Deep Learning Report

Lab Assignment - 3

Ayush Abrol
B20AI052

Question 01

Aim: Use ResNet18 pre-trained on ImageNet. Finetune the model on CIFAR100 dataset for classification task and plot curves for training loss and training accuracy. Report the final top-5 test accuracy. Perform the above task with the following 3 optimizers.

1. Adam

2. Adagrad

3. RMSprop

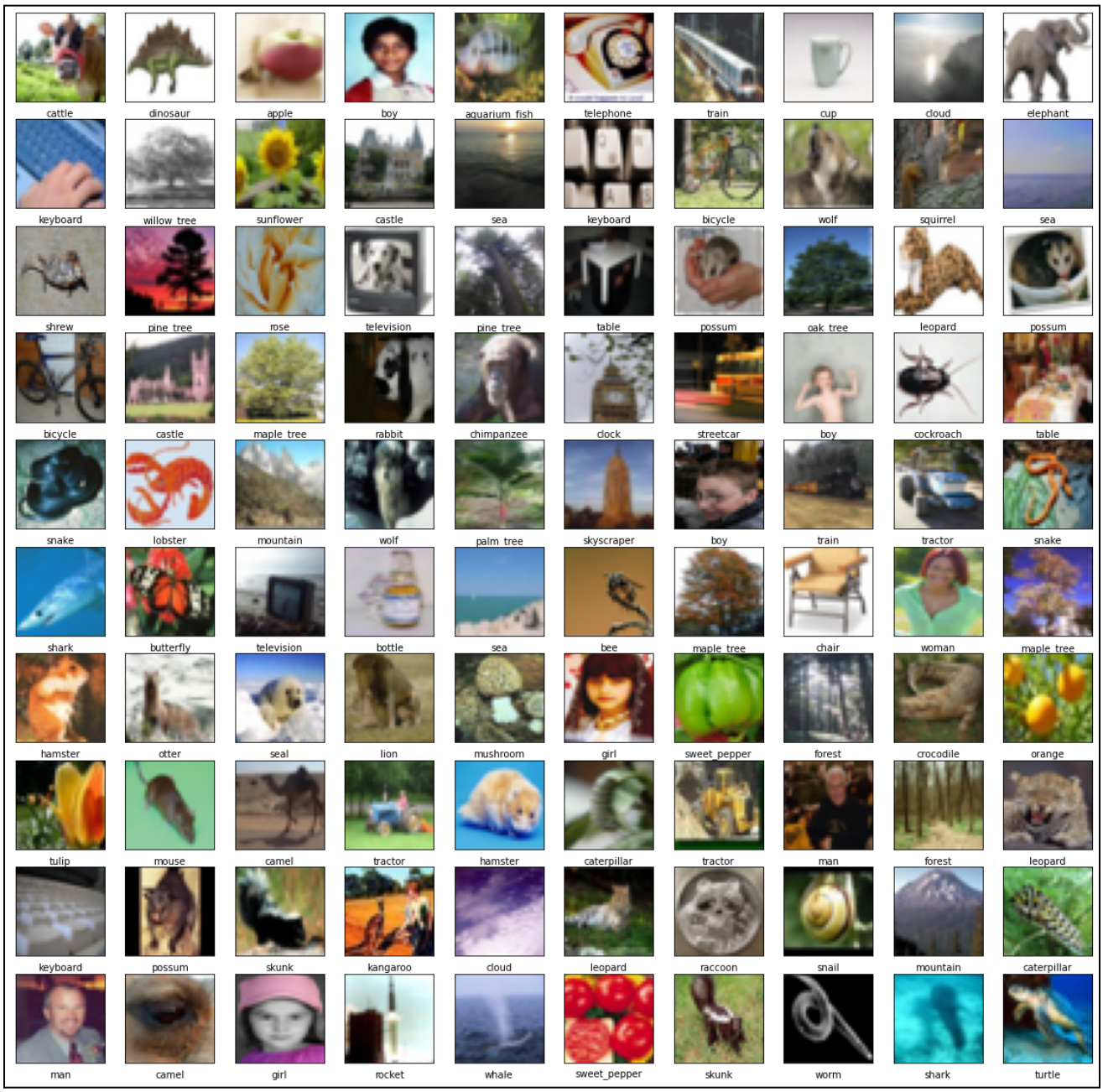
Analyze the results of each optimizer and mention the reason why one is better than the other, or why not. Further, for each optimizer, you need to vary the value of attributes and show its effect on the overall performance. (For ex - In Adam you can change the values of attributes like beta_1, beta_2, epsilon, momentum, weight_decay etc.)

