UNIT V
NEURAL NETWORKS

Perceptron - Multilayer perceptron, activation functions, network training – gradient descent optimization – stochastic gradient descent, error backpropagation, from shallow networks to deep networks –Unit saturation (aka the vanishing gradient problem) – ReLU, hyperparameter tuning, batch normalization, regularization, dropout.

PART A

1. Define neuron.

A neuron is a cell in the brain whose principal function is the collection, processing, and dissemination of electrical signals.

2. Define neural networks.

The brain's information-processing capacity is thought to emerge primarily from *networks* of such neurons. For this reason, some of the earliest **A1** work aimed to create artificial **neural networks.**

3. What are the two main categories of neural network structures?

- acyclic or feed-forward net-works
- cyclic or recurrent networks

4. Define Perceptron.

A network with all the inputs connected directly to the outputs is called a **single layer neural network,** or a **perceptron** network. Since each output unit is independent of the others-each weight affects only one of the outputs.

5. Define Multi Layer Perceptron.

- Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension.
- A multi-layer perception is a neural network that has multiple layers. To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.
- A multi-layer perceptron has one input layer and for each input, there is one neuron (or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes. A schematic diagram of a Multi-Layer Perceptron (MLP).

6. Define Activation function.

- In an artificial neural network, the function which takes the incoming signals as input and produces the output signal is known as the activation function.

7. List down the advantages of multilayer perceptron.

- It can be used to solve complex nonlinear problems.
- It handles large amounts of input data well.
- Makes quick predictions after training.
- The same accuracy ratio can be achieved even with smaller samples.

8. How do you train a neural model?

1. First an ANN will require a **random weight initialization**
2. Split the dataset in **batches (batch size)**
3. Send the batches 1 by 1 to the GPU
4. Calculate the **forward pass** (what would be the output with the current weights)
5. Compare the calculated output to the expected output **(loss)**
6. Adjust the **weights** (using the **learning rate** increment or decrement) according to the **backward pass (backward gradient propagation)**.
7. Go back to step 2.

9. List out the activation functions.

1. ReLU Function
2. Sigmoid Function
3. Linear Function
4. Tanh Function
5. Softmax Function

10. Define Gradient Descent Optimization.

- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea is to tweak parameters iteratively in order to minimize the cost function.
- An important parameter of Gradient Descent (GD) is the size of the steps, determined by the learning rate hyperparameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.

11. List the types of Gradient Descent.

1.Batch Gradient Descent
2.Stochastic Gradient Descent
3.Mini-batch Gradient Descent

12. Define Stochastic Gradient Descent (SGD).

- ➢ In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.
- ➢ In Gradient Descent, there is a term called "batch" which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration.
- ➢ In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.

13. What are the advantages and disadvantages of SGD?

Advantages:

- ✓ **Speed:** SGD is faster than other variants of Gradient Descent.
- ✓ **Memory Efficiency:**it is memory-efficient and can handle large datasets that cannot fit into memory.
- ✓ **Avoidance of Local Minima:** Due to the noisy updates in SGD, it has the ability to escape from local minima and converge to a global minimum.

Disadvantages:

- ✓ **Noisy updates:** The updates in SGD are noisy and have a high variance, which can make the optimization process less stable and lead to oscillations around the minimum.
- ✓ **Slow Convergence:** SGD may require more iterations to converge to the minimum since it updates the parameters for each training example one at a time.
- ✓ **Sensitivity to Learning Rate:** The choice of learning rate can be critical in SGD since using a high learning rate can cause the algorithm to overshoot the minimum, while a low learning rate can make the algorithm converge slowly.
- ✓ **Less Accurate:** Due to the noisy updates, SGD may not converge to the exact global minimum and can result in a suboptimal solution. This can be mitigated by using techniques such as learning rate scheduling and momentum-based updates.

14. Define Backpropagation.

- ➢ The neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as **backpropagation**.

15. List out the types of Backpropagation.

- ➢ Two Types of Backpropagation Networks are:
  1. Static Back-propagation
  2. Recurrent Backpropagation

- **Static back-propagation:**

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

- **Recurrent Backpropagation:**

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

14. What are the advantages and disadvantages of error backropagation?

Advantages:

- ✓ It does not have any parameters to tune except for the number of inputs.
- ✓ It is highly adaptable and efficient and does not require any prior knowledge about the network.
- ✓ It is a standard process that usually works well.

Disadvantages:

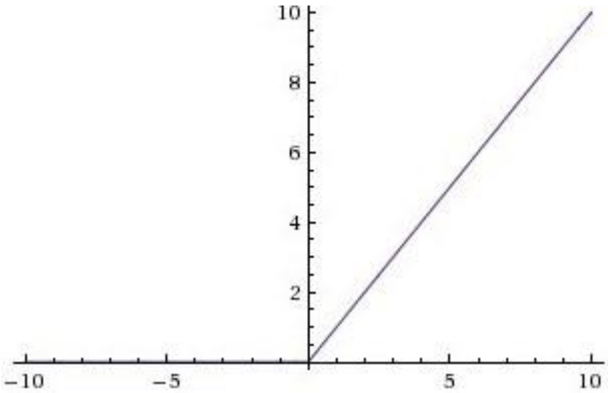
- ✓ The performance of backpropagation relies very heavily on the training data.
- ✓ Backpropagation needs a very large amount of time for training.
- ✓ Backpropagation requires a matrix-based method instead of mini-batch.

15. Define Unit saturation (aka the vanishing gradient problem).

- The vanishing gradient problem is an issue that sometimes arises when training machine learning algorithms through gradient descent. This most often occurs in neural networks that have several neuronal layers such as in a deep learning system, but also occurs in recurrent neural networks.
- The key point is that the calculated partial derivatives used to compute the gradient as one goes deeper into the network. Since the gradients control how much the network learns during training, the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

16. Define Rectified linear unit (ReLU).

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network. (see the below figure).
- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.



ReLU Function

- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.
- In simple words, RELU learns *much faster* than sigmoid and Tanh function.

17. Define Normalization.

- Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

18. Define Batch Normalization.

- **Batch normalization** is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer. A typical neural network is trained using a collected set of input data called **batch**.

19. Define Hyperparameter tuning.

- A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.
- However, there is another kind of parameter, known as ***Hyperparameters***, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

20. What are the two strategies of Hyperparameter tuning?

1. GridSearchCV
2. RandomizedSearchCV

21. Define GridSearchCV.

- In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.

22. Define RandomizedSearchCV.

- RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

23. Define Overfitting.

- When a model performs well on the training data and does not perform well on the testing data, then the model is said to have high generalization error. In other words, in such a scenario, the model has low bias and high variance and is too complex. This is called overfitting.

24. What is Regularization?

- Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Regularization helps choose a simple model rather than a complex one.

25. List out the Regularization Technique.

1. Hints
2. Weight decay:
3. Ridge Regression (or) L2 Regularization.
4. Lasso Regression (or) L1 Regularization.
5. Dropout

26. Define L1 Regularization and L2 Regularization.

Lasso Regression (or) L1 Regularization.

- Least Absolute Shrinkage and Selection Operator (or LASSO) Regression penalizes the coefficients to the extent that it becomes zero.
- It eliminates the insignificant independent variables. This regularization technique uses the L1 norm for regularization.

$$E' = E + \frac{\lambda}{2} \sum_i |w_i|$$

Ridge Regression or L2 Regularization

- The Ridge regression technique is used to analyze the model where the variables may be having multicollinearity.
- It reduces the insignificant independent variables though it does not remove them completely. This type of regularization uses the L_2 norm for regularization.

$$E' = E + \frac{\lambda}{2}\sum_i w_i^2$$

27. Define Dropout.

- **"Dropout"** in machine learning refers to the process of randomly ignoring certain nodes in a layer during training.

28. Difference between Shallow and Deep neural network.

Shallow Network	Deep Network
A shallow neural network has only one hidden layer between the input and output layers	A deep neural network has multiple hidden layers
A shallow network might be used for simple tasks like image classification.	A deep network might be used for more complex tasks like image segmentation or natural language processing.
A shallow network is that it is computationally less expensive to train, and can be sufficient for simple tasks	It is more computationally expensive to train and may require more data to avoid overfitting.
It may not be powerful enough to capture complex patterns in the data.	It capture more complex patterns in the data and potentially achieve higher accuracy.

29. Difference between Stochastic Gradient Descent and Gradient Descent.

Stochastic Gradient Descent	Gradient Descent
Computes gradient using the whole Training sample	Computes gradient using a single Training sample
Not suggested for huge training samples.	Can be used for large training samples.
Gives optimal solution given sufficient time to converge.	Gives good solution but not optimal.
Deterministic in nature.	Stochastic in nature.
Convergence is slow.	Reaches the convergence much faster.

30. What is meant by Training set?

- Training set is a set of pairs of input patterns with corresponding desired output patterns. Each pair represents how the network is supposed to respond to a particular input. The network is trained to respond correctly to each input pattern from the training set.

31. What is meant by Test set?

- The test set is the dataset that the model is trained on.

