will return an output or will not return an output
- In the context of supervised learning, we can then use this to predict the class of a sample in a classification operation

Learning Rules in Neural Networks

- **Cont'd….**
- **Delta learning law** : >> continuous learning rule
  - Learning law is only applicable to differentiable output function or neurons that perform nonlinear activation >> law is also called ***continuous perceptron learning rule***
  - Law depends on the derivative of the output function $f(.)$
  - Change in weight is based on the error between desired output and the actual output value for a given input >> law represents supervised learning
  - Weights are adjusted according to the relationship :
    - $\Delta W_k = \eta[S - f(u)]f^1(u)X$

Learning Rules in Neural Networks

- **Cont'd….**
  - >> $\Delta W_k = \eta[S - f(u)]f^1(u)X$ where $f'(u)$ is a derivative and $f(u)$ is a nonlinear activation function such as logistic sigmoid, S is the given desired output, $u = sgn(\sum w^T{}_k X + b)$
  - >> $f'(u) = f(u)\,[1 - f(u)]$
  - The weights can be initialized to any random values and will converge to final values eventually by repeated use of the input-output pattern pairs.
  - Convergence can be somewhat guaranteed when more layers of neurons are used in between the input and output.
  - Delta learning can be generalized to case of multiple layers.

Learning Rules in Neural Networks

- **Cont'd….**
- **Least Mean Squared (LMS) learning law** :
  - This learning law is a special case of the Delta learning law where the output function is assumed linear >> $f(x) = x$.
  - Change in weight is measured as proportional to negative gradient of error between desired output and continuous activation value >> law represent supervised learning
  - Weights are adjusted according to the relationship :
    - $\Delta W_k = \eta[S - u]X$ *where* $u = \sum w^T_k X + b$
  - Weights are initialized to any value and input-output pairs applied several time to achieve convergence of weights.

Learning Rules in Neural Networks

- **Cont'd….**
- **Hebbian learning law** :
  - Learning law states that the increment in the weight is proportional to the product of the input data and resulting output of the neuron.
  - Unlike the other laws, change in weight is not based on the error between desired output and actual output value >> the law represents an unsupervised learning
  - Weights are adjusted according to the relationship :
    - $\Delta W_k = \eta f(u) X$
  - Law requires weight initialization to small random values.

Learning Rules in Neural Networks

- **Cont'd….**
- **Winner-takes-all learning law :**
  - Learning law is based on competition of neurons in a layer for the right to give output taking into account the influence neurons in the neigbourhood.
  - For a given input vector X, the output from each neuron is computed using the weighted sum $\mathbf{w^T{}_k X}$ and the neuron that gives the maximum weighted sum is identified.
    - $\mathbf{w^T{}_k X = \max(w^T{}_j X)}$
  - Change in weight is based on the weight of winning neuron leading to the input >> represent an unsupervised learning

Learning Rules in Neural Networks

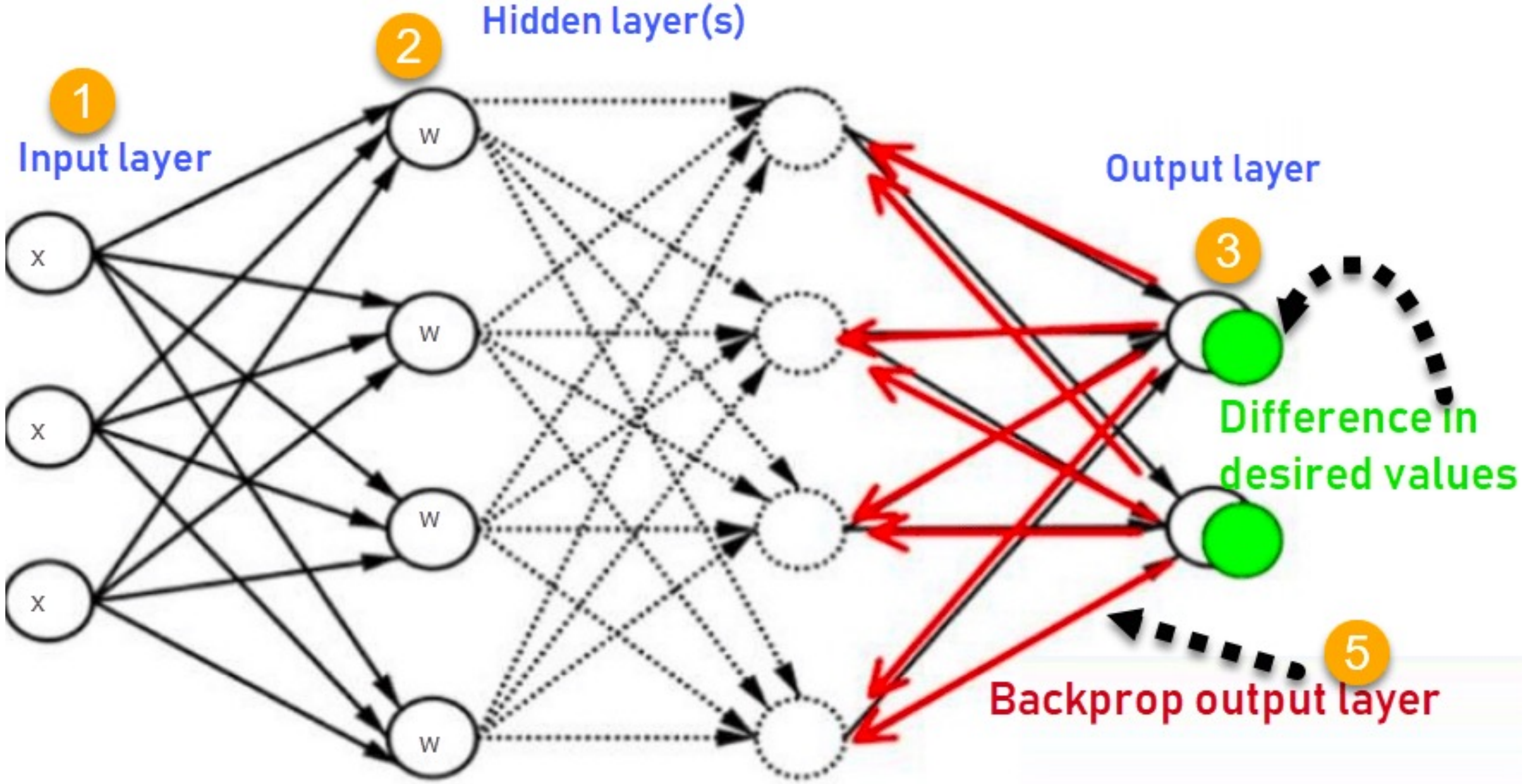
- **Cont'd….**
  - Weight vector leading to the winning neuron is adjusted according to the relationship :
    - $\Delta W_k = \eta(X - w_k)$
  - Law requires weight initialization to random values prior to learning and the vector lengths are normalized during the learning.