Note: Here, as I have used the **CIFAR100 Dataset** as last digit of my roll no. is even (B20AI052). And out of the 4 optimizers provided (Adam, Adagrad, AdaDelta and RMSProp), I have chosen the above mentioned three optimizers.

Procedure:

- All the necessary dependencies and libraries were taken care of initially like:
    - numpy, pandas, matplotlib for data manipulation and visualization.
    - Torch, torchvision, torch.nn, PIL for for implementing Deep Learning architectures where necessary using PyTocrch.
    - Imported **torchvision.models to import the pre-trained model of ResNet18 architecture trained on ImageNet dataset.**
- Then, set the environment to GPU.
- Then, I loaded the pre-trained model (ResNet18) on ImageNet dataset with 1000 output classes. Therefore, I **fine-tuned** the model to convert the output number of nodes of the last **Fully Connected layer to 100** as we have to train for 100 classes of CIFAR100 dataset.
- After that, I loaded the CIFAR100 dataset with **batch_size = 128** and applied a transform so that it transformed into Tensor format, is tuned to be passed into the pre-trained ResNet18 model is normalized accordingly.

- Visulized the CIFAR100 dataset.



- After fine-tuning the model, the output layer has the output classes as 100 because the dataset used is CIFAR100 which has 100 classes.

```
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
...
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=100, bias=True)
)
```

- Defined the loss function as Cross Entropy loss and trained the model for Adam optimizer, Adagrad Optimizer and RMSProp optimizer with default values of the hyperparameters as given in the official PyTorch documentation and set the learning rate to 1e-3. Number of epochs were set to 10 only due to GPU constraints.
    - Training on Adam optimizer

```
Epoch: 1 Training Loss:  2.9484739785304157      Training Accuracy:  0.26724
Epoch: 2 Training Loss:  2.2712170592964154      Training Accuracy:  0.39784
Epoch: 3 Training Loss:  2.0303160049726285      Training Accuracy:  0.45364
Epoch: 4 Training Loss:  1.8612307130223344      Training Accuracy:  0.48918
Epoch: 5 Training Loss:  1.742896107151685       Training Accuracy:  0.51608
Epoch: 6 Training Loss:  1.64370420735205554     Training Accuracy:  0.54014
Epoch: 7 Training Loss:  1.5552414850810605      Training Accuracy:  0.56172
Epoch: 8 Training Loss:  1.479739181830755       Training Accuracy:  0.5801
Epoch: 9 Training Loss:  1.4337613491146155      Training Accuracy:  0.58896
Epoch: 10 Training Loss:  1.4283446756470235     Training Accuracy:  0.5905
Training complete!
```

    - Testing using Adam optimizer and reporting the top-5 accuracy.

```
Top-5 test accuracy for Adam Optimizer:  0.875
Test Accuracy for Adam Optimizer:  0.5400514240506329
```

- Training on AdaGrad optimizer

```
Epoch: 1 Training Loss:  3.0250805558450997      Training Accuracy:  0.27354
Epoch: 2 Training Loss:  2.3077528626107804      Training Accuracy:  0.40312
Epoch: 3 Training Loss:  2.0910195356134866      Training Accuracy:  0.44542
Epoch: 4 Training Loss:  1.9537283380318176      Training Accuracy:  0.4775
Epoch: 5 Training Loss:  1.8591367545944955      Training Accuracy:  0.49814
Epoch: 6 Training Loss:  1.7798262651619094      Training Accuracy:  0.51732
Epoch: 7 Training Loss:  1.720074597831882       Training Accuracy:  0.53222
Epoch: 8 Training Loss:  1.6647907562572937      Training Accuracy:  0.54402
Epoch: 9 Training Loss:  1.6155047291684943      Training Accuracy:  0.5575
Epoch: 10 Training Loss:  1.5725262509587477     Training Accuracy:  0.5659
Training complete!
```

- Testing using AdaGrad optimizer and reporting the top-5 accuracy.

```
Top-5 test accuracy for Adagrad Optimizer:  0.8125
Test Accuracy for Adagrad Optimizer:  0.5266020569620253
```

