Q. **Why do we need neural network instead of straight forward traditional computing?**

**Answer**

Neural networks offer a different way to analyze data, and to recognize patterns within that data, than traditional computing methods. However, they are not a solution for all computing problems. Traditional computing methods work well for problems that can be well characterized. Balancing checkbooks, keeping ledgers, and keeping tabs of inventory are well defined and do not require the special characteristics of neural networks.

Traditional computers are ideal for many applications. They can process data, track inventories, network results, and protect equipment. These applications do not need the special characteristics of neural networks.

Expert systems are an extension of traditional computing and are sometimes called the fifth generation of computing. (First generation computing used switches and wires. The second generation occurred because of the development of the transistor. The third generation involved solid-state technology, the use of integrated circuits, and higher level languages like COBOL, Fortran, and "C". End user tools, "code generators," are known as the fourth generation.) The fifth generation involves artificial intelligence.

| **CHARACTERISTICS** | **TRADITIONAL COMPUTING (including Expert Systems)** | **ARTIFICIAL NEURAL NETWORKS** |
| --- | --- | --- |
| Processing style Functions | Sequential Logically (left brained) via Rules Concepts Calculations | Parallel Gestault (right brained) via Images Pictures Controls |
| Learning Method Applications | by rules (didactically) Accounting word processing math inventory digital communications | by example (Socratically) Sensor processing speech recognition pattern recognition text recognition |

Table 2.6.1 Comparison of Computing Approaches.

**Q. What are the different weight initialization techniques you have used?**

Answer

1. Zero initialization
2. Random initialization
3. He initialization
4. Xavier initialization

**Q. Can you visualize a neural network? if yes provide name of software we can use?**

Answer

Yes, using Netron

**Q. How will you explain training of neural network?**

Answer

To build a good Artificial Neural Network (ANN) you will need the following ingredients

Artificial Neurons (processing node) composed of:

* + (many) input neuron(s) connection(s)
  + a computation unit composed of:
    - a linear function (ax+b)
    - an activation function
  + an output

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 what we call 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.

And this is the magic of Neural Network Adaptability: 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.

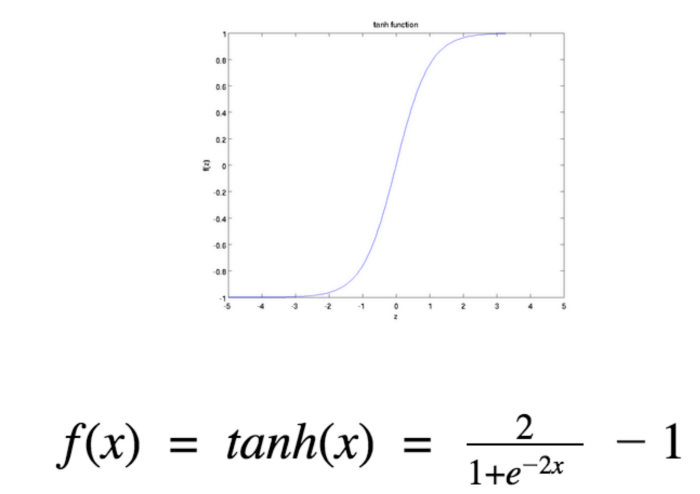
**Q. Can you please explain difference between sigmoid & tanh function.**

Answer

Sigmoid function and tanh function are two activation functions used in deep learning. Also, they look very similar to each other. In this article, I’d like to have a quick comparison.

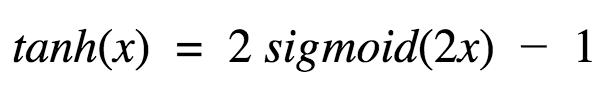
# Sigmoid functionhttps://miro.medium.com/max/700/1*2Vf8SOGNQWM8TqxUWlWSYg.png

# tanh function



# Differences between them

The difference can be seen from the picture below. Sigmoid function has a range of 0 to 1, while tanh function has a range of -1 to 1. “In fact, tanh function is a scaled sigmoid function!”