Multilayer network learning

- Back-propagation (error back propagation) is a method for learning the weights and biases in feed-forward networks.
- The method consist of two passes through the network layers:
  - ***Forward pass*** where input is applied to the network and the effects are propagated through all layers in the network to produce an output under fixed network weights.
  - ***Backward pass*** where the network weights are all adjusted in accordance with the error correction rule to adjust the weights (*actual response of network subtracted from target response to produce error signal which is back propagated*)
- Backpropagation is generalization of error correction learning

Multilayer network learning

- Cont'd…..



Requirements for creating neural networks

- Designing an MLP network for any given application involves:
  - input data and transformation to correct format;
  - number of input layer neurons;
  - number of output layer neurons;
  - architecture >> number hidden layers and neurons;
  - activation function for each layer;
  - network learning algorithm and parameters;
  - network learning rate;
  - network parameters >> epoch, goal etc;
  - network training and testing;
  - network performance evaluation;

Neural Network Model Process

- MLP network design process for application involves :
  - Define input and output data >> right format and structure
  - Define the network
  - Configure the network >> using the input and output data
  - Initialize the network parameters
  - Test the network prior to training >> apply sample input to the network to check if the same output is produced
  - Train the network >> define the network parameters for the training such as learning rate, epoch, goal, etc.
  - Test how well the network has trained >> use train data
  - Test network with unknown data

Practical ANN Application Example 1

- Suppose we want to design a perceptron neural network for the classification of children into either being "fat" or "not fat or slim" using input features of height (cm) and weight (kg); (x_1, x_2). Suppose we have 10 training samples, as follows:
  - >> P = [100,20; 100,26; 100,30; 100,32; 102,21; 105,22; 107,32; 110,35; 111,25; 114,24; 116,36; 118,27];
  - >> T = [0,1,1,1,0,0,1,1,0,0,1,0];
  - >> where "fat" = 1 and "slim" = 0 >> use hard limit
- Find the weight of the network after the training.
- Test the trained model using a child with input as [120,28]
- Draw a plot the data distribution and decision boundary.

ANN Development Solution 1

- We can use the function in Matlab for this exercise as follows:
  - >> net = newp([70 150; 10 70],1); % range of values of the inputs x_1 and x_2 and 1 number of neurons.
  - >> P = [100,20; 100,26; 100,30; 100,32; 102,21; 105,22; 107,32; 110,35; 111,25; 114,24; 116,36; 118,27]' ;
  - >> T = [0,1,1,1,0,0,1,1,0,0,1,0]' ;
  - >> view(net);
  - >> plotpv(P,T);
  - >> net.trainParam.epochs = 70 ; % vary this
  - >> net.trainParam.lr = 0.50 ; % vary this
  - >> net.trainParam.goal = 0.01; % vary this

ANN Development Solution 1

- **Cont'd….**
  - >> net = train(net,P,T);
  - >> view (net);
  - >> plotpc(net.IW{1},net.b{1}); % final weights/bias
  - >> test_data = [120,28]' ;
  - % or test_data = [120;28]
  - >> hold on;
  - >> plot(test_data(1),test_data(2),'+g');
  - >> Net_Out = sim(net,test_data);

Introduction to Multilayer Network

- We now look at a simple multilayer perceptron network with input layer and 3 computational layers of single neuron each.
- At the command line, type **nnstart**;
- Select the pattern recognition and classification tool to go through the inherent MLP network
- Select next and load the dataset for the Iris flowers (similar dataset was used for the ML training using tensorflow
- Select next and indicate your choice for data splitting
- Select next and indicate the number of hidden neurons.
- Select next and start with the network training.
- Select next and visualize the trained network performance.

Practical ANN Application Example 2

- We now explore the Google online ML model development to capture live data acquisition from objects for training and development of models for export to cloud for an application.
- Go to the following link below and for new project, select any of the three project sources for live data acquisition >> image project, audio project or pose project
  - **https://teachablemachine.withgoogle.com/**
- Default is 2-classes but you can add more classes depending on the desired number of output classes for your network
- You can acquire say, 200 sample data of varying degrees for each class >> for example, if it is apple, you can acquire different views of it to get say, 200 samples.