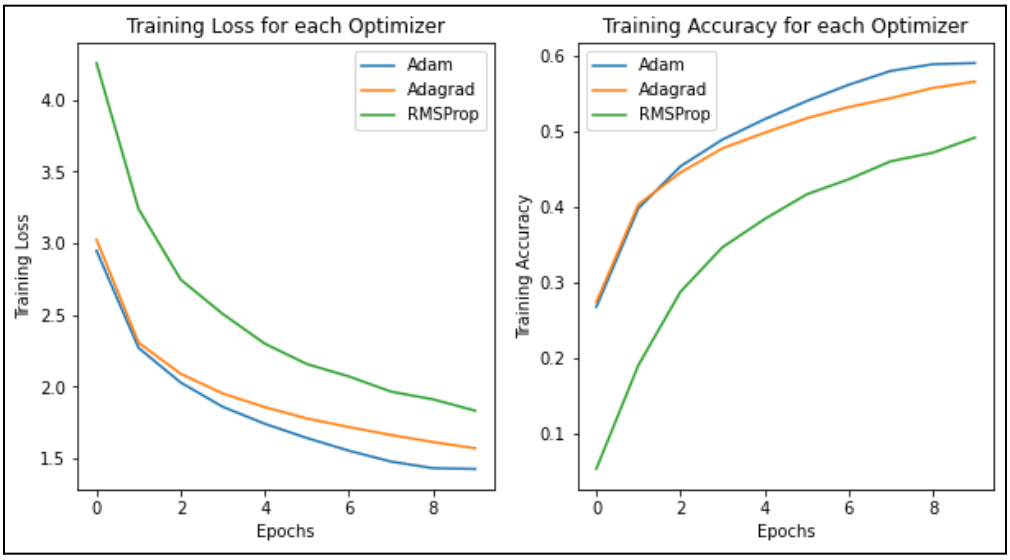
- Training on RMSProp optimizer.

```
Epoch: 1 Training Loss:  4.157836231124371       Training Accuracy:  0.06914
Epoch: 2 Training Loss:  3.093010031048904       Training Accuracy:  0.218
Epoch: 3 Training Loss:  2.6508499496733138      Training Accuracy:  0.30546
Epoch: 4 Training Loss:  2.403722416104563       Training Accuracy:  0.36086
Epoch: 5 Training Loss:  2.221684978136321       Training Accuracy:  0.39946
Epoch: 6 Training Loss:  2.0926798091215244      Training Accuracy:  0.43192
Epoch: 7 Training Loss:  1.9947946486265764      Training Accuracy:  0.4553
Epoch: 8 Training Loss:  1.8992740839643552      Training Accuracy:  0.47602
Epoch: 9 Training Loss:  1.8325910464577053      Training Accuracy:  0.49006
Epoch: 10 Training Loss:  1.7541112707703925     Training Accuracy:  0.50802
Training complete!
```

- Testing using RMSProp optimizer and reporting the top-5 accuracy.

```
Top-5 test accuracy for RMSProp Optimizer:  0.8125
Test Accuracy for RMSProp Optimizer:  0.46914556962025317
```

- Plotted curves between the Training losses and training accuracies for all three optimizers.