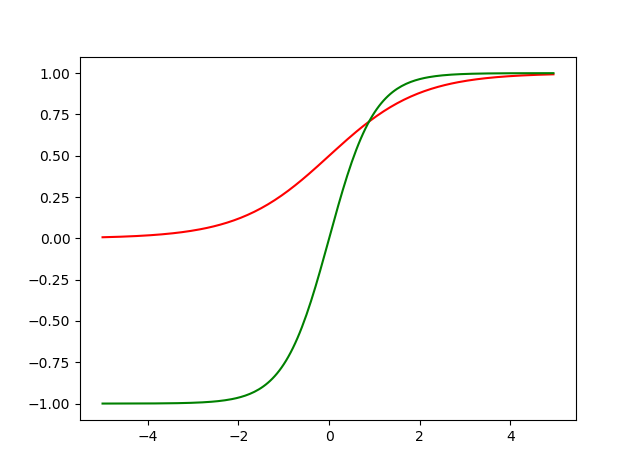
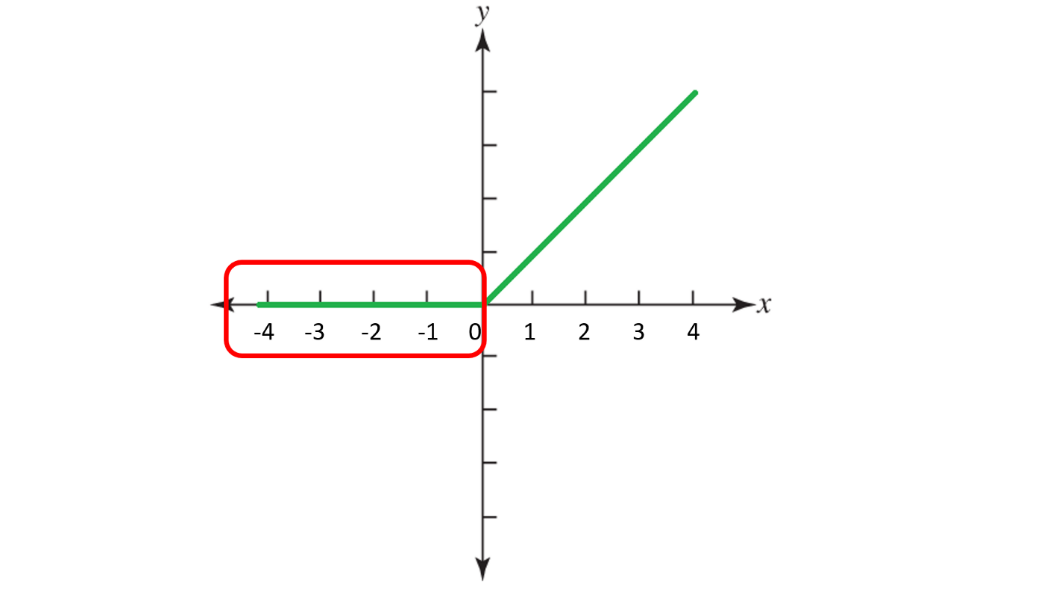


Figure :- The red one is sigmoid and the green one is the tanh function

**100. Explain disadvantage of using RELU function.**

**The dying ReLU Problem**

The dying ReLU problem refers to the scenario when a large number of ReLU neurons only output values of 0. From the red outline below, we can see that this happens when the inputs are in the **negative**range.



Red outline (in the negative x range) demarcating the horizontal segment where ReLU outputs 0

While this characteristic is what gives ReLU its strengths (through network sparsity), it becomes a problem when a majority of the inputs to these ReLU neurons is in the negative range. The worst case scenario is when the entire network dies, meaning that it becomes just a constant function.

When most of these neurons return output zero, the gradients fail to flow during backpropagation and the weights do not get updated. Ultimately a large part of the network becomes inactive and it is unable to learn further.

Because the slope of ReLU in the negativeinput range is also zero, once it becomes dead (i.e. stuck in negative range and giving output 0), it is likely to remain unrecoverable.

However, the dying ReLU problem does not happen all the time, since the optimizer (e.g. stochastic gradient descent) considers multiple input values each time. **As long as NOT all the inputs** push ReLU to the negative segment (i.e. some inputs are in positive range), the neurons can get to stay active, the weights can get updated, and the network can continue learning.

