**Non-Programming Assignment**

**1. Hadamard Matrix Product:**

The **Hadamard product**, also known as the **element-wise product**, is a binary operation that takes two matrices of the same dimensions and produces another matrix where each element is the product of the corresponding elements of the input matrices.

Mathematically, for two matrices AAA and BBB of the same size, the Hadamard product CCC is:

Cij=Aij⋅BijC\_{ij} = A\_{ij} \cdot B\_{ij}Cij​=Aij​⋅Bij​

In contrast to regular matrix multiplication, which involves the sum of products, Hadamard multiplication is simpler and operates independently on each element.

**2. Matrix Multiplication:**

Matrix multiplication is the operation of producing a matrix from two matrices. If AAA is an m×nm \times nm×n matrix and BBB is an n×pn \times pn×p matrix, their product CCC is an m×pm \times pm×p matrix where each element CijC\_{ij}Cij​ is given by the dot product of the iii-th row of AAA and the jjj-th column of BBB:

Cij=∑k=1nAik⋅BkjC\_{ij} = \sum\_{k=1}^{n} A\_{ik} \cdot B\_{kj}Cij​=k=1∑n​Aik​⋅Bkj​

Matrix multiplication is not commutative, i.e., A×B≠B×AA \times B \neq B \times AA×B=B×A in general, and requires that the number of columns in the first matrix matches the number of rows in the second matrix.

**3. Transpose of a Matrix and Vector:**

The **transpose** of a matrix is obtained by swapping the rows and columns. If AAA is an m×nm \times nm×n matrix, its transpose, denoted ATA^TAT, is an n×mn \times mn×m matrix. The element AijA\_{ij}Aij​ in the original matrix becomes AjiTA^T\_{ji}AjiT​ in the transposed matrix:

AjiT=AijA^T\_{ji} = A\_{ij}AjiT​=Aij​

For a **vector**, which can be seen as a matrix with one column or one row, the transpose of a column vector results in a row vector and vice versa.

**4. Training Set Batch:**

In machine learning, a **batch** refers to a subset of the training data used to compute the gradients and update the model parameters. There are three main types of training in terms of batch size:

* **Batch Gradient Descent**: Uses the entire training dataset for each update.
* **Mini-batch Gradient Descent**: Uses small batches of data for each update. This is commonly used in practice.
* **Stochastic Gradient Descent (SGD)**: Uses a single training example at each step for an update.

Using mini-batches or SGD helps in reducing the computation time and can improve convergence.

**5. Entropy-Based Loss Function:**

The **entropy-based loss function**, often called **cross-entropy loss**, is used for classification problems. It measures the difference between the true probability distribution (labels) and the predicted probability distribution (model's output).

For a single example, the cross-entropy loss is:

L=−∑i=1Cyilog⁡(y^i)L = - \sum\_{i=1}^{C} y\_i \log(\hat{y}\_i)L=−i=1∑C​yi​log(y^​i​)

Where:

* yiy\_iyi​ is the true label (0 or 1),
* y^i\hat{y}\_iy^​i​ is the predicted probability for class iii,
* CCC is the number of classes.

**Why it's used**: Cross-entropy is widely used for training neural networks because it is sensitive to how close the predicted probabilities are to the true labels. It heavily penalizes confident wrong predictions, which helps improve the model’s performance.

**6. Neural Network Supervised Training Process:**

In supervised training, a neural network learns from labeled training data. The steps in the process are:

1. **Forward propagation**: The input data is passed through the network, and predictions are generated.
2. **Loss computation**: The error between the predicted output and the true labels (targets) is computed using a loss function (e.g., cross-entropy).
3. **Backpropagation**: The error is propagated backward through the network to compute the gradients of the loss with respect to the network's parameters.
4. **Gradient Descent**: The parameters are updated using gradient descent or a similar optimization algorithm.
5. **Repeat**: The process is repeated for several epochs until the model converges or achieves satisfactory performance.

**7. Forward Propagation and Backpropagation:**

* **Forward Propagation**: Forward propagation is the process where input data passes through the neural network layers to generate a prediction. In each layer, the input is multiplied by the weights, and a bias is added. The result is then passed through an activation function to introduce non-linearity.

For example, for a single neuron:

z=W⋅X+bz = W \cdot X + bz=W⋅X+b

The output aaa is then passed through an activation function:

a=σ(z)a = \sigma(z)a=σ(z)

where σ\sigmaσ could be the sigmoid or ReLU activation function.

* **Backpropagation**: Backpropagation is the process of calculating the gradient of the loss function with respect to each weight by the chain rule, starting from the output layer and propagating backwards through the network.

The key steps are:

* 1. Compute the gradient of the loss with respect to the output (derivative of the loss function).
  2. Compute the gradient of the output with respect to the weights in the final layer.
  3. Continue backpropagating through all the layers using the chain rule to update the weights.

Backpropagation ensures that the network's weights are adjusted in a way that reduces the error in future predictions.