**Assignment-1**

**Q.1) Explain the training procedure of simple NN with suitable diagram?**

* **The training procedure of a simple neural network (NN) involves several steps:**

|  |
| --- |
| machine learning Archives - Concord Analytics, LLC |

1. **Initialization:** First, the neural network is initialized by assigning random weights and biases to each connection between the neurons. These initial weights and biases are typically small random numbers.
2. **Forward Propagation:** During the forward propagation phase, the input data is fed into the network, and the activations of each neuron in the network are calculated. The forward propagation process involves the following steps:
   1. **Input Layer:** The input layer receives the input data and passes it to the neurons in the hidden layer.
   2. **Hidden Layer:** The hidden layer performs calculations using the input data and the weights and biases associated with the connections. Each neuron in the hidden layer applies an activation function (such as sigmoid, ReLU, or tanh) to the weighted sum of its inputs.
   3. **Output Layer:** The output layer applies another activation function to the weighted sum of the inputs it receives from the hidden layer.For example, for binary classification, a sigmoid activation function can be used, while for multi-class classification, a softmax activation function is often employed.
3. **Calculation of Loss:** After obtaining the predicted values, the loss function is calculated to measure the discrepancy between the predicted values and the actual values. The choice of the loss function depends on the specific problem being solved.
4. **Backward Propagation (Backpropagation):** Backpropagation is the key step in training the neural network. It involves calculating the gradients of the loss function with respect to the weights and biases in the network. These gradients indicate the direction and magnitude of the adjustments needed to minimize the loss.
5. **Update Weights and Biases:** Once the gradients are calculated, the weights and biases in the network are updated using an optimization algorithm. The learning rate determines the step size of the updates and plays a crucial role in the convergence and stability of the training process.
6. **Iterative Training:** Steps 2 to 5 are repeated for each training sample in the dataset. The iterative process of forward propagation, loss calculation, backward propagation, and weight update is performed on multiple samples to update the network parameters and minimize the overall loss.

**Q.2) Need of loss function, Explain use of MSE, Binary cross entropy, CCE, Log lOSS?**

* A loss function is a mathematical function used to measure the difference between the predicted output and the actual output for a given set of inputs during training. The goal of machine learning is to minimize the value of the loss function, which means that the predicted output is as close as possible to the actual output.
* Different loss functions are used in machine learning depending on the type of problem being solved, such as regression or classification, and the characteristics of the data being used. Here are some commonly used loss functions and their uses:

|  |
| --- |
| https://static.wixstatic.com/media/3eee0b_b454d6ac19f34adf8108503a79e65f0c~mv2.png/v1/fill/w_317,h_102,al_c,lg_1,q_85,enc_auto/3eee0b_b454d6ac19f34adf8108503a79e65f0c~mv2.png |

* **Mean Squared Error (MSE):** MSE is a commonly used loss function in regression problems, where the goal is to predict a continuous numerical output. It measures the average squared difference between the predicted and actual values. The formula for MSE is: **MSE = (1/n) \* ∑(predicted - actual)^2**

|  |
| --- |
| https://static.wixstatic.com/media/3eee0b_a117ec500fb94ebda8ccc68f0d06358c~mv2.png/v1/fill/w_415,h_76,al_c,q_85,usm_0.66_1.00_0.01,enc_auto/3eee0b_a117ec500fb94ebda8ccc68f0d06358c~mv2.png |

* **Binary Cross Entropy:** BCE is a commonly used loss function in binary classification problems, where the output is either 0 or 1. It measures the difference between the predicted probability of the positive class and the actual class. The formula for BCE is: **BCE = -(actual \* log(predicted) + (1-actual) \* log(1-predicted))**

|  |
| --- |
| https://static.wixstatic.com/media/3eee0b_fab6dab7e72449499a18d8b9f09fdd95~mv2.png |

* **Categorical Cross Entropy (CCE):** CCE is a commonly used loss function in multi-class classification problems, where the output is one of several possible classes. It measures the difference between the predicted probability distribution over the classes and the actual distribution. The formula for CCE is:

|  |
| --- |
| ML | Log Loss and Mean Squared Error - GeeksforGeeks |

* **Log Loss (Binary or Multiclass Cross-Entropy Loss):** Log Loss is similar to BCE and CCE, but it computes the logarithm of the predicted probabilities. It is commonly used in logistic regression and neural network models. The formula for Log Loss is:

**Q.3) Need of Activation function, Explain use of TanH, SOftmax, Relu, Leakyrelu, Sigmoid AF?**

* An activation function is a mathematical function that is applied to the output of a neuron in a neural network. The purpose of an activation function is to introduce non-linearity into the output of a neuron, which enables the neural network to learn complex relationships between the input and output data.
* Different activation functions are used in neural networks depending on the type of problem being solved, and the characteristics of the data being used. Here are some commonly used activation functions and their uses:
* **TanH:** TanH (Hyperbolic Tangent) is a commonly used activation function in neural networks. It maps the input values to a range between -1 and 1, which makes it suitable for problems where the output values are also in the same range. Formula: **TanH(x) = (e^x - e^-x) / (e^x + e^-x)**

Where e is the base of the natural logarithm. TanH is used in problems such as image recognition and speech processing.

