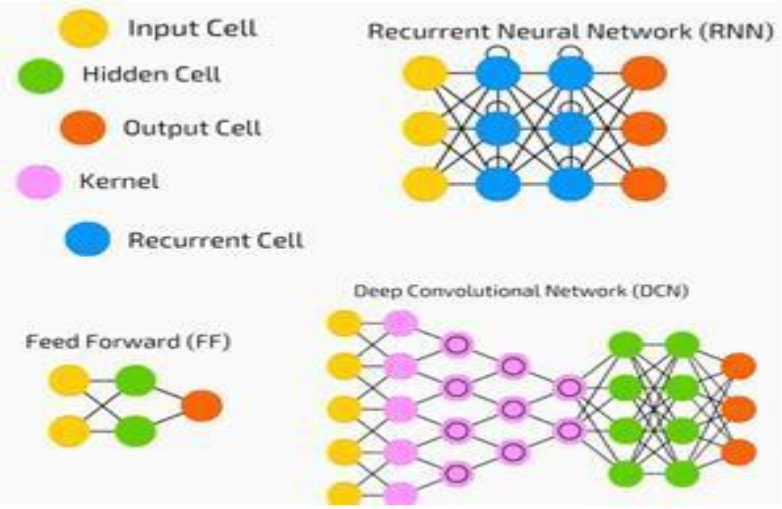
UNIT-II

1. **What is deep neural networks?**

- A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships.

- The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification. We restrict ourselves to feed forward neural networks.

- We have an input, an output, and a flow of sequential data in a deep network. Neural networks are widely used in supervised learning and reinforcement learning problems. These networks are based on a set of layers connected to each other.



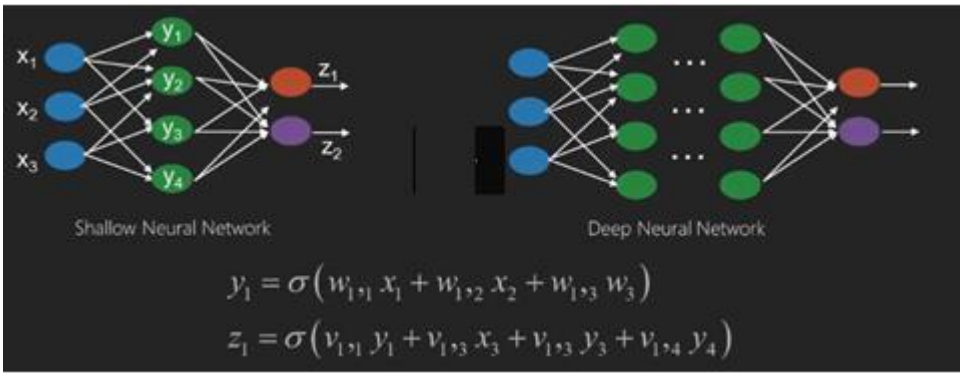
- In deep learning, the number of hidden layers, mostly non-linear, can be large; say about 1000 layers.DL models produce much better results than normal ML networks.

- We mostly use the gradient descent method for optimizing the network and minimising the loss function.

- We can use the Imagenet, a repository of millions of digital images to classify a dataset into categories like cats and dogs. DL nets are increasingly used for dynamic images apart from static ones and for time series and text analysis.

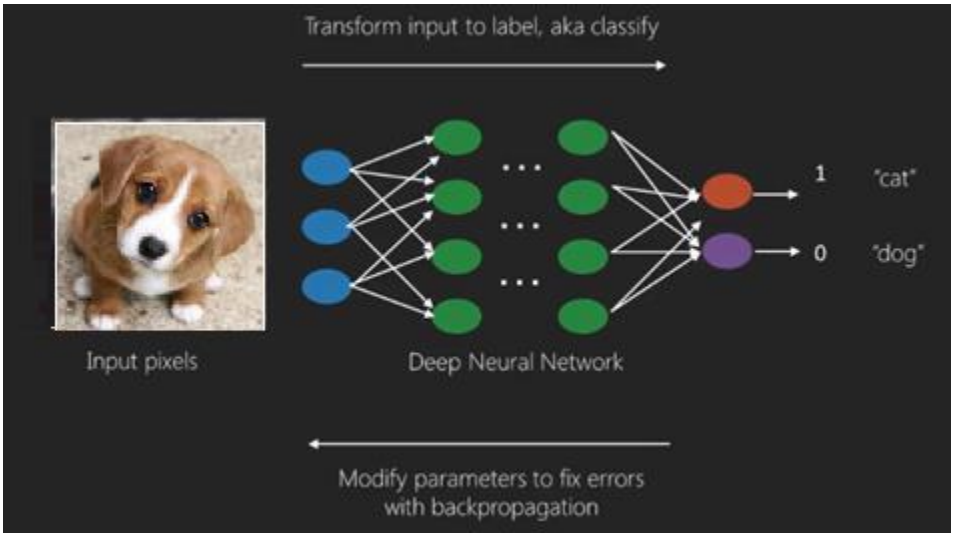
- Training the data sets forms an important part of Deep Learning models. In addition, Backpropagation is the main algorithm in training DL models.DL deals with training large neural networks with complex input output transformations.

- One example of DL is the mapping of a photo to the name of the person(s) in photo as they do on social networks and describing a picture with a phrase is another recent application of DL.



- Neural networks are functions that have inputs like x1,x2,x3…that are transformed to outputs like z1,z2,z3 and so on in two (shallow networks) or several intermediate operations also called layers (deep networks).

- The weights and biases change from layer to layer. ‘w’ and ‘v’ are the weights or synapses of layers of the neural networks. The best use case of deep learning is the supervised learning problem.Here,we have large set of data inputs with a desired set of outputs.



- Here we apply back propagation algorithm to get correct output prediction.The most basic data set of deep learning is the MNIST, a dataset of handwritten digits. We can train deep a Convolutional Neural Network with Keras to classify images of handwritten digits from this dataset.

- We can train deep a Convolutional Neural Network with Keras to classify images of handwritten digits from this dataset.

