COL774 A2.1 Report

Strategies Experimented in Part C:

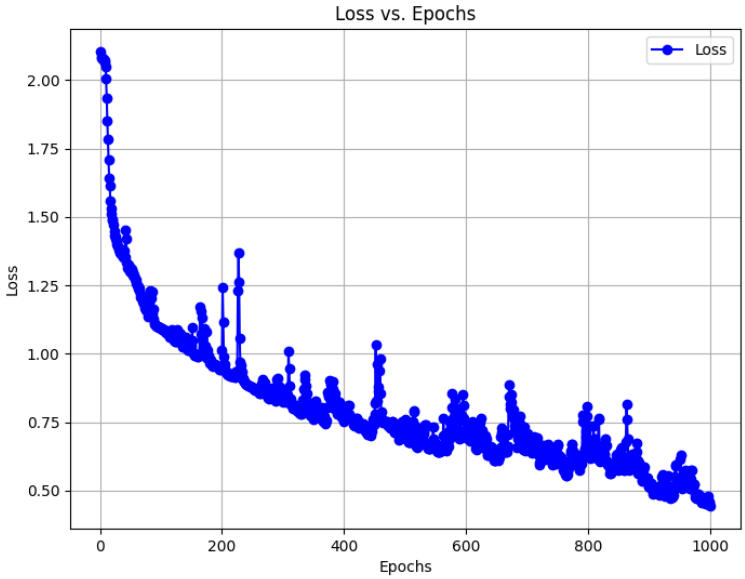
1. Adam's Adaptive Learning Rate Strategy:

   The key idea behind Adam is that it computes individual learning rates for each parameter based on the moving averages of the gradient (first moment) and the squared gradient (second moment).
   Adam maintains two-moment estimates:
   m_t, the exponentially decaying average of past gradients, and **v_t**, the exponentially decaying average of past squared gradients. These moments help in smoothing the gradients and controlling the learning rate. The final update for each parameter is based on these corrected moments, allowing Adam to adaptively change learning rates for each parameter.

   Adam introduces several hyperparameters:

   - **Learning rate (α)**: The step size used for updating the parameters.
   - **Beta1 (β1)**: The exponential decay rate for the first moment (gradient), which controls how much of the past gradients are remembered.
   - **Beta2 (β2)**: The exponential decay rate for the second moment (squared gradient), controlling how much the algorithm considers recent versus older gradient values.
   - **Epsilon (ε)**: A small constant added for numerical stability to prevent division by zero when updating parameters.

   The variation of epochs vs loss in this method is as below:

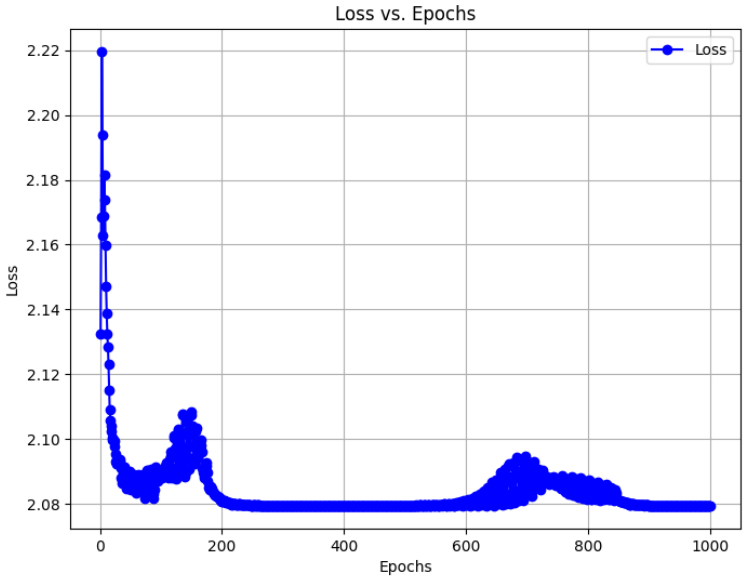


The loss varies from 2.13 to 0.45 over 1000 epochs. This strategy performs the best among all others and hence is implemented in our final submission of part c.