* **Softmax:** Softmax is an activation function that is commonly used in multi-class classification problems, where the output is one of several possible classes. It maps the input values to a probability distribution over the classes. Formula: **Softmax(xi) = e^xi / ∑j(e^xj)**

Where xi is the input to the ith neuron, and j ranges over all the neurons in the layer. Softmax is used in problems such as image classification and natural language processing.

* **ReLU:** ReLU (Rectified Linear Unit) is a commonly used activation function in deep learning. It maps the input values to a range between 0 and infinity. Formula: **ReLU(x) = max(0, x)**

Where x is the input to the neuron. ReLU is used in problems such as image recognition and speech processing.

* **Leaky ReLU:** Leaky ReLU is similar to ReLU, but it introduces a small positive slope for negative input values, which helps to avoid the problem of "dead neurons" that can occur with ReLU. Leaky ReLU is defined by the following formula: **Leaky ReLU(x) = max(αx, x)**

Leaky ReLU is used in problems such as image classification and natural language processing.

* **Sigmoid:** Sigmoid is a commonly used activation function in neural networks. It maps the input values to a range between 0 and 1, which makes it suitable for problems where the output values are also in the same range. Formula: **Sigmoid(x) = 1 / (1 + e^-x)**

**Q.4) Explain all the Steps in Backpropagation learning with suitable diagram?**

* **Backpropagation** is a popular supervised learning algorithm used in neural networks for training models. It is used to adjust the weights and biases of the network based on the error between the predicted output and the actual output. In this answer, we will explain the various steps involved in backpropagation with the help of a diagram.
* **Here is a step-by-step explanation of the backpropagation algorithm:**
* **Feedforward:** In the first step, the input data is fed into the neural network, and a prediction is made using the current weights and biases. This prediction is compared with the actual output to calculate the error.
* **Backward pass:** In the second step, the error is propagated back through the network, starting from the output layer to the input layer. This process is called the backward pass. The goal of the backward pass is to adjust the weights and biases in such a way that the error is minimized.
* **Gradient calculation:** In the third step, the gradients of the error with respect to the weights and biases are calculated using the chain rule of calculus. The gradients indicate how much each weight and bias contributes to the error.

|  |
| --- |
|  |

* **Weight update:** In the fourth step, the weights and biases are updated using the gradients calculated in the previous step. The learning rate is a hyperparameter that controls how much the weights and biases are updated in each iteration. The weight update rule is given by:

**New weight = Old weight - Learning rate \* Gradient**

* **Repeat:** Steps 1-4 are repeated for a fixed number of iterations or until the error is minimized to a satisfactory level.

**Q.5) What is The Role of Optimizer?**

* In deep learning, optimizers play a critical role in the training process of neural networks. Neural networks typically have large numbers of parameters, which need to be optimized to minimize the error between the predicted output and the actual output. The optimizer is responsible for adjusting the parameters of the network in order to minimize the error.
* The role of the optimizer is to determine the optimal values of the parameters of the neural network. This is done by minimizing a loss function, which measures the difference between the predicted output of the neural network and the actual output.
* The optimizer uses an algorithm to compute the gradients of the loss function with respect to the parameters of the network, and then updates the parameters to minimize the loss function.
* The choice of optimizer is an important decision in the training process of a neural network. Different optimizers have different properties and performance characteristics. Some popular optimizers used in deep learning include:
* **Stochastic Gradient Descent (SGD):** This is a simple and widely-used optimizer that updates the parameters in the direction of the negative gradient of the loss function. SGD works by updating the parameters for each sample in the training set, making it well-suited for large datasets.
* **Adam:** This is an adaptive optimizer that combines the benefits of both RMSProp and Momentum optimizers. Adam uses estimates of both the first and second moments of the gradients to adaptively adjust the learning rate during training.
* **Adagrad:** This is an adaptive optimizer that adjusts the learning rate for each parameter based on the historical gradients for that parameter. Adagrad is well-suited for sparse data because it scales the learning rate based on the frequency of the features.
* The choice of optimizer depends on several factors such as the size of the dataset, the complexity of the neural network, and the nature of the problem being solved. The optimizer is one of the key components in the training process of a neural network, and its proper selection can significantly impact the performance of the model.