- The firing or activation of a neural net classifier produces a score. For example,to classify patients as sick and healthy,we consider parameters such as height, weight and body temperature, blood pressure etc.

- A high score means patient is sick and a low score means he is healthy. Each node in output and hidden layers has its own classifiers. The input layer takes inputs and passes on its scores to the next hidden layer for further activation and this goes on till the output is reached.

- This progress from input to output from left to right in the forward direction is called forward propagation.

- Credit assignment path (CAP) in a neural network is the series of transformations starting from the input to the output. CAPs elaborate probable causal connections between the input and the output.

- CAP depth for a given feed forward neural network or the CAP depth is the number of hidden layers plus one as the output layer is included. For recurrent neural networks, where a signal may propagate through a layer several times, the CAP depth can be potentially limitless.

.

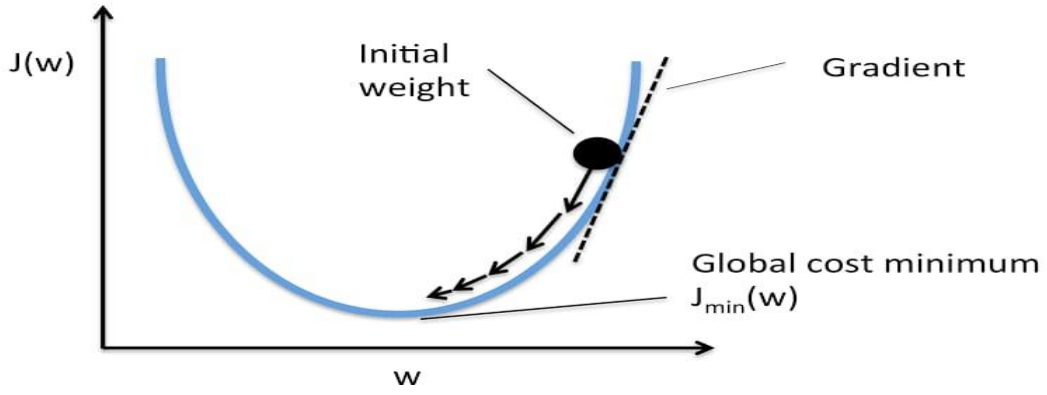
2. What is gradient descent?

- Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a

barometer, gauging its accuracy with each iteration of parameter updates. Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error. Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

- Before we dive into gradient descent, it may help to review some concepts from linear regression. You may recall the following formula for the slope of a line, which is y = mx + b, where m represents the slope and b is the intercept on the y-axis.

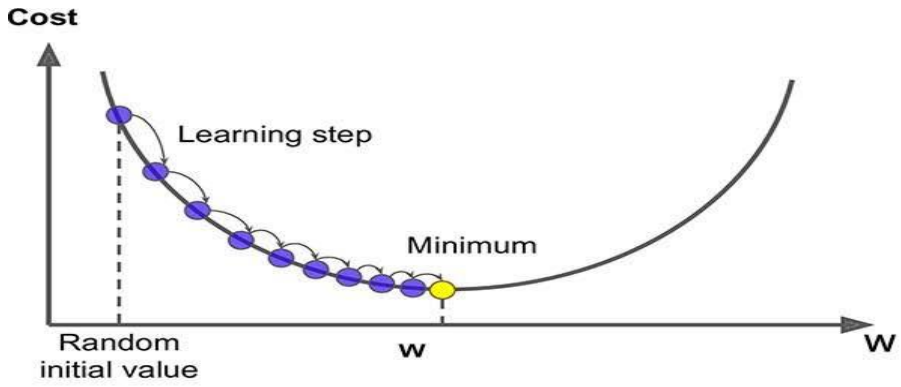
- You may also recall plotting a scatterplot in statistics and finding the line of best fit, which required calculating the error between the actual output and the predicted output (y-hat) using the mean squared error formula. The gradient descent algorithm behaves similarly, but it is based on a convex function.



- The starting point is just an arbitrary point for us to evaluate the performance. From that starting point, we will find the derivative (or slope), and from there, we can use a tangent line to observe the steepness of the slope. The slope will inform the updates to the parameters—i.e. the weights and bias. The slope at the starting point will be steeper, but as new parameters are generated, the steepness should gradually reduce until it reaches the lowest point on the curve, known as the point of convergence.

- Similar to finding the line of best fit in linear regression, the goal of gradient descent is to minimize the cost function, or the error between predicted and actual In order to do this, it requires two data points—a direction and a learning rate. These factors determine the partial derivative calculations of future iterations, allowing it to gradually arrive at the local or global minimum (i.e. point of convergence).

- Learning rate (also referred to as step size or the alpha) is the size of the steps that are taken to reach the minimum. This is typically a small value, and it is evaluated and updated based on the behavior of the cost function. High learning rates result in larger steps but risks overshooting the minimum. Conversely, a low learning rate has small step sizes. While it has the advantage of more precision, the number of iterations compromises overall efficiency as this takes more time and computations to reach the minimum.



- The cost (or loss) function measures the difference, or error, between actual y and predicted y at its current position. This improves the machine learning model's efficacy by providing feedback to the model so that it can adjust the parameters to minimize the error and find the local or global minimum. It continuously iterates, moving along the direction of steepest descent (or the negative gradient) until the cost function is close to or at zero. At this point, the model will stop learning. Additionally, while the terms, cost function and loss function, are considered synonymous, there is a slight difference between them. It's worth noting that a loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

- Types of Gradient Descent There are three types of gradient descent learning algorithms: batch gradient descent, stochastic gradient descent and mini-batch gradient descent.

- Batch gradient descent sums the error for each point in a training set, updating the model only after all training examples have been evaluated. This process referred to as a training epoch.

- While this batching provides computation efficiency, it can still have a long processing time for large training datasets as it still needs to store all of the data into memory. Batch gradient descent also usually produces a stable error gradient and convergence, but sometimes that convergence point isn't the most ideal, finding the local minimum versus the global one.