2. Momentum :

   Momentum updates weights by using a combination of the current gradient and the exponentially weighted average of past gradients, which helps accelerate movement in consistent directions and smooths out oscillations.

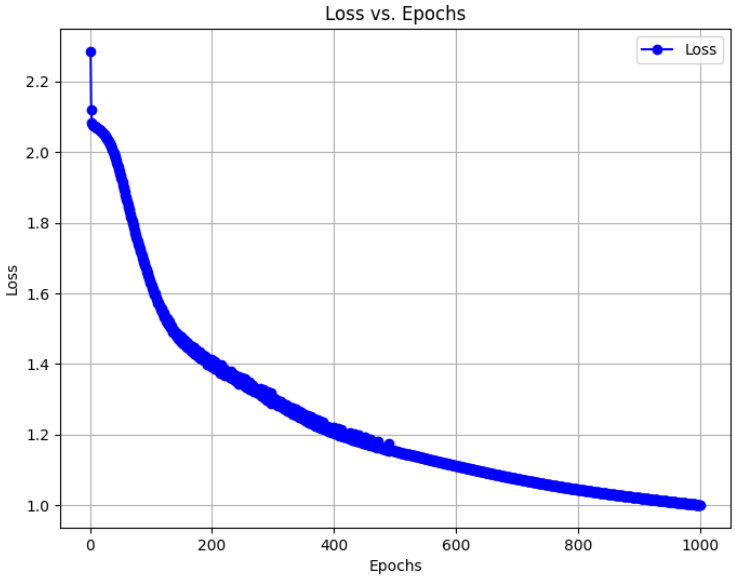
   We implemented a method that involves combining both Momentum and Adam's Strategy, however the performance was poorer (loss : 2.22 to 2.08 over 1000 epochs) compared to the previous strategy. The graph shows the variation of loss vs epochs for this implementation:



3. RMSprop (Root Mean Square) :

   It is an adaptive learning rate method that adjusts the learning rate based on recent gradients. By keeping a running average of the squared gradients for each parameter, RMSprop ensures that frequently occurring updates are dampened, while less frequent ones are given more significance.

   The loss varies from 2.3 to 1.0 over 1000 epochs as shown in the graph below:



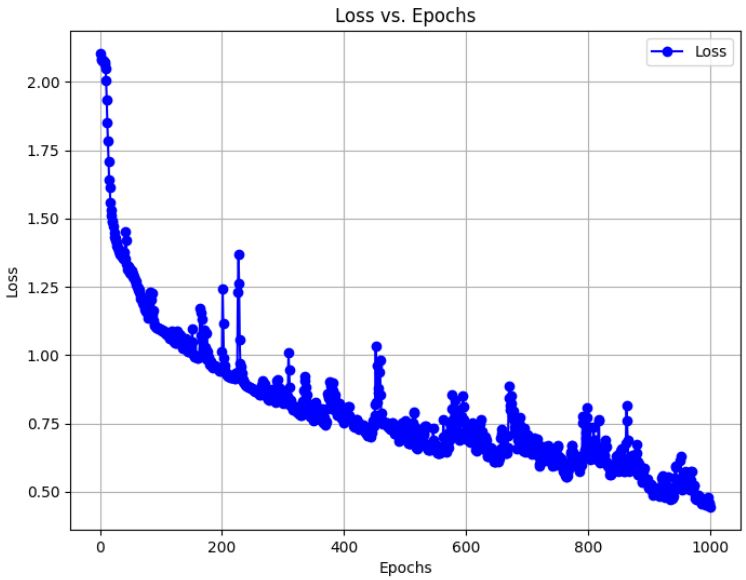
4. We also tried varying other parameters like batch_size, learning_rate and the hyperprameters as per the strategy and the best ones were finally implemented.