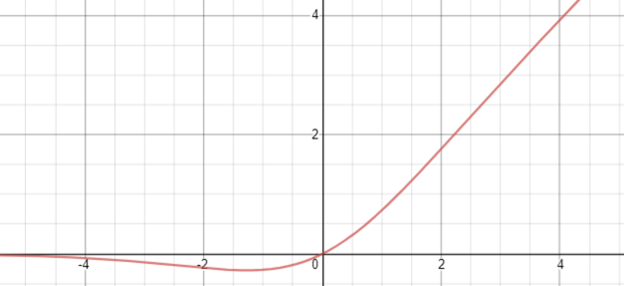
**101. How do you select no. of layers & no. of neurons in neural network?**

These are Hyperparameters so the exact the number is not defined. We take references from different research papers.

**102. Have you ever designed any Neural network architecture by yourself?**

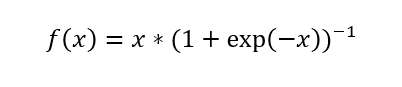
Yes

**103. Can you please explain SWISS Function?**

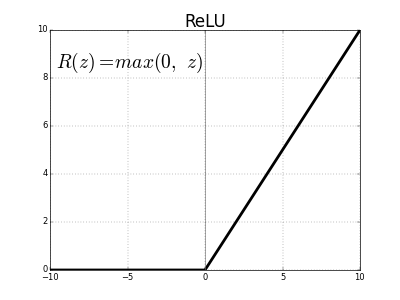


The Swish activation function

Formally stated, the Swish activation function is…

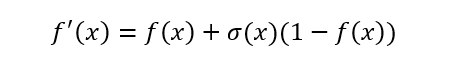


Like ReLU, Swish is bounded below (meaning as *x* approaches negative infinity, *y* approaches some constant value) but unbounded above (meaning as *x* approaches positive infinity, *y* approaches infinity). However, unlike ReLU, Swish is *smooth* (it does not have sudden changes of motion or a vertex):

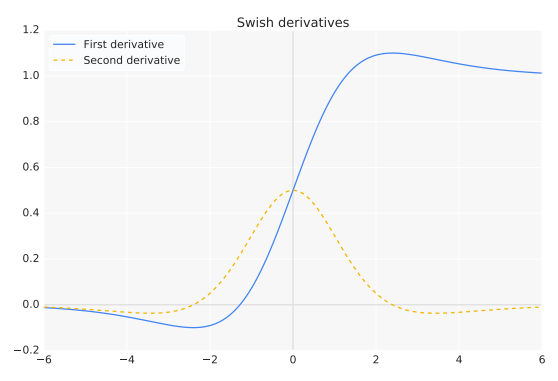


Additionally, Swish is *non-monotonic*, meaning that there is not always a singularly and continually positive (or negative) derivative throughout the entire function. (Restated, the Swish function has a negative derivative at certain points and a positive derivative at other points, instead of only a positive derivative at all points, like Softplus or Sigmoid.

The derivative of the Swish function is…



The first and second derivatives of Swish, plotted:



For inputs less than about 1.25, the derivative has a magnitude of less than 1.