- Stochastic gradient descent (SGD) runs a training epoch for each example within the dataset and it updates each training example's parameters one at a time. Since you only need to hold one training example, they are easier to store in memory. While these frequent updates can offer more detail and speed, it can result in losses in computational efficiency when compared to batch gradient descent. Its frequent updates can result in noisy gradients, but this can also be helpful in escaping the local minimum and finding the global one.

- Mini-batch gradient descent combines concepts from both batch gradient descent and stochastic gradient descent. It splits the training dataset into small batch sizes and performs updates on each of those batches. This approach strikes a balance between the computational efficiency of batch gradient descent

Challenges with the Gradient Descent

- Although we know Gradient Descent is one of the most popular methods for optimization problems, it still also has some challenges. There are a few challenges as follows:

  1. Local Minima and Saddle Point
  2. Vanishing and Exploding Gradient

3. Differentiation Algorithms

Automatic Differentiation

- Deep learning community has been outside the CS community dealing with automatic differentiation
- The back-propagation algorithm is only one approach to automatic differentiation
- It is a special case of a broader class of techniques called reverse mode accumulation

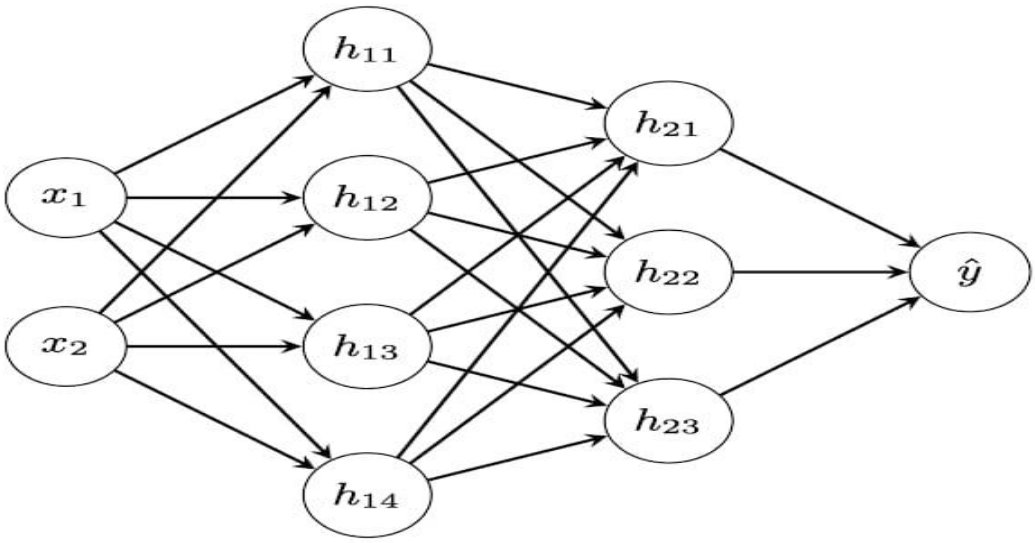
Computational Complexity

- In general, determining the order of evaluation that results in the lowest computational cost is a difficult problem Finding the optimal sequence of operations to compute the gradient is NP-complete (Naumann, 2008) in the sense that it may require simplifying algebraic expressions into their least expensive form

Algebraic substitution

- If pi are probabilities and zi are unnormalized log probabilities. Suppose where we build the softmax function out of exponentiation, summation and division, and construct a cross-entropy loss $J = -\Sigma i\ pi \log qi$. A human mathematician can observe that the derivative of J wrt zi takes a simple form: qi –pi whereas backprop propagates gradients through log and exp operations through the original graph .

- Theano performs some algebraic substitution to improve over graph proposed by pure backpropagation. Future differentiation technology Backprop is not the only- or optimal-way of computing the gradient, but a practical method for deep learning
- In the future, differentiation technology for deep networks may improve with advances in the broader field of automatic differentiation



4. What is Vanishing Gradient ?

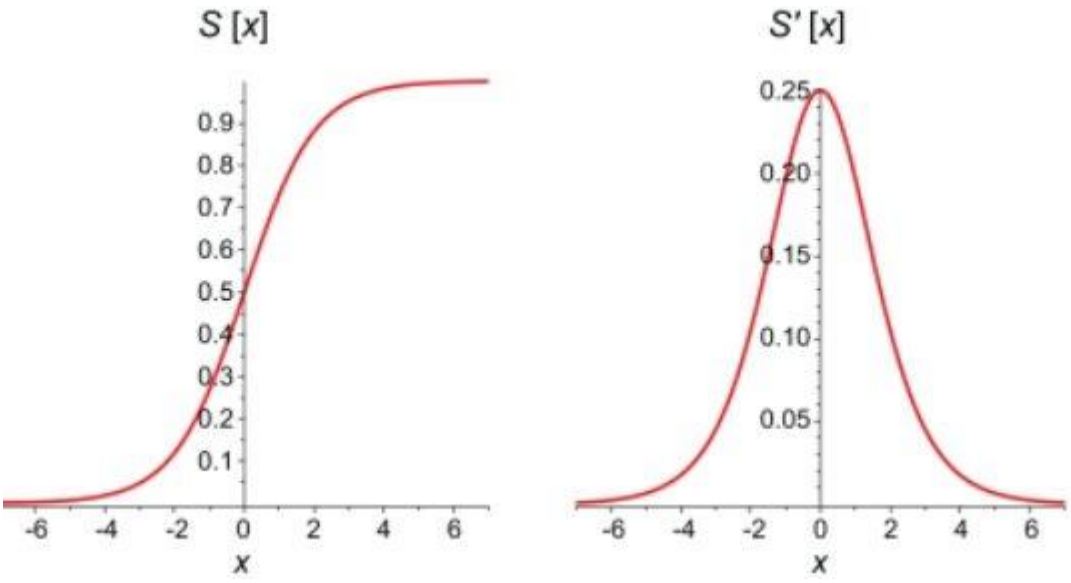
- The sigmoid function is one of the most popular activations functions used for developing deep neural networks. The use of sigmoid function restricted the training of deep neural networks because it caused the vanishing gradient problem. This caused the neural network to learn at a slower pace or in some cases no learning at all. This blog post aims to describe the vanishing gradient problem and explain how use of the sigmoid function resulted in it.

