**Question 1 – Comparison of Optimization Methods**

**1. Gradient Descent (GD)**

* Uses the whole dataset to update weights.
* Main Problem: Very slow on large datasets.
* Next Solution: SGD speeds up training.

**2. Stochastic Gradient Descent (SGD)**

* Updates weights one sample at a time (fast).
* Main Problem: Very noisy updates.
* Next Solution: Mini-Batch reduces noise.

**3. Mini-Batch Gradient Descent**

* Uses a small batch (e.g., 32 samples).
* Main Problem: Still zigzags in narrow valleys.
* Next Solution: Momentum smooths updates.

**4. Momentum**

* Adds velocity to reduce oscillation.
* Main Problem: Can overshoot the minimum.
* Next Solution: NAG looks ahead to avoid overshoot.

**5. Nesterov Accelerated Gradient (NAG)**

* Looks ahead before updating.
* Main Problem: One learning rate for all weights.
* Next Solution: AdaGrad adapts learning rates.

**6. AdaGrad**

* Adaptive per-parameter learning rate.
* Main Problem: Learning rate shrinks too much.
* Next Solution: RMSProp stabilizes learning rate.

**7. RMSProp**

* Controls learning rate with moving averages.
* Main Problem: Lacks momentum.
* Next Solution: Adam combines momentum + adaptivity.

**8. Adam**

* Combines Momentum and RMSProp.
* Main Problem: Poor weight decay handling.
* Next Solution: AdamW improves generalization.

**9. AdamW**

* Fixes weight decay behavior in Adam.
* Main Problem: No Nesterov look-ahead.
* Next Solution: Nadam adds Nesterov step.

**10. Nadam**

* Adam + Nesterov look-ahead for faster convergence.

**Summary Table**

The following table summarizes the relation between the optimizers:

|  |  |  |
| --- | --- | --- |
| Optimizer | Main Problem | Next Solution |
| GD | Too slow | SGD |
| SGD | Too noisy | Mini-Batch |
| Mini-Batch | Zigzag | Momentum |
| Momentum | Overshooting | NAG |
| NAG | One LR for all | AdaGrad |
| AdaGrad | LR shrinks | RMSProp |
| RMSProp | No momentum | Adam |
| Adam | Bad weight decay | AdamW |
| AdamW | No look-ahead | Nadam |
| Nadam | - | - |

Question 2:

<https://colab.research.google.com/drive/1fL9O1zkOVb5SDkHqc1XnrWA9CB3m4q89#scrollTo=PTME7Y1bHvLh>

**1) Purpose**

This code implements a Linear Regression model trained using Stochastic Gradient Descent (SGD) from scratch, without machine learning libraries. It demonstrates how to fit a predictive model through manual gradient computation and parameter updates.

**2) Data Handling**

The program reads a CSV file and automatically selects all numeric columns. The last numeric column is treated as the target variable, while the remaining numeric columns are used as features for the regression model.

**3) Preprocessing (Standardization)**

All features are standardized to zero mean and unit variance. Standardization improves convergence speed and prevents large-scale features from dominating the gradient updates.

**4) Splitting Strategy**

The dataset is split into training, validation, and testing subsets. Validation data is used to monitor training progress, while the final test set is used to evaluate the model after training.

**5) Model Definition**

The model predicts the target using a linear function of the input features:

where represents the weight vector and is a bias term.

**6) Loss Function (MSE)**

The model uses the Mean Squared Error (MSE) as the objective function to minimize:

MSE measures how close the predictions are to the actual target values.

**7) Optimization Using SGD**

Stochastic Gradient Descent updates model parameters after each small batch (or individual sample). This allows faster updates and enables learning from large datasets more efficiently than full-batch methods.

**8) Parameter Updates**

Weights and bias are updated by subtracting the product of the learning rate and the gradient of the loss function. This process continues for multiple epochs to gradually reduce the error.

**9) Early Stopping**

Early Stopping monitors validation MSE during training. If the model does not improve for several consecutive epochs (patience = 6), training stops automatically and the best weights are restored to avoid overfitting.

**10) Model Evaluation**

After training, the model’s performance is measured using MSE on the training, validation, and test sets. A learning curve is also plotted to visualize convergence over epochs.