32. Difference between Data Mining and Machine learning.

Data Mining	Machine Learning
Extracting useful information from large amount of data	Introduce algorithm from data as well as from past experience
Used to understand the data flow	Teaches the computer to learn and understand from the data flow
Human interference is more in it.	No human effort required after design
Huge databases with unstructured data	Existing data as well as algorithms
Historical data	Historical and real-time data
Data mining is more of a research using methods like machine learning	Self learned and trains system to do the intelligent task

33. Define Forward Pass.

- **Forward Propagation** is the way to move from the Input layer (left) to the Output layer (right) in the **neural network**. A neural network can be understood by a collection of connected input/output nodes.

34. Define Backward Pass.

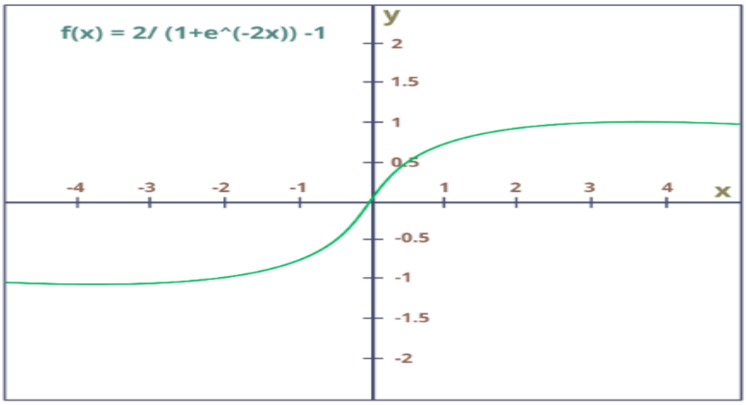
- In the backward pass, the flow is reversed so that we start by propagating the error to the output layer until reaching the input layer passing through the hidden layer(s).
- The process of propagating the network error from the output layer to the input layer is called backward propagation, or simple backpropagation.

35. Define Tanh Function.

- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It’s actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
- **Equation :-**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

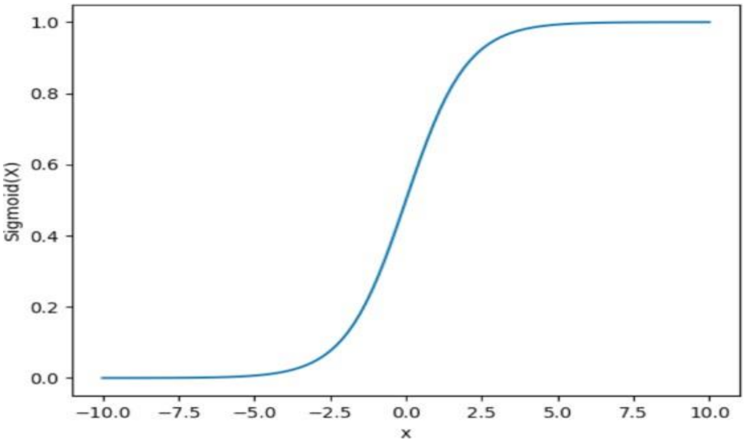
- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.



Tanh Function

36. Define Sigmoid Function.

- ✓ It is a function which is plotted as **'S'** shaped graph.
- ✓ **Equation :** A = 1/(1 + e^{-x})
- ✓ **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- ✓ **Value Range :** 0 to 1
- ✓ **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.



37. What is Leaky ReLU?

- In the *leaky ReLU* (the output is also linear on the negative side but with a smaller slope, just enough to make sure that there will be updates for negative activations, albeit small:

$$\text{leaky-ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha a & \text{otherwise} \end{cases}$$

38. What is Artificial Neuron?

- An artificial neuron is a connection point in an artificial neural network. Artificial neural networks, like the human body's biological neural network, have a layered architecture and each network node (connection point) has the capability to process input and forward output to other nodes in the network.

39. What is meant by Feed forward neural network?

- "The process of receiving an input to produce some kind of output to make some kind of prediction is known as Feed Forward."
- Feed Forward neural network is the core of many other important neural networks such as convolution neural network.

40. Define Bias.

- Neural network bias can be defined as the constant which is added to the product of features and weights. It is used to offset the result. It helps the models to shift the activation function towards the positive or negative side.

PART B

1. Explain in detail about Perceptron.

- ✓ The *perceptron* is the basic processing element.
- ✓ It has inputs that may come from the environment or may be the outputs of other perceptrons.
- ✓ Associated with each input, $x_j \in \Re$, $j=1, \ldots, d$, is a *connection weight*, or *synaptic weight* $w_j \in \Re$, and the output, y, in the simplest case is a weighted sum of the inputs **(see figure 5.1)**

$$y = \sum_{j=1}^{d} w_j x_j + w_0 \longrightarrow 1$$

- w_0 is the intercept value to make the model more general

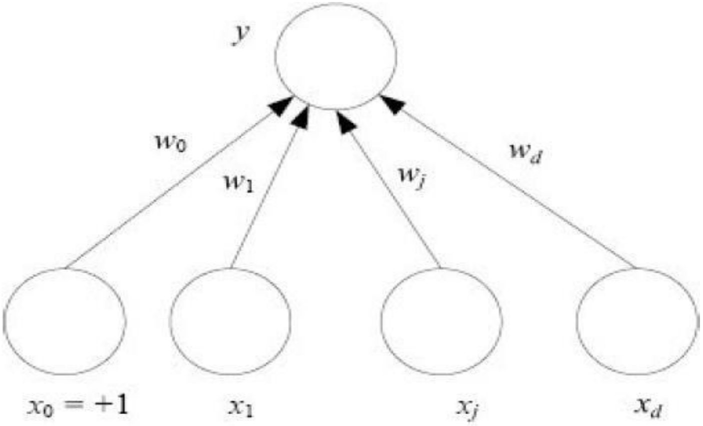


Figure 5.1 Simple Perceptron

- ✓ We can write the output of the perceptron as a dot product

$$\mathbf{Y=w^Tx}$$

- where $w = [w_0, w_1, \ldots, w_d]^T$ and $x = [1, x_1, \ldots, x_d]^T$ are *augmented* vectors to include also the bias weight and input.

- ✓ During testing, with given weights, *w*, for input *x*, we compute the output *y*. To implement a given task, we need to *learn* the weights *w*, the parameters of the system, such that correct outputs are generated given the inputs.
- ✓ When $d = 1$ and x is fed from the environment through an input unit, we have

$$y = w\,x + w_0$$

where w as the slope and w_0 as the intercept

- ✓ Thus this perceptron with one input and one output can be used to implement a linear fit. With more than one input, the line becomes a (hyper)plane, and the perceptron with more than one input can be used to implement multivariate linear fit.
- ✓ The perceptron as defined in equation 1 defines a hyperplane and as such can be used to divide the input space into two:
  - o the half-space where it is positive and the half-space where it is negative
- ✓ By using it to implement a linear discriminant function, the perceptron can separate two classes by checking the sign of the output. If we define $s(\cdot)$ as the *threshold function*

$$s(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

then we can,

$$\text{choose} \begin{cases} C_1 & \text{if } s(\boldsymbol{w}^T\boldsymbol{x}) > 0 \\ C_2 & \text{otherwise} \end{cases}$$

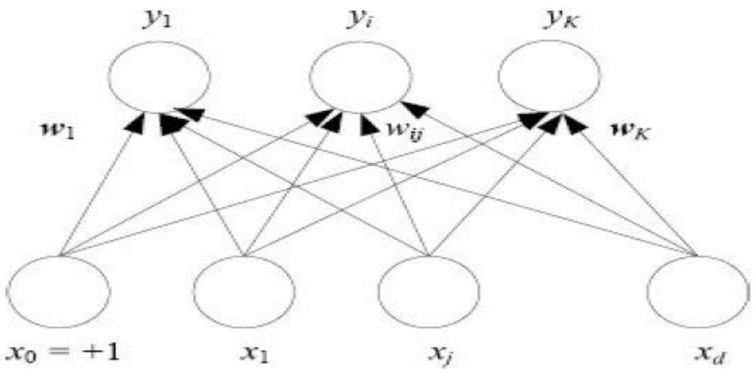


Figure 5.2 *K* parallel perceptrons

From the Figure 5.2,

- ✓ x_j, $j = 0, \ldots, d$ are the inputs and y_i, $i = 1, \ldots, K$ are the outputs. w_{ij} is the weight of the connection from input *xj* to output *yi*. Each output is a weighted sum of the inputs. When used for a *K*-class classification problem, there is a postprocessing to choose the maximum, or softmax if we need the posterior probabilities.
- ✓ it is assumed that a hyperplane $w^Tx = 0$ can be found that separates $x^t \in C1$ and $x^t \in C2$. If at a later stage we need the posterior probability.
- ✓ Example, to calculate risk—we need to use the sigmoid function at the output as

$$\begin{aligned} o &= \boldsymbol{w}^T\boldsymbol{x} \\ y &= \text{sigmoid}(o) = \frac{1}{1+\exp[-\boldsymbol{w}^T\boldsymbol{x}]} \end{aligned}$$