Sigmoid function

- Sigmoid functions are used frequently in neural networks to activate neurons. It is a logarithmic function with a characteristic S shape. The output value of the function is between 0 and 1. The sigmoid function is used for activating the output layers in binary classification problems.

Vanishing Gradient Problem, Explained

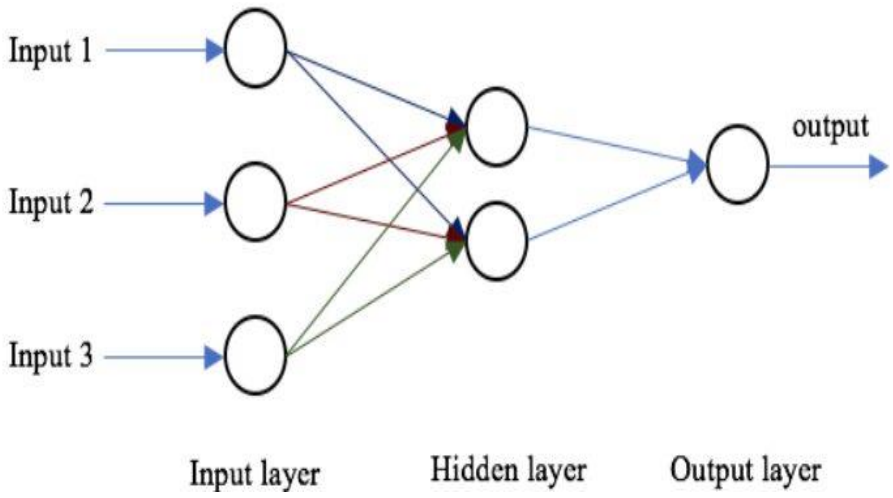
- Our knowledge of how neural networks perform forward and backpropagation is essential to understanding the vanishing gradient problem.



Forward Propagation

- The basic structure of a neural network is an input layer, one or more hidden layers, and a single output layer. The weights of the network are randomly

initialized during forward propagation. The input features are multiplied by the corresponding weights at each node of the hidden layer, and a bias is added to the net sum at each node. This value is then transformed into the output of the node using an activation function. To generate the output of the neural network, the hidden layer output is multiplied by the weights plus bias values, and the total is transformed using another activation function. This will be the predicted value of the neural network for a given input value.



Back Propagation

- As the network generates an output, the loss function(C) indicates how well it predicted the output. The network performs back propagation to minimize the loss. A back propagation method minimizes the loss function by adjusting the weights and biases of the neural network. In this method, the gradient of the loss function is calculated with respect to each weight in the network.

- In back propagation, the new weight(wnew) of a node is calculated using the old weight(wold) and product of the learning rate(η) and gradient of the loss function Vanishing Gradient Problem, Explained.

- With the chain rule of partial derivatives, we can represent gradient of the loss function as a product of gradients of all the activation functions of the nodes with respect to their weights.Therefore, the updated weights of nodes in the network depend on the gradients of the activation functions of each node.

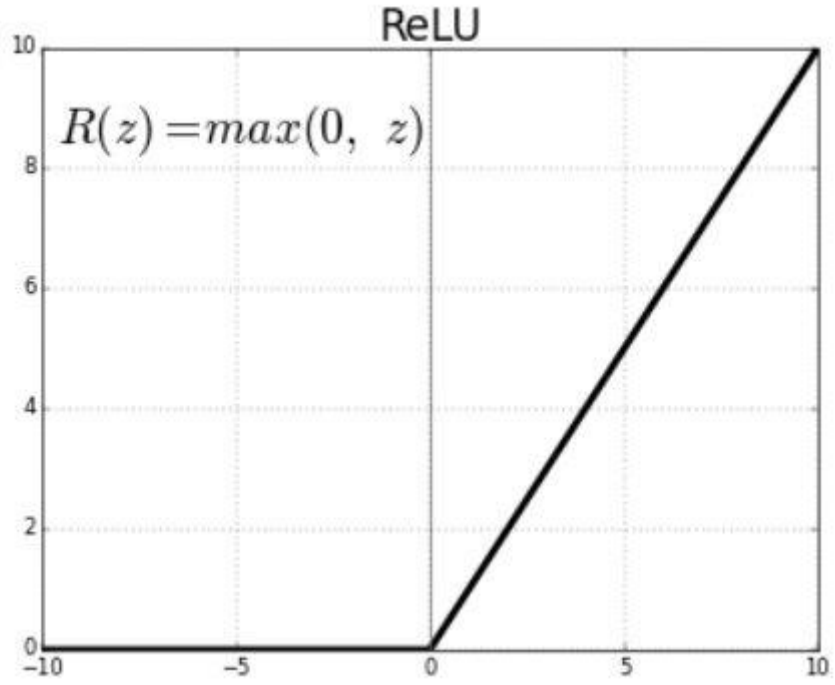
- For the nodes with sigmoid activation functions, we know that the partial derivative of the sigmoid function reaches a maximum value of 0.25. When there are more layers in the network, the value of the product of derivative decreases until at some point the partial derivative of the loss function approaches a value close to zero, and the partial derivative vanishes. We call this the vanishing gradient problem.

- With shallow networks, sigmoid function can be used as the small value of gradient does not become an issue. When it comes to deep networks, the vanishing gradient could have a significant impact on performance. The weights of the network remain unchanged as the derivative vanishes. During back propagation, a neural network learns by updating its weights and biases to reduce the loss function. In a network with vanishing gradient, the weights cannot be updated, so the network cannot learn. The performance of the network will decrease as a result.

