ENPM703- Assignment-2

Part1: Fully Connected Neural Net

Architecture

Affine: Each layer, starts with an affine transformation, where the input features are multiplied by a weight matrix and a bias vector is added. This results in a linear combination of the input, transforming it into a different feature space.

RELU: The ReLU activation introduces non-linearity by passing the input directly if it's positive and outputting zero if it's negative. This non-linear activation is essential for enabling the network to learn complex, non-linear patterns in the data.

The structure of affine → RELU is repeated for the first L - 1 layers, meaning these layers share the same configuration. This repetition of layers allows the network to progressively transform and refine the data across several hidden layers, learning increasingly abstract representations at each stage.

Final Layer: For the final layer, another affine transformation is applied, followed by the softmax function. The affine layer computes the scores of each class and softmax converts the resulting scores into class probabilities, ensuring the output represents a proper probability distribution across the possible classes.

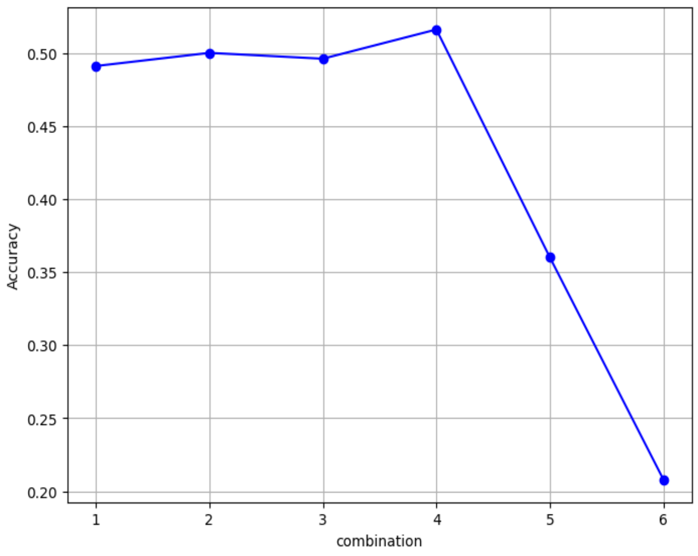
Optimizers

1. **SGD + Momentum**: In this method, weights are updated based on the velocity of the gradient rather than directly using the gradient. By adding a fraction of the previous update to the current one, it helps smooth the update path, preventing the optimization from getting stuck in local minima and speeding up progress in areas with small gradients.
2. **RMSProp**: RMSProp modifies the learning rate for each parameter by using a moving average of recent gradients. It maintains a moving average of squared gradients to normalize updates, which prevents large oscillations and stabilizes the training process.
3. **Adam**: Adam incorporates the benefits of both SGD with momentum and RMSProp. It tracks an exponential moving average of both the gradients (first moment) and squared gradients (second moment), enabling it to adapt the learning rate for each parameter. Additionally, it includes bias correction to address initial overshoots during the early stages of training.

Hyperparameters Tuning

Learning Rate: The learning rate dictates the magnitude of the steps taken during optimization; a higher learning rate may accelerate convergence but risks overshooting the minimum, whereas a lower learning rate allows for more precise updates but can hinder the training process.

Weight Scale: Weight scaling affects to the initialization of weights in a neural network, weights that are excessively small can delay convergence due to vanishing gradients, while weights that are excessively large can result in extreme gradients, causing instability in training.



1: [‘lr:2e-3’, ‘weight_scale: 5e-2'],
2: [‘lr: 2e-3', ‘weight_scale: 1e-2'],
3: [‘lr: 1e-3', ‘weight_scale: 5e-2'],
4: [‘lr: 1e-3', ‘weight_scale: 1e-2'],
5: [‘lr:1e-5', ‘weight_scale: 5e-2'],
6: [‘lr:1e-5, ‘weight_scale: 1e-2']