Strategies Implemented in Part D:

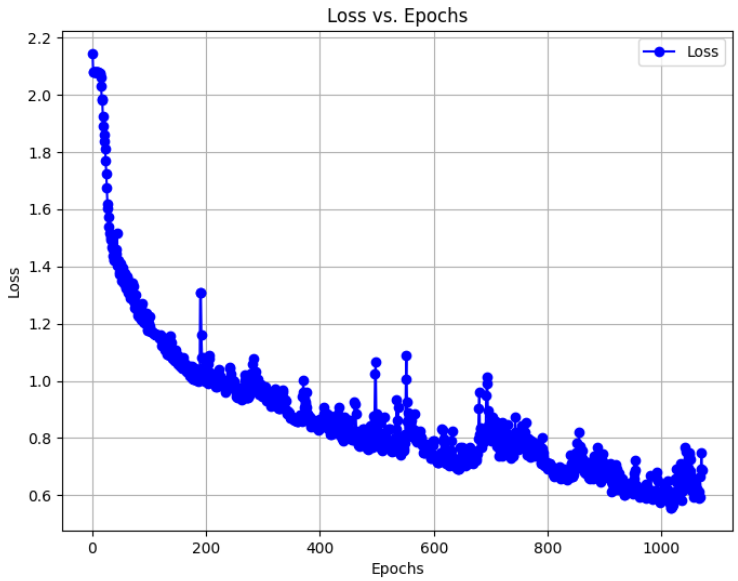
In this part, we used the same learning rate strategy as in part c. Instead we tried changing the network architecture, the following graphs show the different architectures we tried:

1. Baseline Part C Architecture:
    - Input : 625 units
    - Fully Connected Layer (fc1): 625→512
    - Fully Connected Layer (fc2): 512→256
    - Fully Connected Layer (fc3): 256→128
    - Fully Connected Layer (fc4): 128→32
    - Output Layer: 32→8

    The graph below shows the performance :



2. Deepened Network (Adding One More FC Layer) :
    - **Input layer**: 625 units
    - **Layer 1 (Fully connected)**: 625 → 512
    - **Layer 2 (Fully connected)**: 512 → 256
    - **Layer 3 (Fully connected)**: 256 → 128
    - **Layer 4 (Fully connected)**: 128 → 64
    - **Layer 5 (Fully connected)**: 64 → 32
    - **Output layer**: 32 → 8

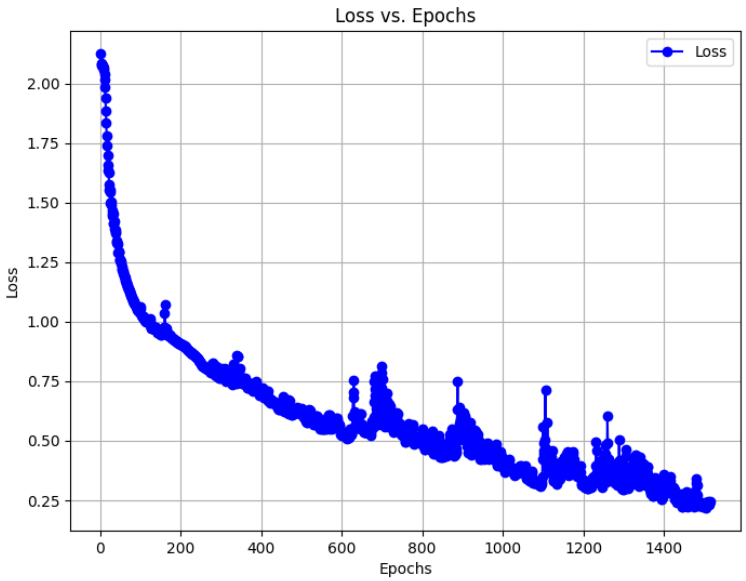


The above graph shows that adding more layers or more parameters doesn't necessarily improve the performance, instead slows down the speed of convergence since in this case loss decreases only upto 0.75. in 13 mins.