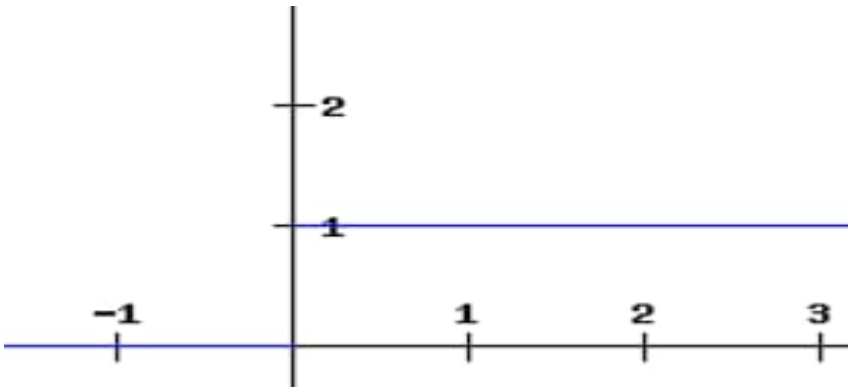
Method to overcome the problem

- The vanishing gradient problem is caused by the derivative of the activation function used to create the neural network. The simplest solution to the problem is to replace the activation function of the network. Instead of sigmoid, use an activation function such as ReLU.

- Rectified Linear Units (ReLU) are activation functions that generate a positive linear output when they are applied to positive input values. If the input is negative, the function will return zero. The derivative of a ReLU function is defined as 1 for inputs that are greater than zero and 0 for inputs that are negative. The graph shared below indicates the derivative of a ReLU function



- If the ReLU function is used for activation in a neural network in place of a sigmoid function, the value of the partial derivative of the loss function will be having values of 0 or 1 which prevents the gradient from vanishing. The use of ReLU function thus prevents the gradient from vanishing. The problem with the use of ReLU is when the gradient has a value of 0. In such cases, the node is considered as a dead node since the old and new values of the weights remain the same. This situation can be avoided by the use of a leaky ReLU function which prevents the gradient from falling to the zero value.
- Another technique to avoid the vanishing gradient problem is weight initialization. This is the process of assigning initial values to the weights in the neural network so that during back propagation, the weights never vanish.



- In conclusion, the vanishing gradient problem arises from the nature of the partial derivative of the activation function used to create the neural network. The problem can bevworse in deep neural networks using Sigmoid activation function. It can be significantly reduced by using activation functions like ReLU and leaky ReLU

5. What is rectified linear activation function?

- In a neural network, the activation function is responsible for transforming the summed weighted input from the node into the activation of the node or output for that input.

- The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

- The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem.

- The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better.

- The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks.

Rectified Linear Activation Function

- In order to use stochastic gradient descent with backpropagation of errors to train deep neural networks, an activation function is needed that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned.

- The function must also provide more sensitivity to the activation sum input and avoid easy saturation.

- The solution is to use the rectified linear activation function, or ReL for short. A node or unit that implements this activation function is referred to as a rectified linear activation unit, or ReLU for short. Often, networks that use the rectifier function for the hidden layers are referred to as rectified networks.

- Adoption of ReLU may easily be considered one of the few milestones in the deep learning revolution, e.g. the techniques that now permit the routine development of very deep neural networks.

- The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.

- We can describe this using a simple if-statement:

  if input > 0:

  return input

  else:

  return 0

- We can describe this function g() mathematically using the max() function over the set of 0.0 and the input z; for example:

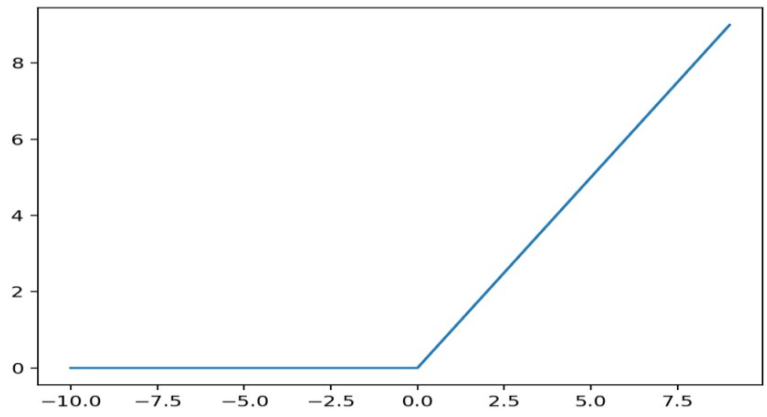
  g(z) = max{0, z}

- The function is linear for values greater than zero, meaning it has a lot of the desirable properties of a linear activation function when training a neural network using backpropagation. Yet, it is a nonlinear function as negative values are

always output as zero, Because the rectified function is linear for half of the input domain and nonlinear for the other half, it is referred to as a piecewise linear function or a hinge function.

- The derivative of the rectified linear function is also easy to calculate. Recall that the derivative of the activation function is required when updating the weights of a node as part of the backpropagation of error.

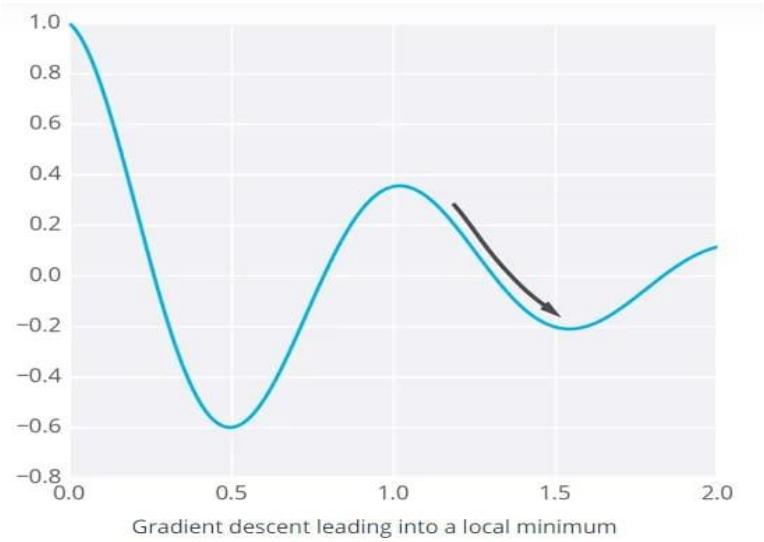
- The derivative of the function is the slope. The slope for negative values is 0.0 and the slope for positive values is 1.0.