- ✓ When there are *K* > 2 outputs, there are *K* perceptrons, each of which has a weight vector *wi* (see figure 5.2)

$$\begin{aligned} y_i &= \sum_{j=1}^{d} w_{ij}x_j + w_{i0} = \boldsymbol{w}_i^T\boldsymbol{x} \\ \boldsymbol{y} &= \mathbf{W}\boldsymbol{x} \end{aligned}$$

- ✓ where w_{ij} is the weight from input x_j to output y_i. W is the *K* ×(*d* + 1) weight matrix of w_{ij} whose rows are the weight vectors of the *K* perceptrons. When used for classification, during testing, we

$$\text{choose } C_i \text{ if } y_i = \max_k y_k$$

- ✓ Each perceptron is a *local* function of its inputs and synaptic weights. In classification, if we need the posterior probabilities and use the softmax, we need the values of all outputs.
- ✓ Implementing this as a neural network results in a two-stage process, where the first calculates the weighted sums, and the second calculates the softmax values; but we denote this as a single layer:

$$\begin{aligned} o_i &= \boldsymbol{w}_i^T\boldsymbol{x} \\ y_i &= \frac{\exp o_i}{\sum_k \exp o_k} \end{aligned}$$

- ✓ a linear transformation from a *d* dimensional space to a *K*-dimensional space and can also be used for dimensionality reduction if *K* < *d*. we have a two-layer network where the first layer of perceptrons implements the linear transformation and the second layer implements the linear regression or classification in the new space.

2. Explain in detail about Multi Layer Perceptron.

- ✓ Multi-layer perception is also known as MLP. It is fully connected dense layers, which transform any input dimension to the desired dimension.
- ✓ A multi-layer perception is a neural network that has multiple layers. To create a neural network we combine neurons together so that the outputs of some neurons are inputs of other neurons.

- ✓ A multi-layer perceptron has one input layer and for each input, there is one neuron (or node), it has one output layer with a single node for each output and it can have any number of hidden layers and each hidden layer can have any number of nodes. A schematic diagram of a Multi-Layer Perceptron (MLP) is shown in **Figure 5.3**.

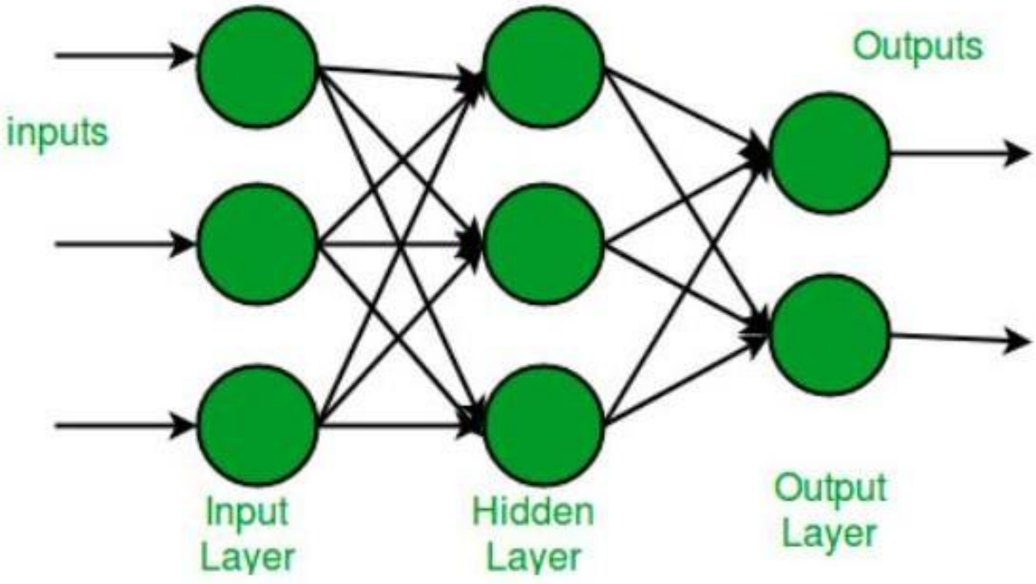


Figure 5.3 A Multi-Layer Perceptron (MLP)

- ✓ In the multi-layer perceptron diagram, we can see that there are three inputs and thus three input nodes and the hidden layer has three nodes.
- ✓ The output layer gives two outputs, therefore there are two output nodes. The nodes in the input layer take input and forward it for further process, in the diagram above the nodes in the input layer forwards their output to each of the three nodes in the hidden layer, and in the same way, the hidden layer processes the information and passes it to the output layer.
- ✓ Every node in the multi-layer perception uses a sigmoid activation function. The sigmoid activation function takes real values as input and converts them to numbers between 0 and 1 using the sigmoid formula.

$$\mathbf{(x)=1/(1+(exp(-x))}$$

- ✓ A perceptron that has a single layer of weights can only approximate linear functions of the input and cannot solve problems like the XOR, where the discrimininant to be estimated is nonlinear.

- ✓ Similarly, a perceptron cannot be used for nonlinear regression. This limitation does not apply to feedforward networks with an intermediate or *hidden layer* between the input and the output layers.
- ✓ If used for classification, such *multilayer perceptrons* (MLP) can implement nonlinear discriminants and, if used for regression, can approximate nonlinear functions of the input.

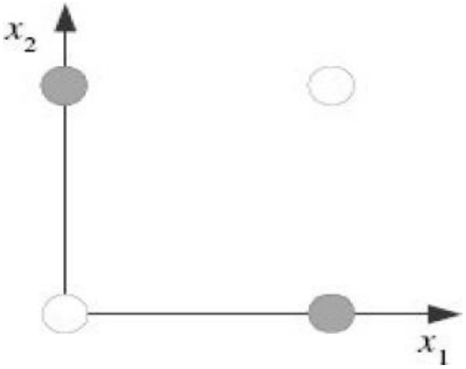


Figure 5.4 XOR problem is not linearly separable.

From the Figure 5.4, We cannot draw a line where the empty circles are on one side and the filled circles on the other side.

- ✓ Input *x* is fed to the input layer (including the bias), the "activation" propagates in the forward direction, and the values of the hidden units z_h are calculated **(see Figure 5.5).** Each hidden unit is a perceptron by itself and applies the nonlinear sigmoid function to its weighted sum:

$$z_h = \text{sigmoid}(\mathbf{w}_h^T \mathbf{x}) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}, \; h = 1, \ldots, H$$

- ✓ The output *yi* are perceptrons in the second layer taking the hidden units as their inputs

$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

- ✓ Where there is also a bias unit in the hidden layer, which we denote by z_0, and v_{i0} are the bias weights. The input layer of x_j is not counted since no computation is done there and when there is a hidden layer, this is a two-layer network.
- ✓ In a two-class discrimination task, there is one sigmoid output unit and when there are $K > 2$ classes, there are *K* outputs with softmax as the output nonlinearity.

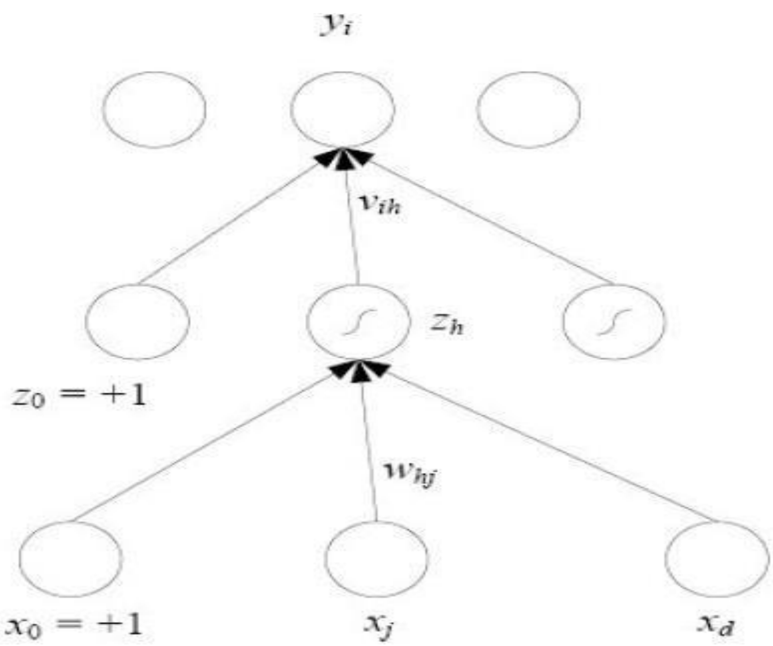
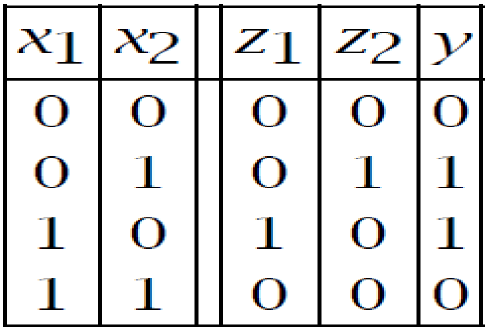


Figure 5.5 The structure of a multilayer perceptron.