**Q.6) Explain in detail gradient descent optimization?**

* Gradient Descent is a widely used optimization algorithm in Deep Learning that is used to minimize the loss function of a model. The goal of the optimization algorithm is to find the values of the model parameters that minimize the loss function. The basic idea behind gradient descent is to iteratively update the model parameters in the direction of the negative gradient of the loss function until convergence.
* The gradient of the loss function with respect to the model parameters is a vector that indicates the direction of steepest descent in the loss function landscape. By updating the model parameters in the direction of this vector, we can find a set of parameters that minimizes the loss function.
* **There are three variants of gradient descent:**

|  |
| --- |
| Gradient Descent in Machine Learning - Javatpoint |

* **Batch Gradient Descent:** In this variant, the model parameters are updated after computing the gradients for the entire training dataset. This can be computationally expensive for large datasets, but it guarantees convergence to the global minimum of the loss function.
* **Stochastic Gradient Descent:** In this variant, the model parameters are updated after computing the gradients for a single training sample. This can be computationally efficient for large datasets, but it can lead to noisy updates and convergence to a local minimum.
* **Mini-batch Gradient Descent:** In this variant, the model parameters are updated after computing the gradients for a small batch of training samples. This combines the advantages of batch and stochastic gradient descent, leading to faster convergence and more stable updates.
* **The update rule for gradient descent is given by: theta = theta - learning\_rate \* gradient**

**where,** theta is the vector of model parameters, learning\_rate is a hyperparameter that determines the step size of the update, and gradient is the gradient of the loss function with respect to the model parameters.

**Q.7) What are the Drawbacks of Gradient Descent?**

* While gradient descent (GD) is a widely used optimization algorithm in Deep learning, it also has some drawbacks that can limit its performance. Some of the key drawbacks of gradient descent are:
* **Slow convergence:** Gradient descent can take a long time to converge to the minimum of the loss function, especially when the function is highly non-convex or has a lot of local minima. This can lead to long training times and limits the ability to explore complex parameter spaces.
* **Sensitive to learning rate**: The learning rate is a hyperparameter that controls the step size in gradient descent updates. If the learning rate is too large, the algorithm may fail to converge or overshoot the minimum. If the learning rate is too small, the algorithm may converge too slowly or get stuck in a local minimum. Finding the right learning rate can be a challenging task.
* **May get stuck in local minima:** Gradient descent can get stuck in local minima of the loss function, which may not be the global minimum. This can limit the performance of the model and lead to suboptimal results.
* **Requires gradient information:** Gradient descent requires computing the gradients of the loss function with respect to the model parameters, which can be computationally expensive for large datasets or complex models. This can limit the scalability of the algorithm.
* **Vulnerable to noise:** Gradient descent can be sensitive to noisy or sparse data, which can lead to noisy gradients and unstable updates.
* To address these drawbacks, various modifications to gradient descent have been proposed, such as momentum-based optimization, adaptive learning rate optimization, and regularization techniques.
* These modifications aim to improve the convergence speed, stability, and robustness of the algorithm. Additionally, other optimization algorithms such as Adam, Adagrad, and RMSProp have been developed that combine the advantages of gradient descent with other optimization techniques to overcome some of its limitations.

**Q.8) What is the significance of Adding momentum in Gradient Descent optimization? Explain in detail ?**

* Adding momentum is a modification of the standard gradient descent optimization algorithm that can help to improve the convergence speed and stability of the algorithm. In traditional gradient descent, the update rule for the model parameters at each iteration is based solely on the current gradient of the loss function. However, in adding momentum, the update rule takes into account the momentum or the direction of the previous updates.
* Specifically, adding momentum involves adding a fraction of the previous update to the current update. This means that the new update is not just based on the current gradient, but also takes into account the accumulated momentum of previous updates. The idea behind adding momentum is that it can help the optimization algorithm to move faster and more smoothly towards the minimum of the loss function.
* The update rule for adding momentum can be written as: **Vt = alpha \* Vt-1 - learning\_rate \* gradient, theta = theta + Vt**

Where, Vt is the momentum vector at time t, alpha is the momentum parameter (usually set to a value between 0.8 and 0.99), and theta is the model parameter vector. The momentum vector is initialized to zero at the beginning of the training process.

* In this update rule, the first term (alpha \* Vt-1) represents the momentum term, which takes into account the accumulated momentum from previous updates. The second term (-learning\_rate \* gradient) represents the current update based on the gradient of the loss function. The two terms are then added together to produce the new update for the model parameters.
* **The addition of momentum in gradient descent optimization can have several benefits, including:**
* **Faster convergence:** The momentum term helps the optimization algorithm to move faster towards the minimum of the loss function by taking into account the accumulated direction of previous updates. This can help to speed up the convergence of the algorithm.
* **Smoother updates:** The momentum term can also help to smooth out the updates by reducing the impact of small fluctuations in the gradient. This can lead to more stable updates and better performance.
* **Improved generalization:** Adding momentum can also help to improve the generalization performance of the model by reducing the risk of Overfitting. By smoothing out the updates and avoiding getting stuck in local minima, adding momentum can help the optimization algorithm to find a better global minimum.