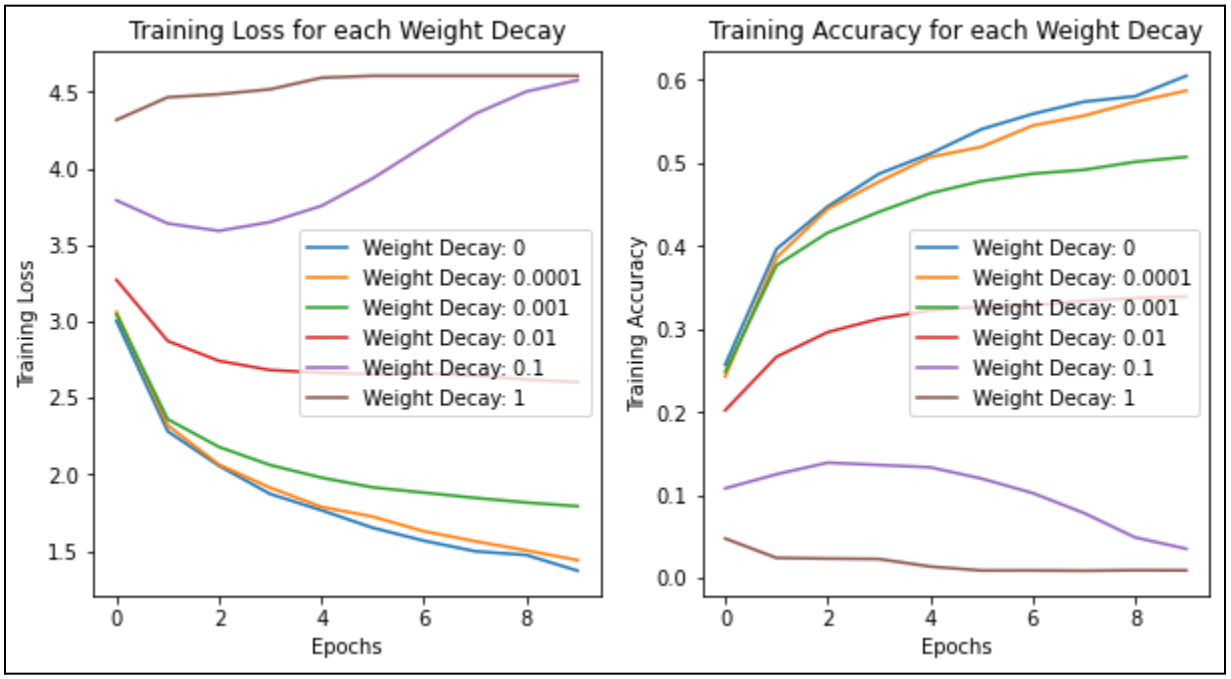


- Here, we observe that Adam optimizer has performed the best amongst all the three, this is due to the reason that Adam optimizer uses adaptive learning rates, which means that it can adjust the learning rate for each parameter based on the magnitude of its gradients. This helps to speed up the convergence of the optimization process and prevent oscillations and Adam optimizer also includes a momentum term that helps to accelerate the optimization process in the direction of the gradients, similar to other momentum-based optimizers. This momentum term helps to reduce the impact of noisy gradients and increase the stability of the optimization process and Compared to Adagrad and RMSProp, Adam optimizer requires fewer iterations to converge, making it more computationally efficient.

- Then, I moved on to the hyper-parameter tuning part.
- First of all, I tuned the different parameters for the Adam optimizer.
    - Weight Decay
    - The weight decay term is added to the gradient before it is used to update the weights, so it does not change the direction of the update. Instead, it adds a regularization term that encourages the weights to be small. The

strength of the weight decay penalty is controlled by the weight decay parameter, which is typically set to a small value, such as 0.001 or 0.0001.

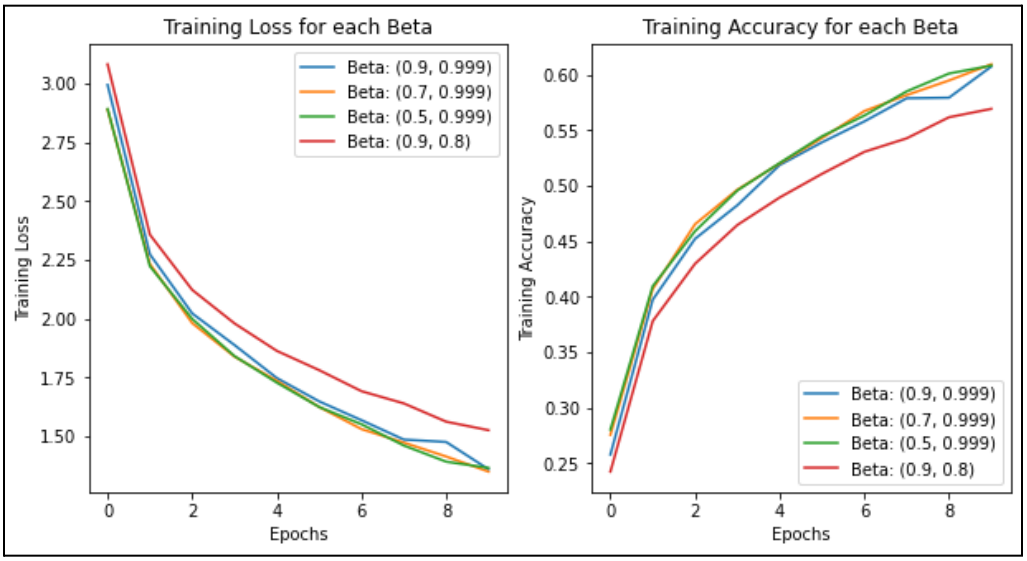
- I iterated over the values [0, 0.0001, 0.001, 0.01, 0.1, 1] for weight_decay parameter and got the following results.