- **From the Figure 5.5** x_j, $j = 0,\dots, d$ are the inputs and z_h, $h = 1, \dots, H$ are the hidden units where H is the dimensionality of this hidden space. z_0 is the bias of the hidden layer. y_i, $i = 1, \dots, K$ are the output units. w_{hj} are weights in the first layer, and v_{ih} are the weights in the second layer.

- The MLP that implements XOR with two hidden units that implement the two ANDs and the output that takes an OR of them.



x_1	x_2	z_1	z_2	y
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

- One is not limited to having one hidden layer, and more hidden layers with their own incoming weights can be placed after the first hidden layer with sigmoid hidden units, thus calculating nonlinear functions of the first layer of hidden units and implementing more complex functions of the inputs.

3. Explain in Detail about Activation function.

- ➢ Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output. Each input is separately weighted, and the sum is passed through a function known as an activation function or transfer function.
- ➢ In an artificial neural network, the function which takes the incoming signals as input and produces the output signal is known as the activation function.
- ➢ Neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.
- ➢ The neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as **Backpropagation**.
- ➢ Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

Need for Non-linear activation function

- ✓ A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.
- ✓ The two main categories of activation functions are:

1. Linear Activation Function
2. Non-linear Activation Functions

Linear Activation Function

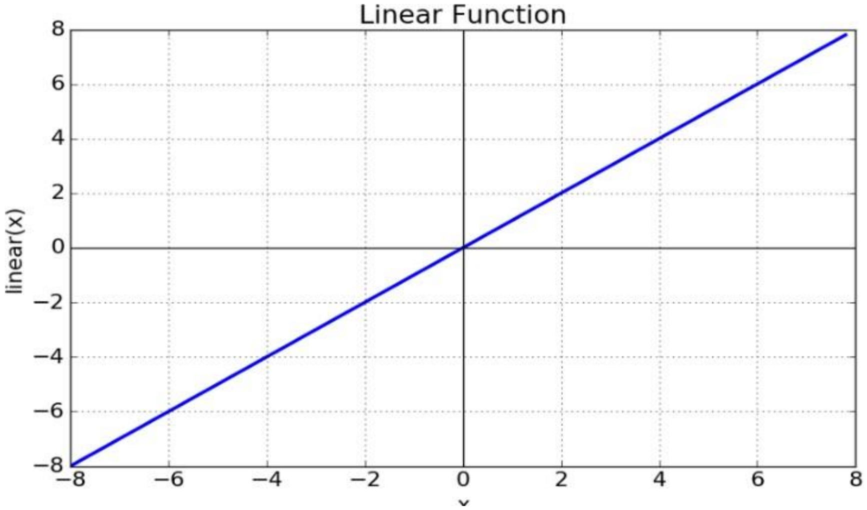


Figure 5.6 Linear Activation Function

Non-linear Activation Functions

1. Linear Function

- Linear function has the equation similar to as of a straight line i.e. **y = x (see in Figure 5.6).**
- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
- **Range :** -inf to +inf
- **Uses : Linear activation function** is used at just one place i.e. output layer.

2. Sigmoid Function

- It is a function which is plotted as **'S'** shaped graph **(Refer Figure 5.7)** .
- **Equation :** A = 1/(1 + e^{-x})
- **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
- **Value Range :** 0 to 1
- **Uses :** Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

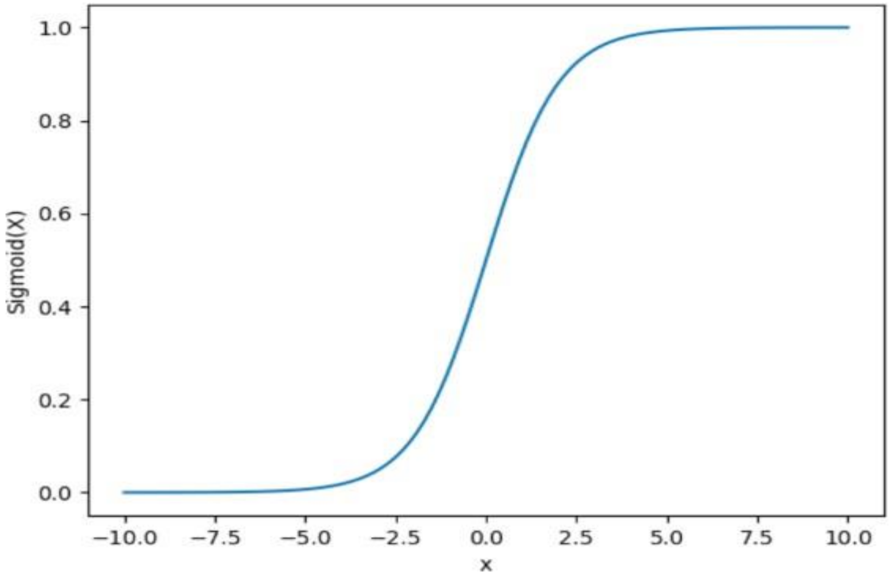


Figure 5.7 Sigmoid Function

3. Tanh Function

- The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It's actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other **(see in Figure 5.8).**

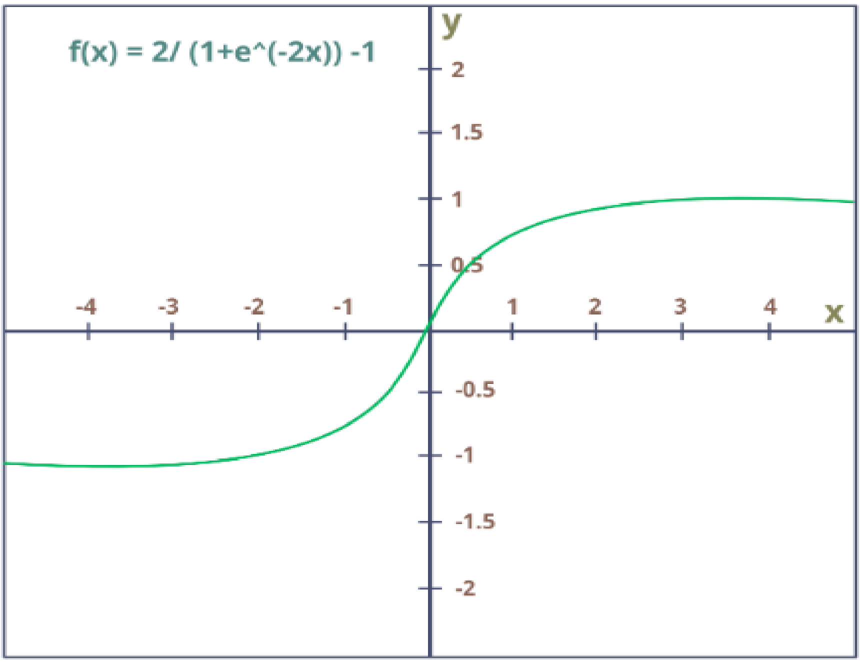


Figure 5.8 Tanh Function

- **Equation :-**

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

- **Value Range :-** -1 to +1
- **Nature :-** non-linear
- **Uses :-** Usually used in hidden layers of a neural network as it's values lies between **-1 to 1** hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

4. ReLU Function

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.

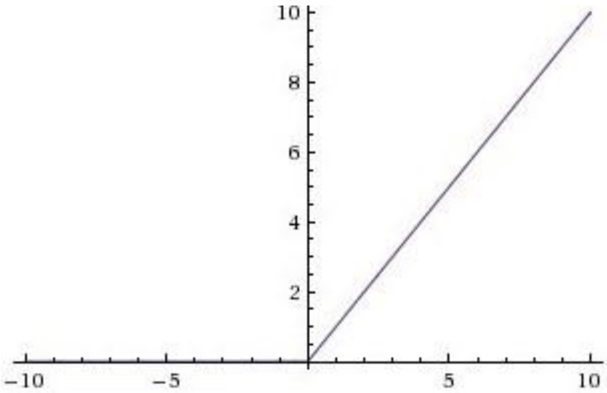


Figure 5.9 ReLU FUNCTION

- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise **(see in Figure 5.9).**
- **Value Range :-** [0, inf]
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.
- In simple words, RELU learns *much faster* than sigmoid and Tanh function.

5. Softmax Function

- ✓ It is a subclass of the sigmoid function, the softmax function comes in handy when dealing with multiclass classification issues.
- ✓ Used frequently when managing several classes. In the output nodes of image classification issues, the softmax was typically present. The softmax function would split by the sum of the outputs and squeeze all outputs for each category between 0 and 1.
- ✓ The output unit of the classifier, where we are actually attempting to obtain the probabilities to determine the class of each input, is where the softmax function is best applied.