**Q.9) Exaplain in detail RMSprop and ADAM optimizer?**

* RMSprop and Adam are two popular optimization algorithms used in deep learning that address some of the limitations of standard gradient descent optimization. Here is an explanation of both RMSprop and Adam optimizers:
* **RMSprop:** RMSprop is a modification of the gradient descent algorithm that adaptively adjusts the learning rate based on the magnitudes of recent gradients. Specifically, RMSprop divides the learning rate by a running average of the magnitudes of recent gradients for each parameter. The update rule for RMSprop can be written as:
* Compute the gradient of the loss function with respect to the model parameters. Compute a running average of the squares of the gradients for each parameter:

**g\_sq = decay\_rate \* g\_sq + (1 - decay\_rate) \* grad^2**

Compute the update for each parameter:

**theta = theta - (learning\_rate / (sqrt(g\_sq) + epsilon)) \* grad**

* In this update rule, g\_sq is the running average of the squares of the gradients, decay\_rate is a hyperparameter that controls the decay rate of the running average (typically set to 0.9), and epsilon is a small constant added to the denominator to avoid division by zero.
* The key idea behind RMSprop is that it normalizes the learning rate based on the magnitude of recent gradients, which can help to prevent oscillations and slow convergence. By adapting the learning rate for each parameter based on its recent gradient history, RMSprop can also help to deal with the problem of different scales in the gradients for different parameters.
* **Adam:** Adam is another optimization algorithm that combines the advantages of RMSprop and momentum-based optimization. In addition to adaptively adjusting the learning rate based on the magnitude of recent gradients, Adam also maintains a running average of the previous gradients to incorporate momentum information. The update rule for Adam can be written as:

Compute the gradient of the loss function with respect to the model parameters.

Compute a running average of the first and second moments of the gradients for each parameter:

**m = beta1 \* m + (1 - beta1) \* grad, v = beta2 \* v + (1 - beta2) \* grad^2**

Compute bias-corrected estimates of the first and second moments:

**m\_hat = m / (1 - beta1^t), v\_hat = v / (1 - beta2^t)**

Compute the update for each parameter:

**theta = theta - (learning\_rate / (sqrt(v\_hat) + epsilon)) \* m\_hat**

* In this update rule, m is the running average of the first moment (the gradient), v is the running average of the second moment (the squared gradient), beta1 and beta2 are hyperparameters that control the decay rates of the running averages (typically set to 0.9 and 0.999, respectively), epsilon is a small constant added to the denominator to avoid division by zero, and t is the time step (or iteration number).
* The key idea behind Adam is that it adapts the learning rate for each parameter based on both the magnitude of recent gradients and the momentum information. By incorporating momentum information, Adam can help to smooth out the updates and improve the convergence speed and stability of the optimization algorithm.
* RMSprop and Adam are optimization algorithms that address some of the limitations of standard gradient descent optimization. RMSprop adaptively adjusts the learning rate based on the magnitudes of recent gradients, while Adam combines this with momentum information to improve the convergence speed and stability of the algorithm.

**Q.10) What is perceptron? Explain MLP?**

* **A perceptron** is a simple type of neural network that was introduced in the late 1950s. It consists of a single layer of input nodes connected to a single output node, where each input node is connected to the output node with a weighted connection.
* **A multilayer perceptron (MLP),** on the other hand, is a type of neural network that consists of multiple layers of nodes, including one or more hidden layers between the input and output layers. The hidden layers are composed of nodes that apply a nonlinear transformation to the input values, and the weights between the nodes are adjusted during training to minimize a loss function.

|  |
| --- |
| Multi-layer Perceptron in TensorFlow - Javatpoint |

* The architecture of an MLP typically consists of an input layer, one or more hidden layers, and an output layer.
* The input layer consists of nodes that represent the features of the input data, and the output layer consists of nodes that represent the output of the model, which can be a class label, a continuous value, or a probability distribution.
* The hidden layers, as the name suggests, are not directly observable and are used to extract features from the input data.
* The goal of the optimization algorithm is to minimize a loss function that measures the difference between the predicted output of the model and the true output. The weights are updated by computing the gradient of the loss function with respect to the weights and using this gradient to update the weights in a way that decreases the loss.
* The activation function used in an MLP is typically a nonlinear function such as the sigmoid, tanh, ReLU, or softmax function. The choice of activation function depends on the nature of the problem being solved and the type of output being predicted.
* **For example,** the sigmoid function is commonly used for binary classification problems, while the softmax function is used for multiclass classification problems.