Non-Programming Assignment

**Q1.** What is Learning Rate Decay and Why is it Needed?

**Learning rate decay** is a technique used to gradually reduce the learning rate as training progresses. The learning rate determines how much to adjust the model’s weights with respect to the loss gradient during each update.

**Why Learning Rate Decay is Needed:**

* **Stabilize Training**: Early in training, a high learning rate helps the model converge quickly. However, as training progresses, if the learning rate remains high, it can cause the model to oscillate around the minimum rather than converge.
* **Avoid Overshooting**: Lowering the learning rate over time helps the model settle into a more stable state, making it less likely to overshoot the optimal weights.
* **Achieve Better Accuracy**: Using a decaying learning rate can lead to better generalization and improved model performance since it fine-tunes the weights more delicately as it approaches convergence.

**Common Learning Rate Decay Strategies:**

* **Step Decay**: Reduce the learning rate by a fixed factor after a certain number of epochs.
* **Exponential Decay**: Multiply the learning rate by a factor based on an exponential function.
* **Polynomial Decay**: Reduce the learning rate according to a polynomial function.
* **Adaptive Methods**: Algorithms like Adam, RMSprop, and AdaGrad adjust the learning rate dynamically based on the training process.

**Q2.** What are Saddle and Plateau Problems?

**Saddle Points:**

* A **saddle point** is a point in the loss surface where the gradient is zero but it is not a local minimum (it’s like the top of a hill in one direction and the bottom of a valley in another).
* In high-dimensional spaces, saddle points are more common than local minima and can significantly slow down convergence because the gradients near saddle points are close to zero.
* The optimization process might get stuck or take a long time to move away from these regions.

**Plateaus:**

* A **plateau** is a flat region in the loss surface where gradients are very small or nearly zero.
* When the model reaches a plateau, it appears to be stuck because the loss doesn't decrease even though it's not at an optimal point.
* Plateaus can cause training to stagnate, requiring techniques like learning rate decay or momentum to help the optimizer escape.

**Addressing Saddle and Plateau Problems:**

* **Learning Rate Decay**: Helps adjust the learning rate when the model is stuck.
* **Momentum Optimization**: Keeps the optimizer moving past saddle points and plateaus.
* **Adaptive Optimizers**: Methods like Adam and RMSprop adjust the learning rate based on the gradient’s variance.

**Q3. Why Should We Avoid Grid Approach in Hyperparameter Choice?**

The **grid search** approach involves systematically testing every combination of hyperparameters in a predefined range. Although it may seem comprehensive, it has several drawbacks:

**Reasons to Avoid Grid Search:**

* **Inefficiency**: It is computationally expensive and time-consuming, especially for models with many hyperparameters. Testing every combination can be infeasible for large models.
* **Curse of Dimensionality**: As the number of hyperparameters increases, the search space grows exponentially, making it impractical to explore exhaustively.
* **Ineffectiveness**: Many hyperparameter configurations are irrelevant and do not significantly affect model performance. Grid search wastes time on these unimportant combinations.

**Alternatives:**

* **Random Search**: Randomly samples combinations of hyperparameters. It is more efficient than grid search and often finds better configurations in less time.
* **Bayesian Optimization**: Uses probabilistic models to find the optimal hyperparameters more intelligently.
* **Hyperband**: A method that dynamically allocates resources to promising hyperparameter configurations while discarding less effective ones.

**Q4. What is a Mini-batch and How is it Used?**

**What is a Mini-batch?**

* A **mini-batch** is a small, randomly selected subset of the training dataset used to perform one update of the model's weights.
* The size of a mini-batch is usually between **32 to 512** samples, depending on the dataset size and model.

**Mini-batches usage:**

* **Training Efficiency**: Instead of computing the gradient over the entire dataset, mini-batch training is more efficient and faster. It uses a small sample of data to approximate the gradient, allowing for quicker updates.
* **Noise in Gradients**: Mini-batch gradients introduce a bit of randomness, which can help the model escape saddle points and plateaus more effectively than using the entire dataset.
* **Memory Efficiency**: Using mini-batches reduces memory requirements since only a portion of the dataset is loaded into memory at a time.

**Mini-batch Gradient Descent:**

* **Stochastic Gradient Descent (SGD)**: Uses a single data point for each update (extreme mini-batch of size 1).
* **Mini-batch Gradient Descent**: Strikes a balance between full-batch gradient descent (which uses the entire dataset) and stochastic gradient descent.
* **Full-batch Gradient Descent**: Uses the entire dataset for each update but is computationally expensive and slow.