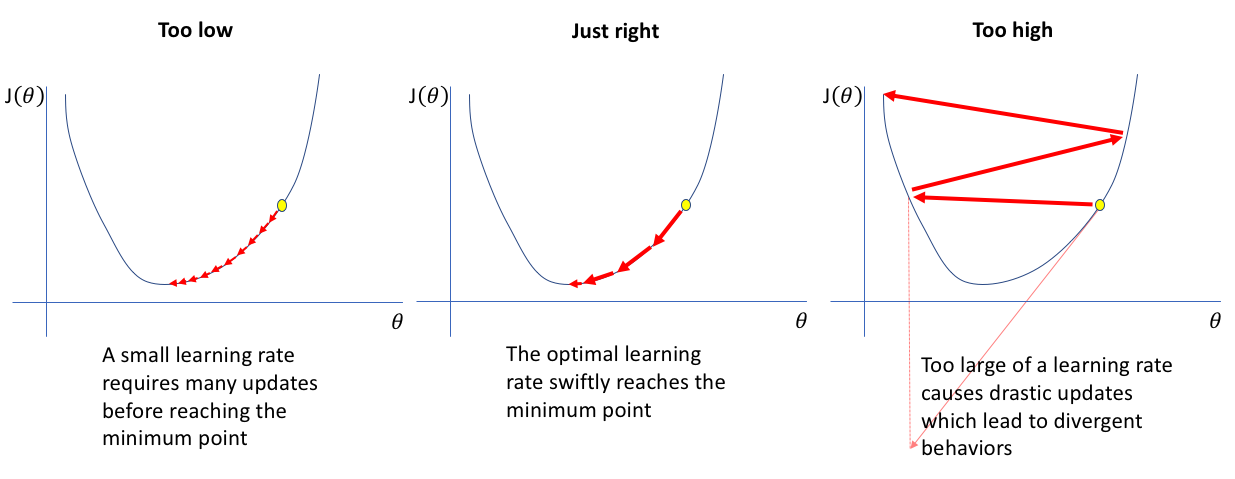
**104. What is learning rate in laymen way and how do you control learning rate?**

**Answer**

The learning rate is **a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function**

One of the key hyperparameters to set in order to train a neural network is the learning rate for gradient descent. As a reminder, this parameter scales the magnitude of our weight updates in order to minimize the network's loss function.

If your learning rate is set too low, training will progress very slowly as you are making very tiny updates to the weights in your network. However, if your learning rate is set too high, it can cause undesirable divergent behavior in your loss function. I'll visualize these cases below - if you find these visuals hard to interpret, I'd recommend reading (at least) the first section in my post on gradient descent.



**105. What is diff between batch, minibatch & stochastic gradient decent.**

**Answer**

**Batch Gradient Descent**

In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that’s just one step of gradient descent in one epoch.

Batch Gradient Descent is great for convex or relatively smooth error manifolds. In this case, we move somewhat directly towards an optimum solution.

# Stochastic Gradient Descent

In Batch Gradient Descent we were considering all the examples for every step of Gradient Descent. But what if our dataset is very huge. Deep learning models crave for data. The more the data the more chances of a model to be good. Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples. This does not seem an efficient way. To tackle this problem we have Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a single step.

SGD can be used for larger datasets. It converges faster when the dataset is large as it causes updates to the parameters more frequently.

**Mini Batch**

Neither we use all the dataset all at once nor we use the single example at a time. We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both the former variants we saw.

So, when we are using the mini-batch gradient descent we are updating our parameters frequently as well as we can use vectorized implementation for faster computations.

**106. What do you understand by batch size while training Neural N/w with example**

Answer

The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.

Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.

A training dataset can be divided into one or more batches.

When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.

* Batch Gradient Descent. Batch Size = Size of Training Set
* Stochastic Gradient Descent. Batch Size = 1
* Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set

In the case of mini-batch gradient descent, popular batch sizes include 32, 64, and 128 samples.

**107. Explain 5 best optimizer you know with mathematical explanation.**

**Answer**

**Stochastic Gradient Descent**

It’s a variant of Gradient Descent. It tries to update the model’s parameters more frequently. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Gradient Descent.

**θ=θ−α⋅∇J(θ;x(i);y(i)) , where {x(i) ,y(i)} are the training examples**.

As the model parameters are frequently updated parameters have high variance and fluctuations in loss functions at different intensities.

**Advantages**:

1. Frequent updates of model parameters hence, converges in less time.
2. Requires less memory as no need to store values of loss functions.
3. May get new minima’s.

**Disadvantages**:

1. High variance in model parameters.
2. May shoot even after achieving global minima.
3. To get the same convergence as gradient descent needs to slowly reduce the value of learning rate.

**Mini-Batch Gradient Descent**

It’s best among all the variations of gradient descent algorithms. It is an improvement on both SGD and standard gradient descent. It updates the model parameters after every batch. So, the dataset is divided into various batches and after every batch, the parameters are updated.

