**HW to Chapters 6 “Deep Neural Networks” and 7 “Activation Functions”**

**Non-programming Assignment:**

1. **Why are multilayer (deep) neural networks needed?**

Deep Neural Networks (DNNs) are necessary for learning complex representations of data. Each layer extracts increasingly abstract features from the input, allowing the network to understand intricate patterns and relationships within the data. A DNN, with multiple hidden layers, can model these complex relationships more efficiently than a shallow network. Without multiple layers, a network would need to be exponentially large to achieve the same learning capacity​.

1. **What is the structure of weight matrix (how many rows and columns)?**

In DNNs, for a given layer [s], the weight matrix W[s] is structured as follows:

Rows: The number of rows is equal to the number of neurons in the receiving layer, denoted as Ns.

Columns: The number of columns corresponds to the number of neurons in the sending layer, denoted as Ns−1​.

So, W[s] is of size Ns×Ns−1​​.

1. **Describe the gradient descent method.**

Gradient descent is an optimization algorithm used to minimize the cost function during neural network training. It involves the following steps:

Initialization: Initialize the weights and biases.

Forward Propagation: Calculate predictions by passing inputs through the network.

Backward Propagation: Calculate gradients (partial derivatives) of the cost function with respect to the weights and biases by propagating the error backward through the network.

Update Parameters: Adjust the weights and biases by moving in the direction opposite to the gradients using a learning rate. This process is repeated iteratively over the dataset until the cost function converges to a minimum.

1. **Describe in detail forward propagation and backpropagation for deep neural networks.**

**Forward Propagation**:

Pass the input X through each layer.

Calculate the weighted sum for each layer: Z[s]=W[s]A[s−1]+b[s], where A[s−1] is the activation from the previous layer.

Apply the activation function f[s] to Z[s]: A[s]=f[s](Z[s]). Continue this process until the output layer.

For a classification task, typical activation functions for hidden layers are ReLU or linear, while the output layer uses sigmoid, tanh, or softmax​.

**Backpropagation**:

Compute the error at the output layer as the difference between the predicted and actual values: δZ[L]=A−Y.

Compute the gradient of the cost function w.r.t the weights and biases for each layer:

δW[L]=(1/M)δZ[L]A[L−1]T

δb[L]=(1/M)δZ[L]EM

Propagate the error backward through each layer using the chain rule.

Update weights and biases based on the calculated gradients.

1. **Describe linear, ReLU, sigmoid, tanh, and softmax activation functions and explain for what purposes and where they are typically used.**

**Linear Activation Function**:

f(z)=z, which returns the input as the output.

Simple derivative, used mostly for hidden layers when a linear relationship is needed or for regression problems without bounded output​.

**ReLU (Rectified Linear Unit)**:

f(z)=max(0,z)

Used for hidden layers due to computational efficiency and capability to handle the vanishing gradient problem. However, it can lead to “dead neurons” when gradients are zero for negative inputs.

Suitable for MLPs and CNNs​.

**Sigmoid Activation Function**:

f(z)=1+e−z1​

Outputs values in the range (0, 1), making it useful for binary classification problems in the output layer. It suffers from the vanishing gradient problem and is less commonly used in hidden layers​.

**Tanh (Hyperbolic Tangent) Activation Function**:

f(z)=ez+e−zez−e−z​​

Outputs values in the range (-1, 1), offering stronger gradients for low input values compared to sigmoid.

Often used in hidden layers when stronger derivatives around zero are needed for gradient descent optimization​.

**Softmax Activation Function**:

f(zi​)=∑j​ezj​ezi​​​​, where zzz is the input vector.

Converts the raw output scores into probabilities that sum to 1, making it suitable for multi-class classification in the output layer​.