3. Slimmed Network (Decreasing one FC Layer):
    - **Input layer**: 625 (input features)
    - **Layer 1 (Fully connected)**: 625 → 512
    - **Layer 2 (Fully connected)**: 512 → 128

- **Layer 3 (Fully connected)**: 128 → 32
- **Output layer**: 32 → 8

This network has lesser number of parameters compared to the 2 networks above and shows best performance by decreasing loss upto 0.2 in 13 mins. It also runs for greater number of epochs compared to others. The below graph displays the variation:



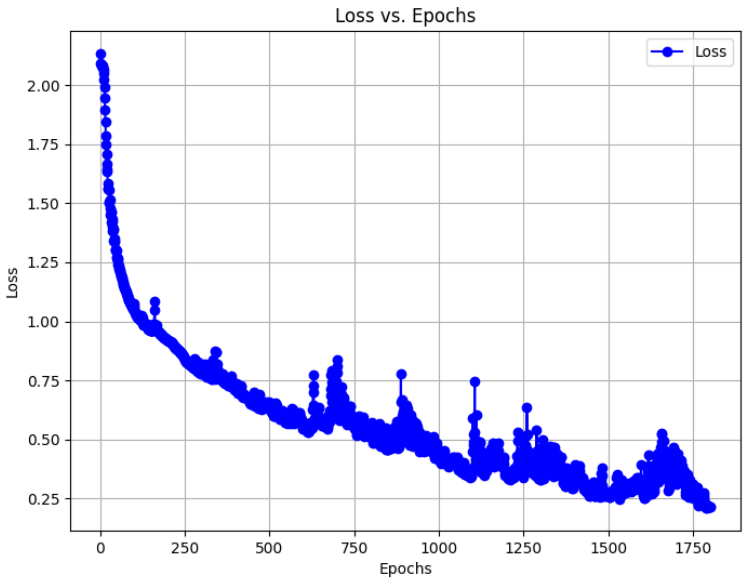
This network is implemented in our final submission.

Other Strategies Tried:

a. **Weight Penalty**: Introducing L2 Regularization for Better Generalization

   We incorporated an L2 regularization penalty into the loss function to prevent large weights. However, the model's performance slightly declined compared to previous versions. This might be attributed to the increased computation time per epoch due to the regularization, leading to fewer epochs being completed within the 13-minute time limit.

   The graph below shows the variation of epochs vs loss for this model.



b. **Optimal Tracking**: Storing Global Minimum Weights and Biases

   As shown in the graph, the loss curve demonstrates fluctuating behavior with multiple spikes, indicating local maxima and temporary increases in loss before eventually stabilizing. To address this, we introduced global weights and biases, which tracked the parameters that produced the minimum loss across all epochs. While this approach reduced the training loss to 0.12 within 13 minutes, the validation accuracy dropped to 62%, which was lower than previous implementations. This decline in validation performance is likely due to the overfitting of the training data.

c. **Stabilizing Training**: Learning Rate Decay and Gradient Clipping for Smoother Convergence

   To mitigate the fluctuations, we experimented with additional techniques, such as applying a decay rate that reduces the learning rate after every 50 epochs and clipping the gradients to prevent large updates to the weights and biases. However, this approach proved computationally expensive, and the reduced learning rates in the later stages slowed down convergence toward optimal weights and biases. As a result, the overall performance was lower than anticipated.

Summary

Strategies	Accuracy on train set (in%)	Cross entropy loss on train set	Accuracy on val set (in%)	Cross entropy loss on val set
Baseline Part C	83.15	4.82	62.21	10.41
Deepened Network	74.06	7.16	61.75	10.56
Slimmed Network	89.09	3.01	65.12	9.57
Weight Penalty	82.69	4.78	62.38	10.39
Optimal Tracking	86.06	3.85	62.00	10.49
Stabilizing Training	79.26	5.70	62.50	10.36