**θ=θ−α⋅∇J(θ; B(i)), where {B(i)} are the batches of training examples**.

**Advantages**:

1. Frequently updates the model parameters and also has less variance.
2. Requires medium amount of memory.

**All types of Gradient Descent have some challenges:**

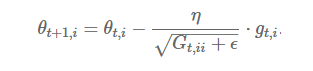
1. Choosing an optimum value of the learning rate. If the learning rate is too small than gradient descent may take ages to converge.
2. Have a constant learning rate for all the parameters. There may be some parameters which we may not want to change at the same rate.
3. May get trapped at local minima.

# Adagrad

One of the disadvantages of all the optimizers explained is that the learning rate is constant for all parameters and for each cycle. This optimizer changes the learning rate. It changes the learning rate **‘η’**for each parameter and at every time step **‘t’.**It’s a type second order optimization algorithm. It works on the derivative of an error function.

https://miro.medium.com/max/213/1*zCRt57Wf8KYkvYmqWFyqew.png

A derivative of loss function for given parameters at a given time t.



Update parameters for given input i and at time/iteration t

**η** is a learning rate which is modified for given parameter **θ(i)**at a given time based on previous gradients calculated for given parameter**θ(i).**

We store the sum of the squares of the gradients w.r.t. **θ(i)** up to time step **t**, while **ϵ** is a smoothing term that avoids division by zero (usually on the order of 1e−8). Interestingly, without the square root operation, the algorithm performs much worse.

It makes big updates for less frequent parameters and a small step for frequent parameters.

**Advantages**:

1. Learning rate changes for each training parameter.
2. Don’t need to manually tune the learning rate.
3. Able to train on sparse data.

**Disadvantages**:

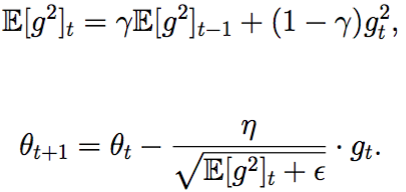
1. Computationally expensive as a need to calculate the second order derivative.
2. The learning rate is always decreasing results in slow training.

# AdaDelta

It is an extension of **AdaGrad** which tends to remove the *decaying learning Rate* problem of it. Instead of accumulating all previously squared gradients, ***Adadelta*** limits the window of accumulated past gradients to some fixed size **w**. In this exponentially moving average is used rather than the sum of all the gradients.

**E[g²](t)=γ.E[g²](t−1)+(1−γ).g²(t)**

We set **γ** to a similar value as the momentum term, around 0.9.



Update the parameters

**Advantages**:

1. Now the learning rate does not decay and the training does not stop.

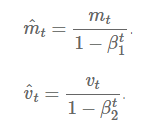
**Disadvantages**:

1. Computationally expensive.

# Adam

[Adam](https://arxiv.org/pdf/1412.6980.pdf) (Adaptive Moment Estimation) works with momentums of first and second order. The intuition behind the Adam is that we don’t want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients like **AdaDelta**, ***Adam***also keeps an exponentially decaying average of past gradients **M(t).**

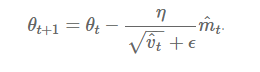
**M(t) and V(t)** are values of the first moment which is the ***Mean*** and the second moment which is the ***uncentered variance*** of the gradientsrespectively.



First and second order of momentum

Here, we are taking mean of **M(t)** and **V(t)** so that **E[m(t)]** can be equal to **E[g(t)]** where, **E[f(x)]** is an expected value of **f(x)**.

To update the parameter:



Update the parameters

The values for β1 is 0.9 , 0.999 for β2, and (10 x exp(-8)) for ‘**ϵ’**.

**Advantages**:

1. The method is too fast and converges rapidly.
2. Rectifies vanishing learning rate, high variance.

**Disadvantages**:

Computationally costly.

