1. B) As number of hidden layers increase, model capacity increases
2. C) It normalizes (changes) all the input before sending it to the next layer
3. A) Network will not converge
4. D) All of these
5. C)
6. B) Simulate the network on a test dataset after every epoch of training. Stop training when the generalization error starts to increase
7. B) Stochastic Gradient Descent
8. A) Freeze all the layers except the last, re-train the last layer
9. B) Training is too slow & C) Restrict activations to become too high or low
10. B) sigmoid & C) softmax

11  If we have a neural network without the activation functions, in that case, every neuron will only be performing a linear transformation on the inputs using the weights and biases. Although linear transformations make the neural network simpler, but this network would be less powerful and will not be able to learn the complex patterns from the data. Thus we use a non linear transformation to the inputs of the neuron and this non-linearity in the network is introduced by an activation function. Thus we use a non linear transformation to the inputs of the neuron and this non-linearity in the network is introduced by an activation function.

12. In neural networks, we forward propagate to get the output and compare it with the real value to get the error.

Now, to minimize the error, we propagate backwards by finding the derivative of error with respect to each weight and then subtracting this value from the weight value.

The basic learning that has to be done in neural networks is training neurons when to get activated. Each neuron should activate only for particular type of inputs and not all inputs. Therefore, by propagating forward we see how well our neural network is behaving and find the error. After we find out that your network has error, we back propagate and use a form of gradient descent to update new values of weights. Then, we will again forward propagate to see how well those weights are performing and then will backward propagate to update the weights. This will go on until we reach some minima for error value.

13. **Batch Gradient Descent** -It is the first basic type of gradient descent in which we use the complete dataset available to compute the gradient of cost function. As we need to calculate the gradient on the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that don’t fit in memory.

**Stochastic Gradient Descent** turns out to be a  slower algorithm. So, for faster computation, we prefer to use stochastic gradient descent. The first step of algorithm is to randomize the whole training set. Then, for updation of every parameter we use only one training example in every iteration to compute the gradient of cost function. As it uses one training example in every iteration this algo is faster for larger data set. In SGD, one might not achieve accuracy, but the computation of results are faster.

**Mini batch algorithm** is the most favorable and widely used algorithm that makes precise and faster results using a batch of ‘ m ‘  training examples. In mini batch algorithm rather than using  the complete data set, in every iteration we use a set of ‘m’ training examples called batch to compute the gradient of the cost function. Common mini-batch sizes range between 50 and 256, but can vary for different applications.

14. This is a mixture of both stochastic and batch gradient descent, thereby has several benefits, some of them are-

1. Easily fits in the memory

2.It is computationally efficient

3.Benefit from vectorization

4.If stuck in local minimums, some noisy steps can lead the way out of them

5.Average of the training samples produces stable error gradients and convergence

**15**. Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

There are two common approaches to use transfer learning they are discussed as follows:

1.Develop Model Approach

2.Pre-trained Model Approach

**Develop Model Approach**

1.Select Source Task. You must select a related predictive modeling problem with an abundance of data where there is some relationship in the input data, output data, and/or concepts learned during the mapping from input to output data.

2.Develop Source Model. Next, you must develop a skillful model for this first task. The model must be better than a naive model to ensure that some feature learning has been performed.

3.Reuse Model. The model fit on the source task can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.

4.Tune Model. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

Pre-trained Model Approach

1.Select Source Model. A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.

2.Reuse Model. The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.

3.Tune Model. Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

The second kind of approach is used mostly.

.