4. Discuss in detail about how the network is training.

- A **biological neuron** is composed of multiple **dendrites**, a **nucleus** and a axon. When a stimuli is sent to the brain, it is received through the **synapse** located at the extremity of the dendrite.
- When a **stimuli** arrives at the brain it is transmitted to the neuron via the **synaptic receptors** which **adjust the strength of the signal sent to the nucleus**. This message

is **transported** by the **dendrites** to the **nucleus** to then be **processed** in **combination** with other signals emanating from other receptors on the other dendrites. T**hus the combination of all these signals takes place in the nucleus.**

- After processing all these signals, **the nucleus will emit an output signal through its single axon**. The axon will then stream this signal to several other downstream neurons via its **axon terminations**. Thus a neuron analysis is pushed in the subsequent layers of neurons.
- On the other hand, **artificial neural networks** are built on the principle of bio-mimicry. **External stimuli (the data),** whose signal strength is adjusted by the **neuronal weights**, **circulates to the neuron** via the dendrites. The result of the calculation – called the **output** – is then re-transmitted (via the axon) to several other neurons and then subsequent layers are combined, and so on.(see in **Figure 5.10 & 5.11**).

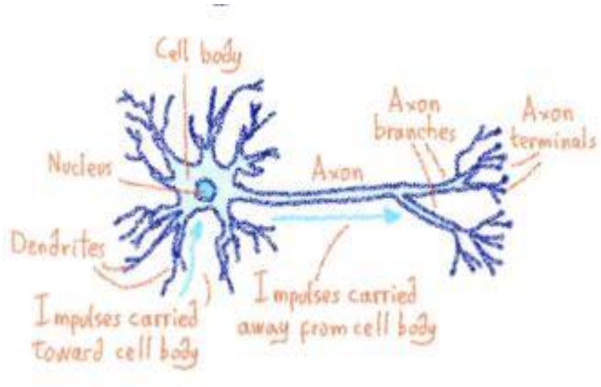

Cell body
Axon branches
Axon terminals
Nucleus
Axon
Dendrites
Impulses carried toward cell body
Impulses carried away from cell body

Figure 5.10 Biological Neuron

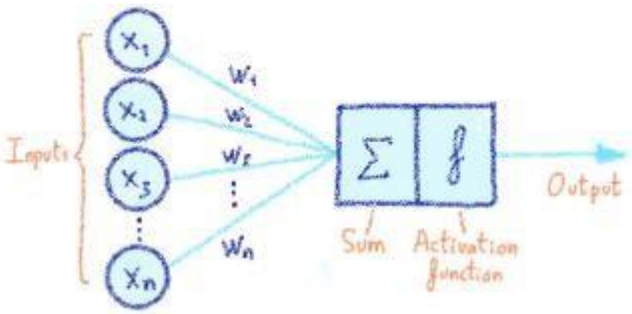


Figure 5.11 Artificial Neuron

Example:

1. Decide on the **number of output classes** (meaning the number of image classes – for example two for cat vs dog)
2. Draw as many computation units as the **number of output classes** (congrats you just create the **Output Layer** of the ANN)

3. Add as many **Hidden Layers** as needed within the defined **architecture**.
4. Stack those **Hidden Layers** to the **Output Layer** using **Neural Connections**
5. It is important to understand that the **Input Layer** is basically a layer of data ingestion
6. Add an **Input Layer** that is adapted to ingest your data
7. Assemble many Artificial Neurons together in a way where the **output** (axon) an **Neuron** on a given **Layer** is (one) of the **input** of another **Neuron** on a subsequent **Layer**. As a consequence, the **Input Layer** is linked to the **Hidden Layers** which are then linked to the **Output Layer** using **Neural Connections** (also shown in **Figure 5.12**).

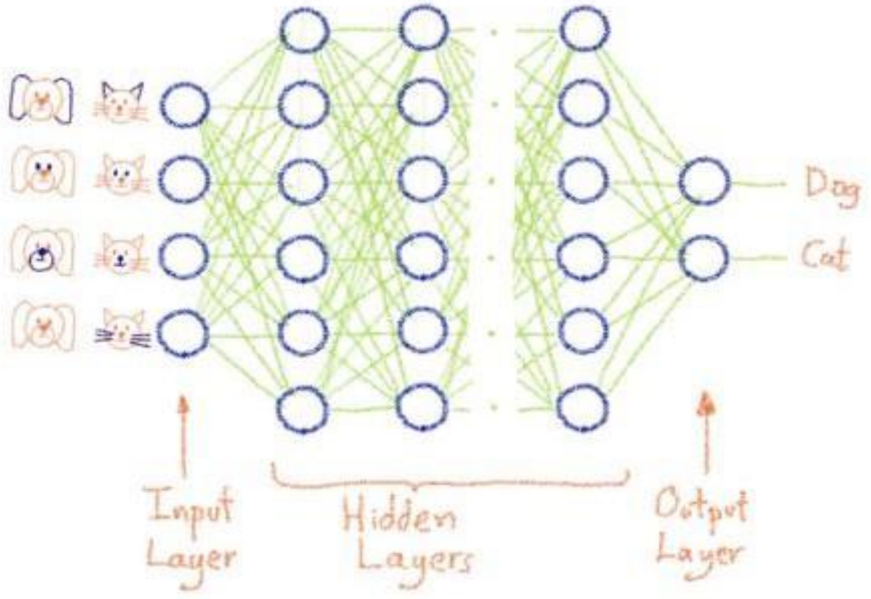


Figure 5.12 A Neural Network Architecture

Train an Artificial Neural Network

- All **Neurons** of a given **Layer** are generating an **Output**, but they don't have the same **Weight** for the next **Neurons Layer**. This means that if a Neuron on a layer observes a given pattern it might mean less for the overall picture and will be partially or completely muted. This is called **Weighting**.
- A **big weight means that the Input is important** and of course **a small weight means that we should ignore it**. Every **Neural Connection** between **Neurons** will have **an associated Weight**.
- **Weights** will be adjusted over the training to fit the **objectives** we have set (recognize that a dog is a dog and that a cat is a cat).

- **In simple terms:** Training a Neural Network means finding the appropriate Weights of the Neural Connections thanks to a feedback loop called Gradient Backward propagation .

Steps to Training an Artificial Neural Network

1. First an ANN will require a **random weight initialization**
2. Split the dataset in **batches (batch size)**
3. Send the batches 1 by 1 to the GPU
4. Calculate the **forward pass** (what would be the output with the current weights)
5. Compare the calculated output to the expected output **(loss)**
6. Adjust the **weights** (using the **learning rate** increment or decrement) according to the **backward pass (backward gradient propagation)**.
7. Go back to step 2

5. Discuss in detail about Gradient descent optimization Algorithm.

- Gradient Descent is a generic optimization algorithm capable of finding optimal solutions to a wide range of problems.
- The general idea is to tweak parameters iteratively in order to minimize the cost function.
- An important parameter of Gradient Descent (GD) is the size of the steps, determined by the learning rate hyperparameters. If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time, and if it is too high we may jump the optimal value.

Types of Gradient Descent:

- Typically, there are three types of Gradient Descent:

1. Batch Gradient Descent

Batch Gradient Descent involves calculations over the full training set at each step as a result of which it is very slow on very large training data. Thus, it becomes very computationally expensive to do Batch GD.

2. Stochastic Gradient Descent

In SGD, only one training example is used to compute the gradient and update the parameters at each iteration. This can be faster than batch gradient descent but may lead to more noise in the updates.

3. Mini-batch Gradient Descent

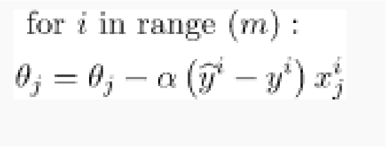
In mini-batch gradient descent, a small batch of training examples is used to compute the gradient and update the parameters at each iteration. This can be a good compromise between batch gradient descent and SGD, as it can be faster than batch gradient descent and less noisy than SGD.

6. Explain in detail about Stochastic gradient descent .

Stochastic Gradient Descent:

- In Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.
- In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration.
- In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset. Although using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, the problem arises when our dataset gets big.
- Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima are reached. Hence, it becomes computationally very expensive to perform.
- This problem is solved by Stochastic Gradient Descent. In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration.
- The sample is randomly shuffled and selected for performing the iteration.

SCD Algorithm



for i in range (m) :

$$\theta_j = \theta_j - \alpha \left(\widehat{y}^i - y^i\right) x_j^i$$

- In SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.
- In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm. But that doesn’t matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with a significantly shorter training time **(see in Figure 5.13 & 5.14)**.