Advantages of the Rectified Linear Activation Function

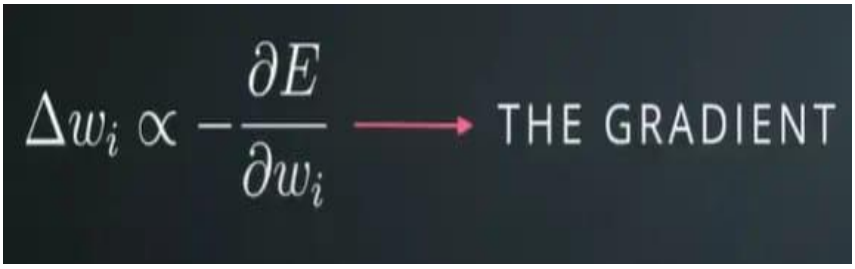
- The rectified linear activation function has rapidly become the default activation function when developing most types of neural networks.

6. Avoid Local Minima in Gradient Descent?



Gradient descent leading into a local minimum

- We've learned how to improve a model by measuring its performance, updating the weights, which are dependent on the change of the error on the change of the weights. If this sounds complicated, let me show you the formulation:

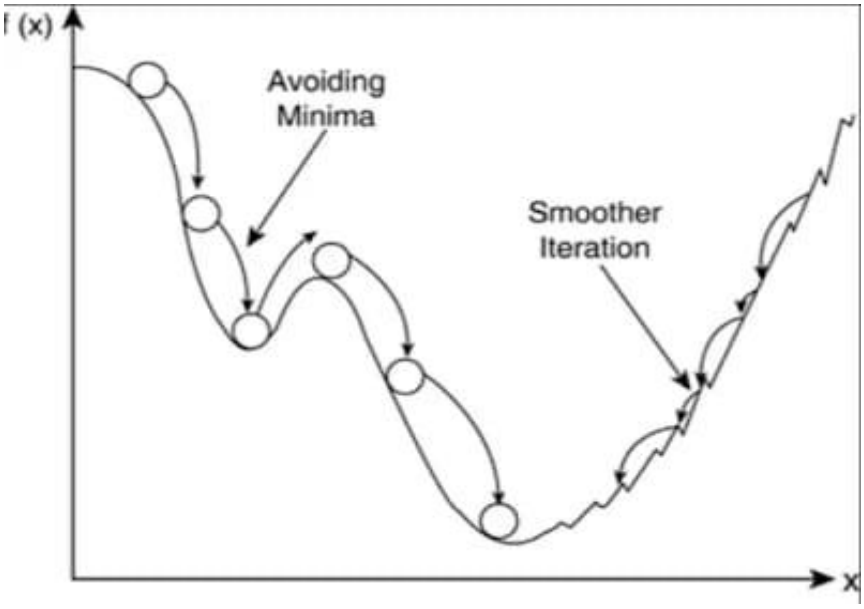


- Above, in order to update the weights, or to have new weights which help a neural network to make better predictions, we need to calculate the gradient, ratio of error

change to the weight change. Cool. But what if we could not reach out to the global minima which will reduce the error the most? We know that we need to use the direction to reach the lowest point of the error, and we can do this using a gradient (by taking the derivative of error).

- So, if we use a method that allows us to reach different directions, we can get rid of the local minima, right? If we use a noisy gradient, a gradient that points in different directions, rather than in one direction, our problem is solved.

- This is called **stochastic gradient descent (SGD) or batch gradient**, by using SGD we avoid the local minima and reach better minima.



7. Heuristics for faster training

- A heuristic is a technique that is used to solve a problem faster than the classic methods. These techniques are used to find the approximate solution of a problem when classical methods do not. Heuristics are said to be the problem-solving techniques that result in practical and quick solutions.

- Heuristics are strategies that are derived from past experience with similar problems. Heuristics use practical methods and shortcuts used to produce the solutions that may or may not be optimal, but those solutions are sufficient in a given limited timeframe.

- Heuristics are used in situations in which there is the requirement of a short-term solution. On facing complex situations with limited resources and time, Heuristics can help the companies to make quick decisions by shortcuts and approximated calculations. Most of the heuristic methods involve mental shortcuts to make decisions on past experiences.

- The heuristic method might not always provide us the finest solution, but it is assured that it helps us find a good solution in a reasonable time.

- Based on context, there can be different heuristic methods that correlate with the problem's scope. The most common heuristic methods are - trial and error, guesswork, the process of elimination, historical data analysis. These methods involve simply available information that is not particular to the problem but is

most appropriate. They can include representative, affect, and availability heuristics Direct Heuristic Search techniques in AI

- It includes Blind Search, Uninformed Search, and Blind control strategy. These search techniques are not always possible as they require much memory and time. These techniques search the complete space for a solution and use the arbitrary ordering of operations.

- The examples of Direct Heuristic search techniques include Breadth-First Search (BFS) and Depth First Search (DFS).

- Weak Heuristic Search techniques in AI. It includes Informed Search, Heuristic Search, and Heuristic control strategy. These techniques are helpful when they are applied properly to the right types of tasks. They usually require domain-specific information.

- The examples of Weak Heuristic search techniques include Best First Search (BFS) and A*. Before describing certain heuristic techniques, let's see some of the techniques listed below:

  Bidirectional Search

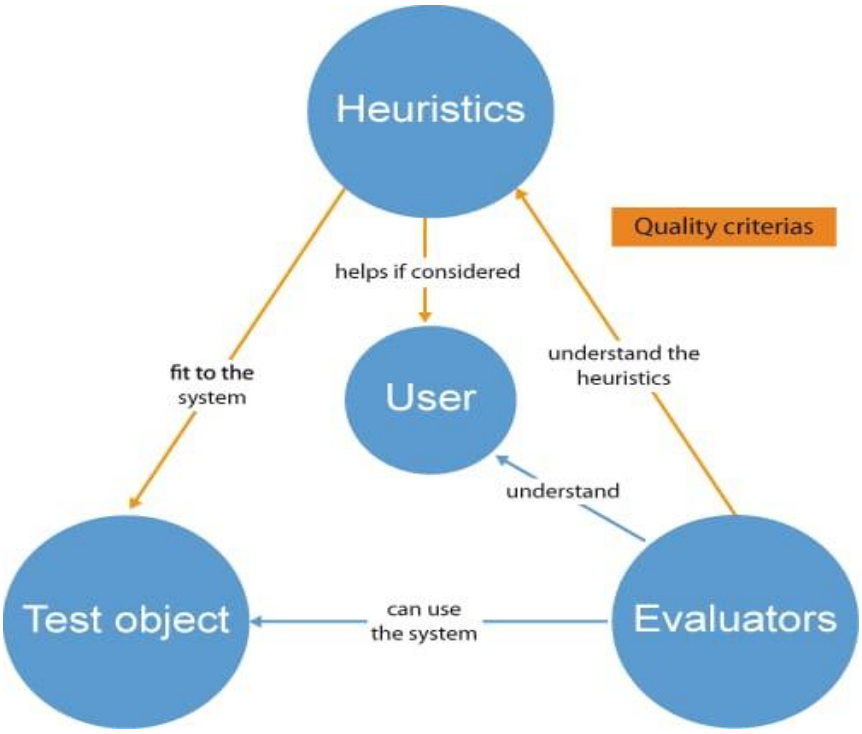
  A* search

  Simulated Annealing

  Hill Climbing

  Best First search

  Beam search



8. Nesterov momentum

- Nesterov Momentum is an extension to the gradient descent optimization algorithm.

- The approach was described by (and named for) Yurii Nesterov in his 1983 paper titled "A Method For Solving The Convex Programming Problem With Convergence Rate O(1/k^2)."

- Ilya Sutskever, et al. are responsible for popularizing the application of Nesterov Momentum in the training of neural networks with stochastic gradient descent described in their 2013 paper "On The Importance Of Initialization And Momentum In Deep Learning." They referred to the approach as "Nesterov's Accelerated Gradient," or NAG for short.

- Nesterov Momentum is just like more traditional momentum except the update is performed using the partial derivative of the projected update rather than the derivative current variable value, Traditional momentum involves maintaining an additional variable that represents the last update performed to the variable, an exponentially decaying moving average of past gradients.
- This last update or last change to the variable is then added to the variable scaled by a "momentum" hyperparameter that controls how much of the last change to add, e.g. 0.9 for 90%.

- It is easier to think about this update in terms of two steps, e.g calculate the change in the variable using the partial derivative then calculate the new value for the variable.

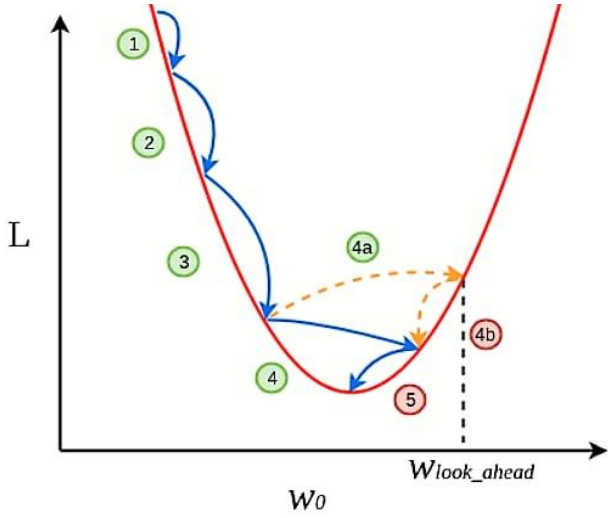
- change(t+1) = (momentum * change(t)) – (step_size * f'(x(t))) x(t+1) = x(t) + change(t+1) We can think of momentum in terms of a ball rolling downhill that will accelerate and continue to go in the same direction even in the presence of small hills.

- A problem with momentum is that acceleration can sometimes cause the search to overshoot the minima at the bottom of a basin or valley floor.

- Nesterov Momentum can be thought of as a modification to momentum to overcome this problem of overshooting the minima.

- It involves first calculating the projected position of the variable using the change from the last iteration and using the derivative of the projected position in the calculation of the new position for the variable.

- Calculating the gradient of the projected position acts like a correction factor for the acceleration that has been accumulated.



Nesterov Momentum is easy to think about this in terms of the four steps:

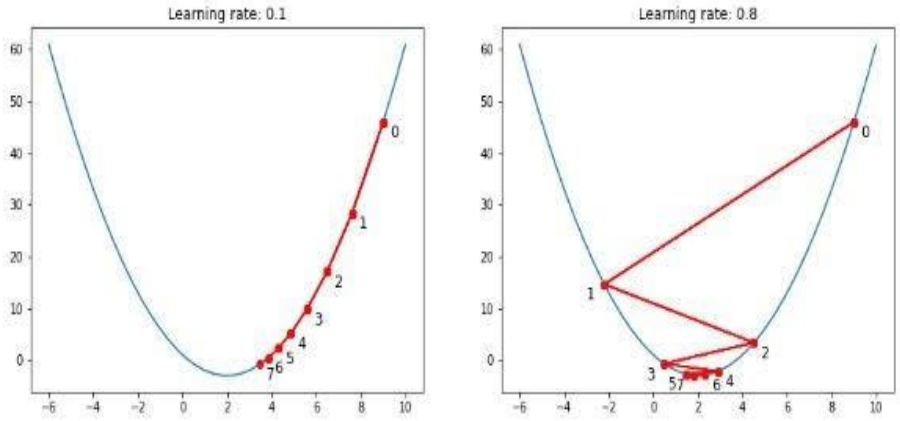
- Project the position of the solution.
- Calculate the gradient of the projection.
- Calculate the change in the variable using the partial derivative.
- Update the variable.
- Let's go through these steps in more detail.

