**Deep Learning Woksheet-3**

(Solutions)

**ANS 1)** C) As learning rate increases, model capacity increases.

**ANS 2)** C) It normalizes (changes) all the input before sending it to the next layer.

**ANS 3)** A) Network will not converge

**ANS 4)** D) All of these

**ANS 5)** C) (-4, -4, 3)

**ANS 6)** B) Simulate the network on a test dataset after every epoch of training. Stop training when the generalization error starts to increase.

**ANS 7)** B) Stochastic Gradient Descent

**ANS 8)** A) Freeze all the layers except the last, re-train the last layer.

**ANS 9)**

B) Training is too slow

C) Restrict activations to become too high or low

**ANS 10)** B) sigmoid

**ANS 11)**

Activation functions are really important for an Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable. They introduce non-linear properties. Their main purpose is to convert an input signal of a node in an A-NN to an output signal.

If we do not apply an Activation function then the output signal would simply be a simple linear function. A linear function is just a polynomial of one degree. A linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data. A Neural Network without Activation function would simply be a Linear-regression Model, which has limited power and does not perform good most of the times.

**ANS 12)**

**Forward pass**

Propagating the computations of all neurons within all layers moving from left to right. This starts with the feeding of your feature vector(s)/tensors into the input layer, and ends with the final prediction generated by the output layer. Forward pass computations occur during training in order to evaluate the objective/loss function under the current network parameter settings in each iteration, as well as during inference (prediction after training) when applied to new/unseen data.

**Backward pass**

Known as *back-propagation*, or “backprop”, this is a step executed during training in order to compute the objective/loss function gradient with respect to the network’s parameters for updating them during a single iteration of some form of gradient descent (Adam, RMSProp, etc.). It is named as such because, when viewing a neural network as a computation graph, it starts by computing objective/loss functionderivatives at the output layer, and propagates them back towards the input layer (effectively, this is the [chain rule](https://en.m.wikipedia.org/wiki/Chain_rule) from Calculus in action) in order to compute derivatives for, and make updates to, all parameters in all layers.

**ANS 13)**

* **Batch Gradient Descent:** This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.
* **Stochastic Gradient Descent:** This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.
* **Mini Batch gradient descent:** This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here *b* examples where, *b<m* are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

**ANS 14)**

* High throughput: With mini-batch one can process a large number of input examples per second. The mini batching style of gradient descent is perhaps the only way to use the large number of cores at once in a GPU.
* (Sometimes) faster convergence: The high through-put may also translate to faster convergence depending on the variance in the dataset and the learning rate used.
* High quality gradient: Mini batching allows for a high quality gradient and this will be really useful allowing one to use high learning rates.

**ANS 15)**

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.