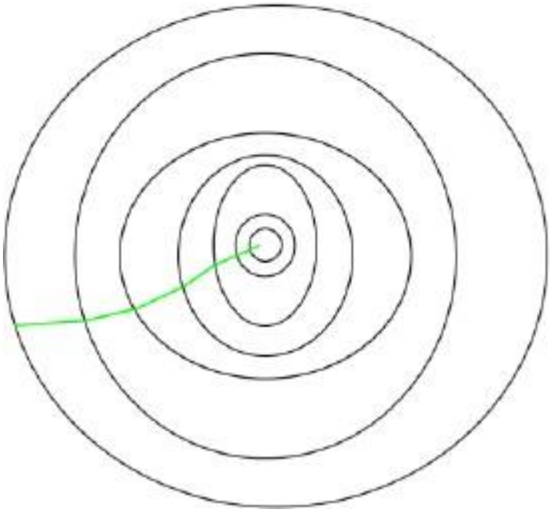


Figure 5.13 The path taken by Batch Gradient Descent

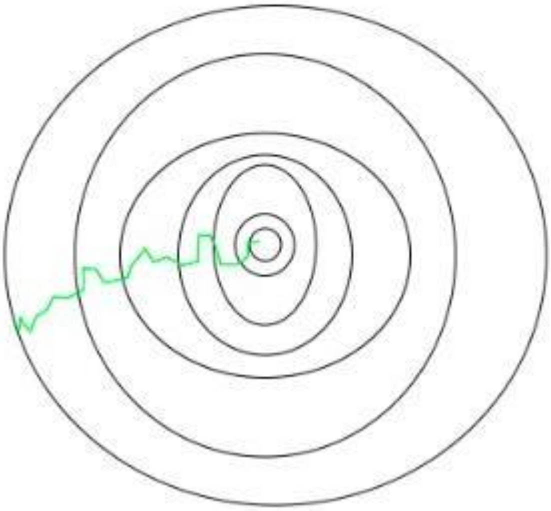


Figure 5.14 The path taken by Stochastic Gradient Descent

- ✓ SGD is generally noisier than typical Gradient Descent, it usually took a higher number of iterations to reach the minima, because of its randomness in its descent.
- ✓ Even though it requires a higher number of iterations to reach the minima than typical Gradient Descent, it is still computationally much less expensive than typical Gradient Descent. Hence, in most scenarios, SGD is preferred over Batch Gradient Descent for optimizing a learning algorithm.

Advantages:

- ✓ **Speed:** SGD is faster than other variants of Gradient Descent.
- ✓ **Memory Efficiency:**it is memory-efficient and can handle large datasets that cannot fit into memory.
- ✓ **Avoidance of Local Minima:** Due to the noisy updates in SGD, it has the ability to escape from local minima and converge to a global minimum.

Disadvantages:

- ✓ **Noisy updates:** The updates in SGD are noisy and have a high variance, which can make the optimization process less stable and lead to oscillations around the minimum.
- ✓ **Slow Convergence:** SGD may require more iterations to converge to the minimum since it updates the parameters for each training example one at a time.
- ✓ **Sensitivity to Learning Rate:** The choice of learning rate can be critical in SGD since using a high learning rate can cause the algorithm to overshoot the minimum, while a low learning rate can make the algorithm converge slowly.
- ✓ **Less Accurate:** Due to the noisy updates, SGD may not converge to the exact global minimum and can result in a suboptimal solution. This can be mitigated by using techniques such as learning rate scheduling and momentum-based updates.

7. Explain in detail about error backpropagation.

- ➢ **Backpropagation** is one of the important concepts of a neural network. or a single training example, **Backpropagation** algorithm calculates the gradient of the **error function**.
- ➢ Backpropagation algorithms are a set of methods used to efficiently train artificial neural networks following a gradient descent approach which exploits the chain rule.
- ➢ The **main features of Backpropagation are the iterative**, recursive and efficient method through which it calculates the updated weight to improve the network until it is not able to perform the task for which it is being trained.
- ➢ Derivatives of the activation function to be known at network design time is required to Backpropagation.

How Backpropagation Algorithm Works?

- ➢ The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.(see in **Figure 5.16**)

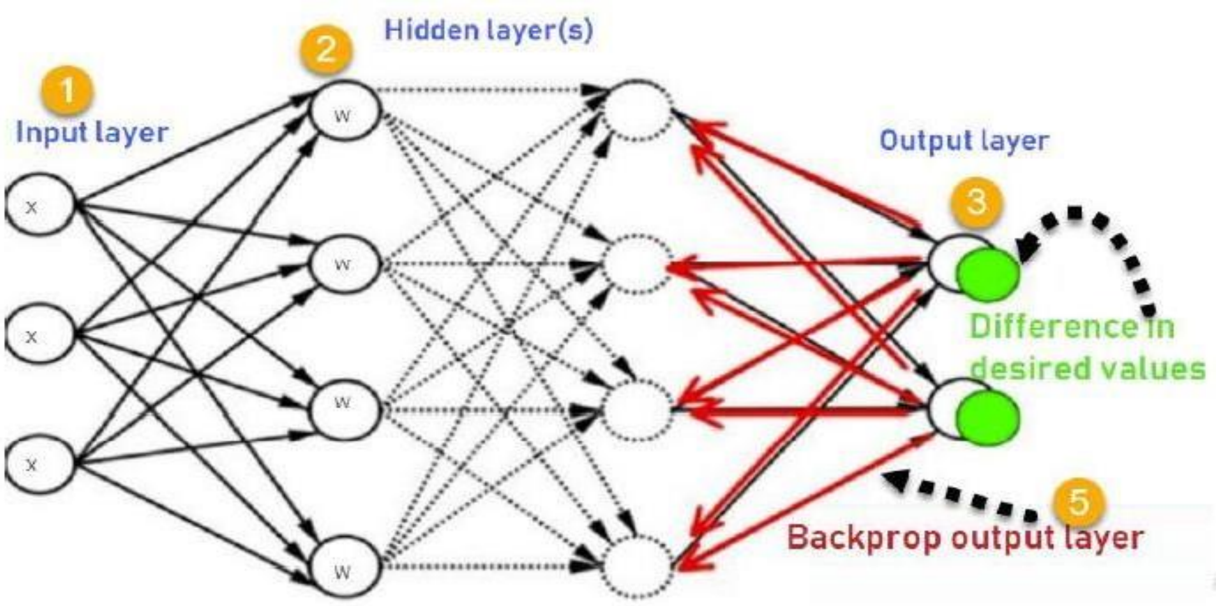


Figure 5.16 Back propagation neural network

1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

$$\mathbf{Error_B = Actual\ Output - Desired\ Output}$$

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
6. Keep repeating the process until the desired output is achieved

Types of Backpropagation Networks

- Two Types of Backpropagation Networks are:
  3. Static Back-propagation
  4. Recurrent Backpropagation

Static back-propagation:

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent Backpropagation:

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

Advantages:

- ✓ It does not have any parameters to tune except for the number of inputs.
- ✓ It is highly adaptable and efficient and does not require any prior knowledge about the network.
- ✓ It is a standard process that usually works well.

Disadvantages:

- ✓ The performance of backpropagation relies very heavily on the training data.
- ✓ Backpropagation needs a very large amount of time for training.
- ✓ Backpropagation requires a matrix-based method instead of mini-batch.

8. Explain in detail about Unit saturation (aka the vanishing gradient problem).

- ➢ The vanishing gradient problem is an issue that sometimes arises when training machine learning algorithms through gradient descent. This most often occurs in neural networks that have several neuronal layers such as in a deep learning system, but also occurs in recurrent neural networks.
- ➢ The key point is that the calculated partial derivatives used to compute the gradient as one goes deeper into the network. Since the gradients control how much the network learns during training, the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

The problem:

- ➢ As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

Why:

- ➢ Certain activation functions, like the sigmoid function, squishes a large input space into a small input space between 0 and 1. Therefore, a large change in the input of the sigmoid function will cause a small change in the output. Hence, the derivative becomes small.

The sigmoid function and its derivative

- ➢ As an example, the below **Figure 5.17** is the sigmoid function and its derivative. Note how when the inputs of the sigmoid function becomes larger or smaller (when $|x|$ becomes bigger), the derivative becomes close to zero.

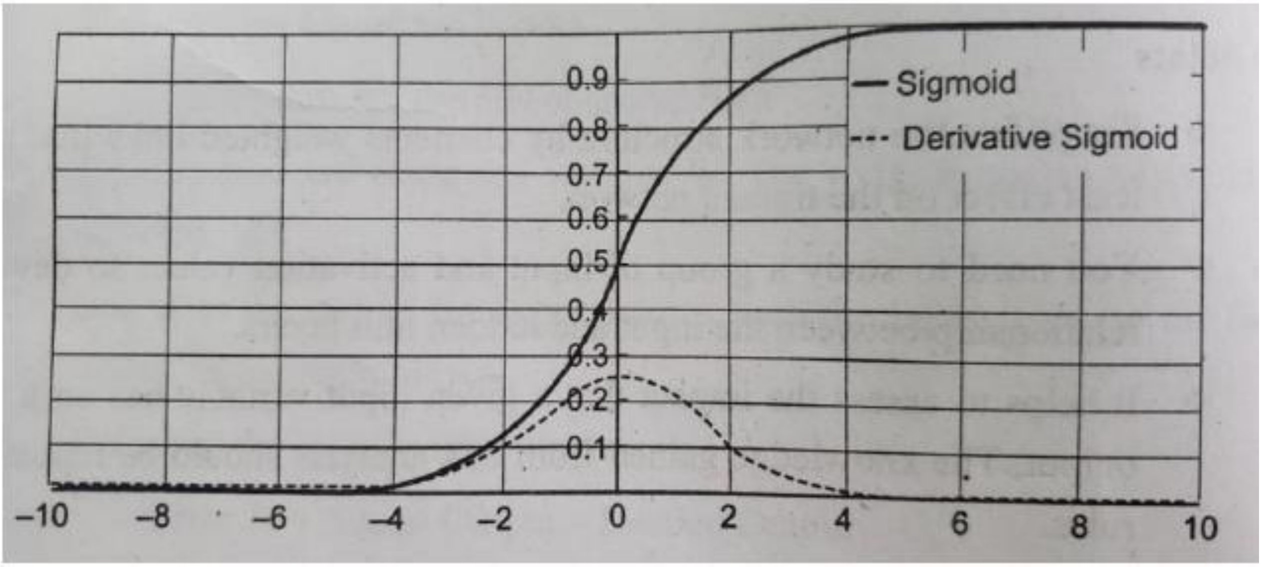


Figure 5.17 The sigmoid function and its derivative

Why it's significant:

- For shallow network with only a few layers that use these activations, this isn't a big problem. However, when more layers are used, it can cause the gradient to be too small for training to work effectively. Gradients of neural networks are found using backpropagation. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network (from the final layer to the initial) to compute the derivatives of the initial layers.
- However, when n hidden layers use activation like the sigmoid an function, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers. A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

Solution:

- The simplest solution is to use other activation functions, such as ReLU, which doesn't cause a small derivative. Residual networks are another solution, as they provide residual connections straight to earlier layers.
- The residual connection directly adds the value at the beginning of the block, x, to the end of the block (F(x) + x). This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.(see in **Figure 5.18)**

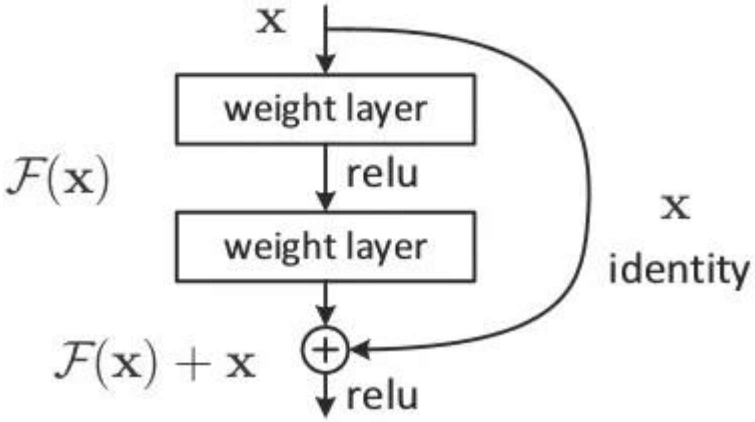

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 5.18 A residual block

- Finally, batch normalization layers can also resolve the issue. As stated before, the problem arises when a large input space is mapped to a small one, causing the derivatives to disappear.

9. Explain in detail about Rectified linear unit (ReLU).

- It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network. (see the below **figure 5.19**)

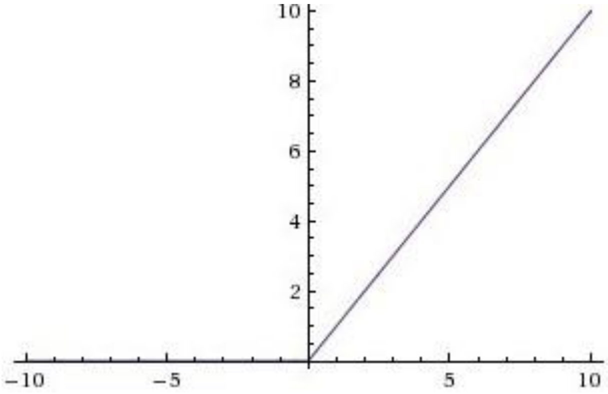


Figure 5.19 RELU FUNCTION

- **Equation :-** ***A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
- **Value Range :-** [0, inf)
- **Nature :-** non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
- **Uses :-** ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

- In simple words, RELU learns *much faster* than sigmoid and Tanh function.
- An activation function for hidden units that has become popular recently with deep networks is the *rectified linear unit* (ReLU), which is defined as

$$\text{ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

Though ReLU is not differentiable at $a = 0$, we use it anyway; we use the left derivative:

$$\text{ReLU}'(a) = \begin{cases} 1 & \text{if } a > 0 \\ 0 & \text{otherwise} \end{cases}$$

Leaky ReLU

In the *leaky ReLU* (the output is also linear on the negative side but with a smaller slope, just enough to make sure that there will be updates for negative activations, albeit small:

$$\text{leaky-ReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha a & \text{otherwise} \end{cases}$$

Advantage:

- ✓ it does not saturate (unlike sigmoid and tanh), updates can still be done for large positive *a* for some inputs, some hidden unit activations will be zero, meaning that we will have a sparse representation
- ✓ Sparse representations lead to faster Training

Disadvantage:

- ✓ The derivative is zero for $a \leq 0$, there is no further training if, for a hidden unit, the weighted sum somehow becomes negative. This implies that one should be careful in initializing the weights so that the initial activation for all hidden units is positive.

10. Explain in detail about Batch Normalization.

Normalization

- Normalization is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.

Batch normalization

- **Batch normalization** is a process to make neural networks faster and more stable through adding extra layers in a deep neural network. The new layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer. A typical neural network is trained using a collected set of input data called **batch**. Similarly, the normalizing process in batch normalization takes place in batches, not as a single input.
- A similar case can also be made for the hidden units, and this is the idea behind *batch normalization.*
- We normalize hidden unit values *before* applying the activation function, such as sigmoid or ReLU. Let us call that weighted sum *aj*. For each batch or minibatch, for each hidden unit *j* we calculate the mean *mj* and standard deviation *sj* of its values, and we first z-normalize:

$$\tilde{a}_j = \frac{a_j - m_j}{s_j}$$

- We can then map these to have arbitrary mean *γj* and scale *βj* and then we apply the activation function.

$$\hat{a}_j = \gamma_j \tilde{a}_j + \beta_j$$

- First, *mj* and *sj* are calculated anew for each batch, and we see immediately that batch normalization is not meaningful with online learning or very small minibatches.
- Second, *γj* and *βj* are parameters that are initialized and updated (after each batch or minibatch) using gradient descent, just like the connection weights. So they require extra memory and computation.
- These new parameters allow each hidden unit to have its arbitrary (but learned) mean and standard deviation for its activation.
- Once learning is done, we can go over the whole training data (or a large enough subset) and calculate *mj* and *sj* to use during testing, when we see one instance at a time.

Why batch normalization?

- **An internal covariate shift** occurs when there is a change in the input distribution to our network. When the input distribution changes, hidden layers try to learn to adapt to the new distribution. This slows down the training process. If a process slows down, it takes a long time to converge to a global minimum.

Example: Suppose we are training an image classification model, that classifies the images into Dog or Not Dog. Let’s say we have the images of white dogs only, these images will have certain distribution as well. Using these images model will update its parameters.

Later, if we get a new set of images, consisting of non-white dogs. These new images will have a slightly different distribution from the previous images. Now the model will change its parameters according to these new images. Hence the distribution of the hidden activation will also change. This change in hidden activation is known as an internal covariate shift. Here batch normalization works.

Advantages of Batch Normalization

- ✓ Speed Up the Training
- ✓ Handles internal covariate shift
- ✓ **The model is less delicate to** hyperparameter tuning**.**
- ✓ Batch normalization smoothens the loss function that in turn by optimizing the model parameters improves the training speed of the model.

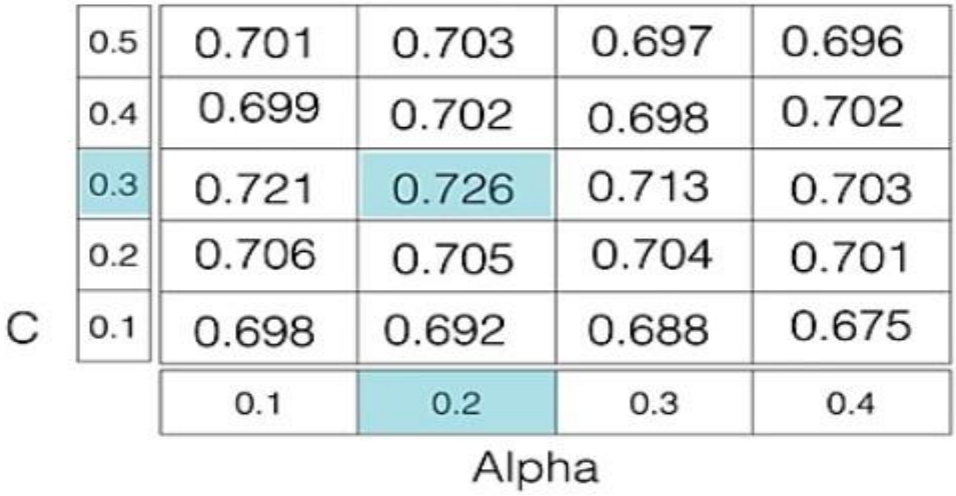
11. Explain in detail about hyperparameter tuning.

- ➢ A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.
- ➢ However, there is another kind of parameter, known as ***Hyperparameters***, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.
- ➢ Some examples of model hyperparameters include:
  1. The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization
  2. The learning rate for training a neural network.
  3. The C and sigma hyperparameters for support vector machines.
  4. The k in k-nearest neighbors.
- ➢ Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. The two best strategies for Hyperparameter tuning are:
  1. GridSearchCV
  2. RandomizedSearchCV

➢ GridSearchCV

- ✓ In GridSearchCV approach, the machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for the best set of hyperparameters from a grid of hyperparameters values.