**108. Can you build Neural network without using any library? If yes, prove it.**

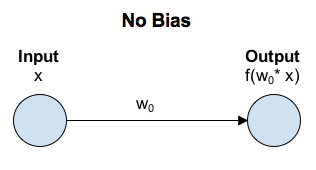
Answer

Notebook Link: - <https://colab.research.google.com/drive/1iNt4eKU6PjDG_ygv-_FhcuBhumeI-v3B>

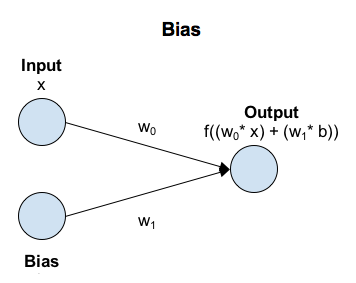
**109. What is use of biases in neural network?**

Answer

The activation function in Neural Networks takes an input 'x' multiplied by a weight 'w'. Bias allows you to shift the activation function by adding a constant (i.e. the given bias) to the input. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value.



In a scenario with no bias, the input to the activation function is 'x' multiplied by the connection weight 'w0'.



In a scenario with bias, the input to the activation function is 'x' times the connection weight 'w0' plus the bias times the connection weight for the bias 'w1'. This has the effect of shifting the activation function by a constant amount (b \* w1).

**110. How do you do hyper-parameter tuning for neural network**

**Answer**

1. Step 1 — Deciding on the network topology
2. Step 2 — Adjusting the learning rate. ...
3. Step 3 — Choosing an optimizer and a loss function. ...
4. Step 4 — Deciding on the batch size and number of epochs. ...
5. Step 5 — Random restarts.
6. Step 6 – Define the Input Shape
7. Step 7- Choose the Right activation function
8. Step 8 – Choosing the no of kernels and layers in CNN

**111. What kind of regularization you used wrt neural network.**

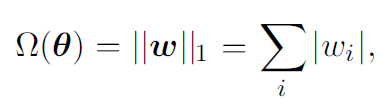
## **L2 Parameter Regularization**

This regularization is popularly known as weight decay. This strategy drives the weights closer to the origin by adding the regularization term omega which is defined as:

https://miro.medium.com/max/235/0*sG75kkuPx3QM5AI0

This technique is also known as ridge regression or Tikhonov regularization.

## **L1 Regularization**

Here the regularization term is defined as:

# Dataset Augmentation

The best and easiest way to make a model generalize is to train it on a large amount of data but mostly we are provided with limited data. One way is to create fake data and add it to our training dataset, for some domains this is fairly straightforward and easy.

# Noise Robustness

Noise is often introduced to the inputs as a dataset augmentation strategy. the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights. Noise injection is much more powerful than simply shrinking the parameters, especially when the noise is added to the hidden units.

# Early Stopping of Training

When training a large model on a sufficiently large dataset, if the training is done for a long amount of time rather than increasing the generalization capability of the model, it increases the overfitting. As in the training process, the training error keeps on reducing but after a certain point, the validation error starts to increase hence signifying that our model has started to overfit.

# Dropout

Dropout is a computationally inexpensive but powerful regularization method, dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks. The method of bagging cannot be directly applied to large neural networks as it involves training multiple models, and evaluating multiple models on each test example. since training and evaluating such networks is costly in terms of runtime and memory, this method is impractical for neural networks.

# Bagging

Bagging or bootstrap aggregating is a technique for reducing generalization error by combining several models. The idea is to train several different models separately, then have all of the models vote on the output for test examples. This is an example of a general strategy in machine learning called model averaging. Techniques employing this strategy are knownas ensemble methods. This is an efficient method as different models don’t make the same types of errors.

**112. What are the libraries you have used for neural network implementation?**

Answer

Keras Tuner, Optuna, HyperOPT, Tune

**113. What do you understand by custom layer and a custom model?**

Answer

Reference Link :- https://keras.io/guides/making\_new\_layers\_and\_models\_via\_subclassing

**114. How do you implement differentiation using TensorFlow or Pytorch library?**

Answer

Tensorflow :- <https://jonathan-hui.medium.com/tensorflow-automatic-differentiation-autodiff-1a70763285cb>

Notebook Link :- https://colab.research.google.com/github/tensorflow/tensorflow/blob/r1.9/tensorflow/contrib/eager/python/examples/notebooks/automatic\_differentiation.ipynb