- First, the projected position of the entire solution is calculated using the change calculated in the last iteration of the algorithm.
- projection(t+1) = x(t) + (momentum * change(t))
- We can then calculate the gradient for this new position.
- gradient(t+1) = f'(projection(t+1))
- Now we can calculate the new position of each variable using the gradient of the projection, first by calculating the change in each variable.
- change(t+1) = (momentum * change(t)) – (step_size * gradient(t+1))

- And finally, calculating the new value for each variable using the calculated change.
- x(t+1) = x(t) + change(t+1)

- In the field of convex optimization more generally, Nesterov Momentum is known to improve the rate of convergence of the optimization algorithm e.g. reduce the number of iterations required to find the solution

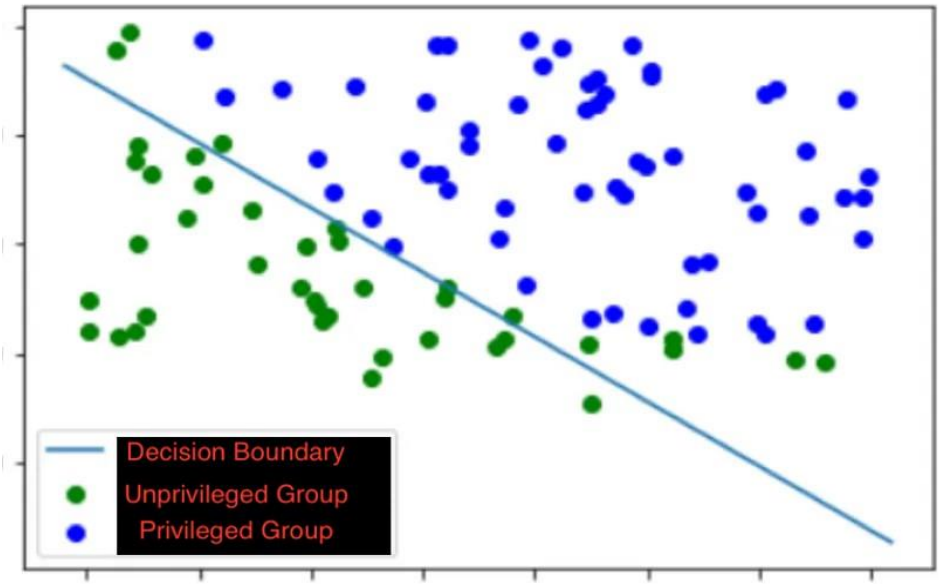
- The simplest solution is to use other activation functions, such as ReLU, which doesn't cause a small derivative. Residual networks are another solution, as they provide residual connections straight to earlier layers.



9. Bias in Machine Learning?

- Bias takes many different forms and impact all groups of people. It can range from implicit to explicit and is often very difficult to detect. In the field of machine learning bias is often subtle and hard to identify, let alone solve. Why is this a problem? Implicit bias in machine learning has very real consequences including denial of a loan, a lengthier prison sentence, and many other harmful outcomes for underprivileged groups.

- The data scientists designing models and the computers running them may not be explicitly biased against a particular group, so how does bias enter the picture? Whether it is along lines of race, gender, religion, sexual orientation, or other forms of identification there are correlations between groups and factors contributing to unfavorable outcomes.



- This is the classic correlation vs. causation issue and has real-world consequences for the groups of people who fall victim to this paradigm. MLFairnessPipeline serves two purposes:

1. Detect bias against underprivileged groups

2. Mitigate bias against underprivileged groups and provide more equitable and fair predictions without sacrificing performance and classification accuracy

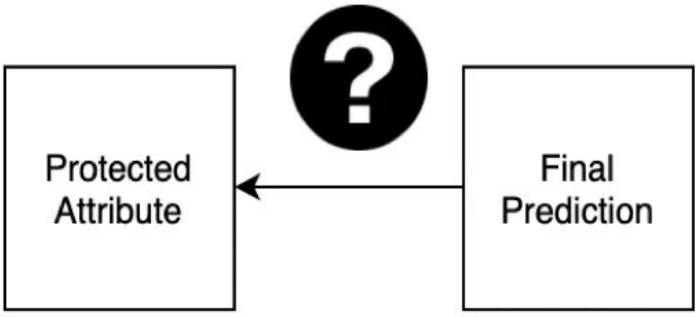
ML Fairness Pipeline is an end-to-end machine learning pipeline with the three following stages:

Pre-processing — Factor re-weighting

In-processing — Adversarial debiasing neural network

- Post-processing — Reject Option Based Classification

- MLFairnessPipeline roots out bias in each of the three stages above. A protected attribute is used to split data into privileged and underprivileged groups. This attribute can be essentially any feature but most common use cases are for race and gender. The pipeline maintains accuracy and performance while at the same time mitigating bias.



- A good example use case of this split would be to design models determining prison sentence length. When determining the length of a prison sentence the individual's likelihood of re-committing a crime is calculated and weighed very heavily.

- Due to systemic racial bias, minorities are often predicted to be more likely to re-commit a crime so if we were to try to mitigate bias in this case we would use "race" as our protected attribute and identify African Americans and Hispanics as the underprivileged group and Caucasians as the privileged group because they often receive more favorable outcomes and preferential treatment.

Pre-processing

- Before the model is trained and after our protected attribute and groups are selected, features are re-weighted in favor of the underprivileged group to give them a boost before training a model even begins.

- In the use case above, Caucasians are about 10–15% more likely to receive a favorable outcome than minorities when being assessed on likelihood of re-committing a crime. After re-weighting, this 10–15% difference in favorable outcomes is reduced to 0.

	Training Data	Test Data
Difference in mean outcome before reweighting	-0.147152	-0.098170
Difference in mean outcome after reweighting	0.00	0.00

In-processing

- After pre-processing, we move on to the in-processing stage where the learning takes place and we build our model. MLFairnessPipeline then builds a neural network using TensorFlow and leverages adversarial debiasing. This actually entails two models: one specified by the user to try to predict a specified outcome from a set of features and a second adversarial model to try and predict the protected attribute based on the outcome of the trained model.

- What this does is ensure a one-way relationship between the protected attribute and the outcome, ensuring that the protected attribute cannot be guessed based on the outcome.

- By breaking the link between the protected attribute and outcome we are ensuring a more equal set of outcomes across both favorable and unfavorable groups.