- ✓ **For example,** if we want to set two hyperparameters C and Alpha of the Logistic Regression Classifier model, with different sets of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.
- ✓ As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination of ***C=0.3 and Alpha=0.2***, the performance score comes out to be **0.726(Highest)**, therefore it is selected. (see in **Figure 5.20)**



	0.5	0.701	0.703	0.697	0.696
	0.4	0.699	0.702	0.698	0.702
	0.3	0.721	0.726	0.713	0.703
	0.2	0.706	0.705	0.704	0.701
C	0.1	0.698	0.692	0.688	0.675
		0.1	0.2	0.3	0.4
		Alpha			

Figure 5.20 GridSearchCV

The following code illustrates how to use GridSearchCV

```
# Necessary imports
from sklearn.linear_model import Logistic Regression
from sklearn.model_selection import GridSearchCV

# Creating the hyperparameter grid
c_space = np.logspace(-5, 8, 15) param_grid = {'C': c_space}

#Instantiating logistic regression classifier
logreg = Logistic Regression()

# Instantiating the GridSearchCV object
logreg_cv= GridSearchCV(logreg, param_grid, cv = 5)
logreg_cv.fit(X,y)

# Print the tuned parameters and score
print("Tuned Logistic Regression Parameters:{}".format(logreg_cv.best_params_))
print("Best score is {}".format(logrcg_cv.best_score_))
```

Output:

Tuned Logistic Regression Parameters: {'C': 3.7275937203149381) Best score is 0.7708333333333334

Drawback:

GridSearch CV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive.

- **RandomizedSearchCV**

  RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in a random fashion to find the best set of hyperparameters. This approach reduces unnecessary computation.

12. Explain in detail about Regularization.

Overfitting:

- ✓ When a model performs well on the training data and does not perform well on the testing data, then the model is said to have high generalization error. In other words, in such a scenario, the model has low bias and high variance and is too complex. This is called overfitting.
- ✓ Overfitting means that the model is a good fit on the train data compared to the data. Overfitting is also a result of the model being too complex

Regularization:

- ✓ Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Regularization helps choose a simple model rather than a complex one.
- ✓ Generalization error is "a measure of how accurately an algorithm can predict outcome values for previously unseen data." Regularization refers to the modifications that can be made to a leaming algorithm that helps to reduce this generalization error and not the training error.

The commonly used regularization techniques are:

1. **Hints**

- *Hints* are properties of the target function that are known to us independent of the training examples

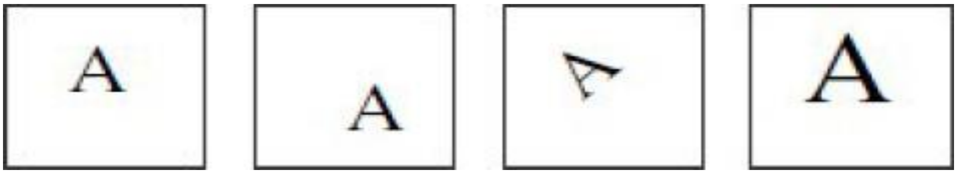


Figure 5.21 HINTS

- The identity of the object does not change when it is translated, rotated, or scaled. Note that this may not always be true, or may be true up to a point: ‘b’ and ‘q’ are rotated versions of each other. These are hints that can be incorporated into the learning process to make learning easier.
- In image recognition, there are invariance hints: The identity of an object does not change when it is rotated, translated, or scaled (see **Figure 5.21).** Hints are auxiliary information that can be used to guide the learning process and are especially useful when the training set is limited.
- There are different ways in which hints can be used:
  - ✓ Hints can be used to create *virtual examples*.
  - ✓ The hint may be incorporated into the network structure.

2. Weight decay:

- Incentivize the network to use smaller weights by adding a penalty to the loss function.
- Even if we start with a weight close to zero, because of some noisy instances, it may move away from zero; the idea in *weight decay* is to add some small constant background force that always pulls a weight toward zero, unless it is absolutely necessary that it be large (in magnitude) to decrease error. For any weight *wi*, the update rule is

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda' w_i$$

3. Ridge Regression

- The Ridge regression technique is used to analyze the model where the variables may be having multicollinearity.
- It reduces the insignificant independent variables though it does not remove them completely. This type of regularization uses the L_2 norm for regularization.

$$E' = E + \frac{\lambda}{2} \sum_i w_i^2$$

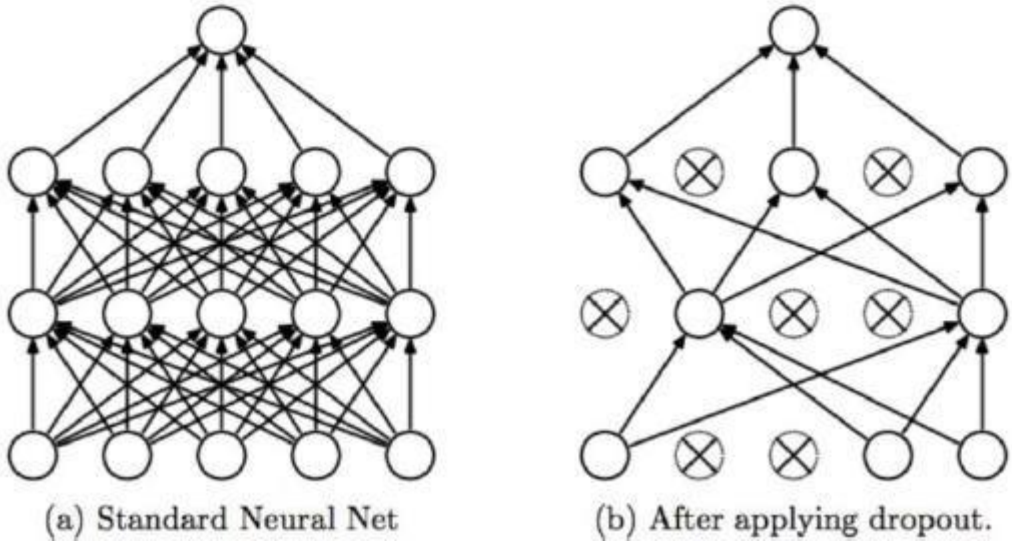
4. Lasso Regression

- Least Absolute Shrinkage and Selection Operator (or LASSO) Regression penalizes the coefficients to the extent that it becomes zero. It eliminates the insignificant independent variables. This regularization technique uses the L1 norm for regularization.

$$E' = E + \frac{\lambda}{2}\sum_i |w_i|$$

5. Dropout

- **"Dropout"** in machine learning refers to the process of randomly ignoring certain nodes in a layer during training.
- In the **Figure 5.22**, the neural network on the left represents a typical neural network where all units are activated. On the right, the red units have been dropped out of the model- the values of their weights and biases are not considered during training.


(a) Standard Neural Net

(b) After applying dropout.

Figure 5.22 Dropout

- Dropout is used as a regularization technique - it prevents overfitting by ensuring that no units are codependent.
- In *dropout*, we have a hyperparameter *p*, and we drop the input or hidden unit with probability *p*, that is, set its output to zero, or keep it with probability 1 – *p*.
- *p* is adjusted using cross-validation; with more inputs or hidden units in a layer, we can afford a larger *p* (see **Figure 5.23**).

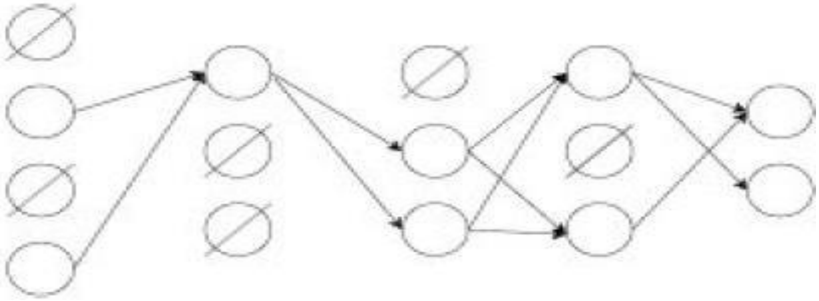


Figure 5.23 In dropout, the output of a random subset of the units are set to zero, and backpropagation is done on the remaining smaller network.

- In each batch or minibatch, for each unit independently we decide randomly to keep it or not. Let us say that *p* = 0.25. So, on average, we remove a quarter of the units and we do backpropagation as usual on the remaining network for that batch or minibatch. We need to make up for the loss of units, though: In each layer, we divide the activation of the remaining units by 1 − *p* to make sure that they provide a vector of similar magnitude to the next layer. There is no dropout during testing.
- In each batch or minibatch, a smaller network (with smaller variance) is trained. Thus dropout is effectively sampling from a pool of possible networks of different depths and widths.
- There is a version called *dropconnect* that drops out or not connections independently, which allows a larger set of possible networks to sample from, and this may be preferable in smaller networks.