**Pytorch**

Using autograd to Find and Solve a Derivative

First, it should be obvious that we have to represent our original function in Python as such:

y = 5\*x\*\*4 + 3\*x\*\*3 + 7\*x\*\*2 + 9\*x - 5

**import** **torch**

x = torch.autograd.Variable(torch.Tensor([2]),requires\_grad=True)

y = 5\*x\*\*4 + 3\*x\*\*3 + 7\*x\*\*2 + 9\*x - 5

y.backward()

x.grad

Line by line, the above code:

* imports the torch library
* defines the function we want to compute the derivative of
* defines the value (2) we want to compute the derivative with regard to as a PyTorch Variable object and specifies that it should be instantiated in such a way that it tracks where in the computation graph it connects to in order to perform differentiation by the chain rule (requires\_grad)
* uses autograd's [backward()](https://pytorch.org/docs/master/autograd.html#torch.autograd.backward) to compute the sum of gradients, using the chain rule
* outputs the value stored in the *x* tensor's grad attribute, which, as shown below

tensor([ 233.])

This value, 233, matches what we calculated by hand, above.

**115. What is meaning of epoch in simple terms?**

**Answer**

An epoch is **a term used in machine learning** and indicates the number of passes of the entire training dataset the machine learning algorithm has completed. Datasets are usually grouped into batches (especially when the amount of data is very large).

The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm.

You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.

**116. What do you understand by a TensorFlow record?**

Answer

The TFRecord format is **a simple format for storing a sequence of binary records**. Protocol buffers are a cross-platform, cross-language library for efficient serialization of structured data. Protocol messages are defined by . proto files, these are often the easiest way to understand a message type.

More Depth :- https://www.tensorflow.org/tutorials/load\_data/tfrecord

**117. Explain the technique for doing data augmentation in deep learning**

**Answer**

**Data augmentation** is the technique of increasing the size of data used for training a model. For reliable predictions, the deep learning models often require a lot of training data, which is not always available. Therefore, the existing data is augmented in order to make a better generalized model.

Although data augmentation can be applied in various domains, it's commonly used in computer vision. Some of the most common data augmentation techniques used for images are:

* **Position augmentation**
  + **Scaling**
  + **Cropping**
  + **Flipping**
  + **Padding**
  + **Rotation**
  + **Translation**
  + **Affine transformation**
* **Color augmentation**
  + **Brightness**
  + **Contrast**
  + **Saturation**
  + **Hue**

**118. List down diff CNN network you heard of.**

### LeNet

### AlexNet

### ZFNet

### Inception (GoogLeNet)

### VGG

### ResNet (MSRA)

### ResNeXt

### SENet

### PNASNet

1. EfficientNet
2. DenseNet

**119. List down a names of object detection algorithm you know**

1. RCNN
2. Fasster RCNN
3. Faster RCnn
4. Yolo
5. SSD
6. CenterNet

**120. What is difference between object detection and classification?**

Consider the below image:



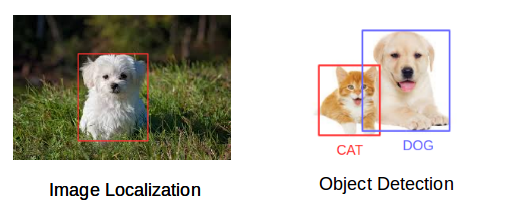
You will have instantly recognized it. It’s a dog.

There’s only one object here: a dog. We can easily use image classification model and predict that there’s a dog in the given image. But what if we have both a cat and a dog in a single image?



We can train a multi-label classifier, in that instance. Now, there’s another caveat - we won’t know the location of either animal/object in the image.

**Image Localization**  helps us to identify the location of a single object in the given image. In case we have multiple objects present, we then rely on the concept of [**Object Detection**](https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/). We can predict the location along with the class for each object using OD.



**121. List down major tasks we perform in CNN.**

1. Image Classification

2. Object Detection

3. Image Segmentaion

4. Image Captioning

5. Visual Question Answering

6. Image Generation

**122. List down algorithms for segmentation**

1. Region-Based Segmentation
   1. Threshold Segmentation
   2. Regional Growth Segmentation
2. Edge Detection Segmentation
   1. Sobel Operator
   2. Laplacian Operator
3. Segmentation based on Clustering

Reference: - https://arxiv.org/ftp/arxiv/papers/1707/1707.02051.pdf