- Here, we can observe the keeping the value of weight_decay parameter to be tending to zero is better than others as when the weight decay parameter is set to a large value, the penalty term can dominate the loss function and cause the weights to be pushed towards zero, resulting in underfitting. On the other hand, when the weight decay parameter is set to a small value or zero, the regularization penalty is negligible, and the optimization algorithm focuses primarily on minimizing the training loss.
- Then, I tuned the beta1 and beta2 parameters for the Adam optimizer.
- These are the coefficients used for computing running averages of gradient and its square.
- Iterated over the values:

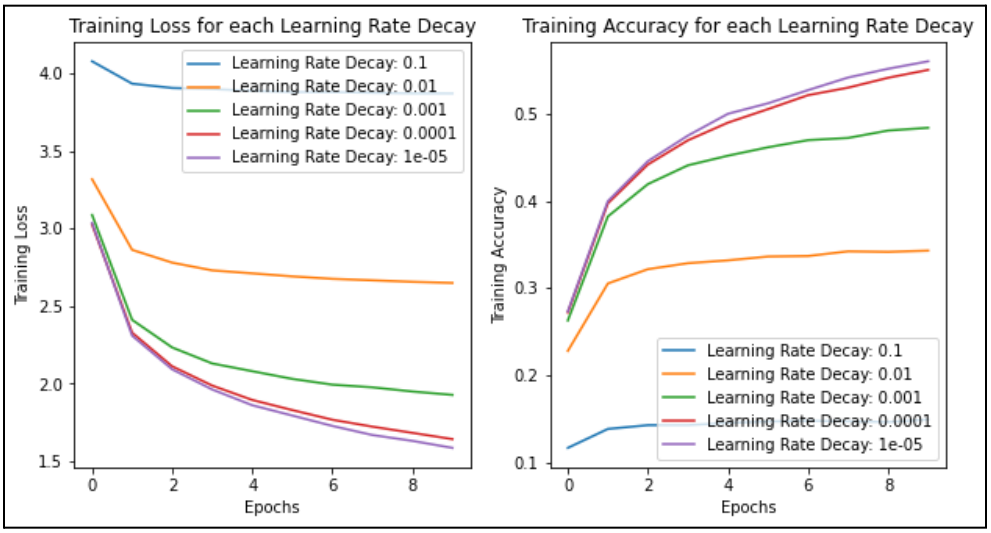
  betas = [(0.9, 0.999), (0.7, 0.999), (0.5, 0.999), (0.9, 0.8), (0.9, 0.6), (0.9, 0.4)]

- And got the following results:

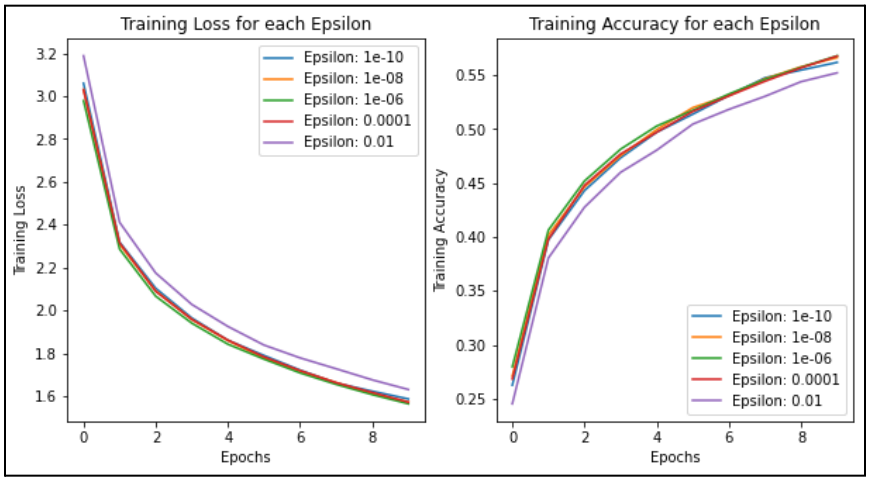


- In summary, the values of beta = (0.9, 0.999) are often used in Adam optimizer because they strike a good balance between giving enough weight to recent gradient information and avoiding oscillations and instability in the optimization process. These values have been found to work well for a wide range of optimization problems and are a good default choice in many cases.

- Then, I tuned the different parameters for the AdaGrad optimizer.
  - Tuned the lr_decay parameter.
  - The idea behind lr_decay is to slowly decrease the learning rate over time as the optimization algorithm gets closer to the optimum, so that the updates become smaller and more precise. This can help prevent the optimization algorithm from overshooting the minimum and oscillating around it, leading to faster convergence and better optimization performance.
  - Iterated over the following values of the lr_decay parameter for AdaGrad:
  - lr_decays = [0.1, 0.01, 0.001, 0.0001, 0.00001]
  - Obtained the following results:

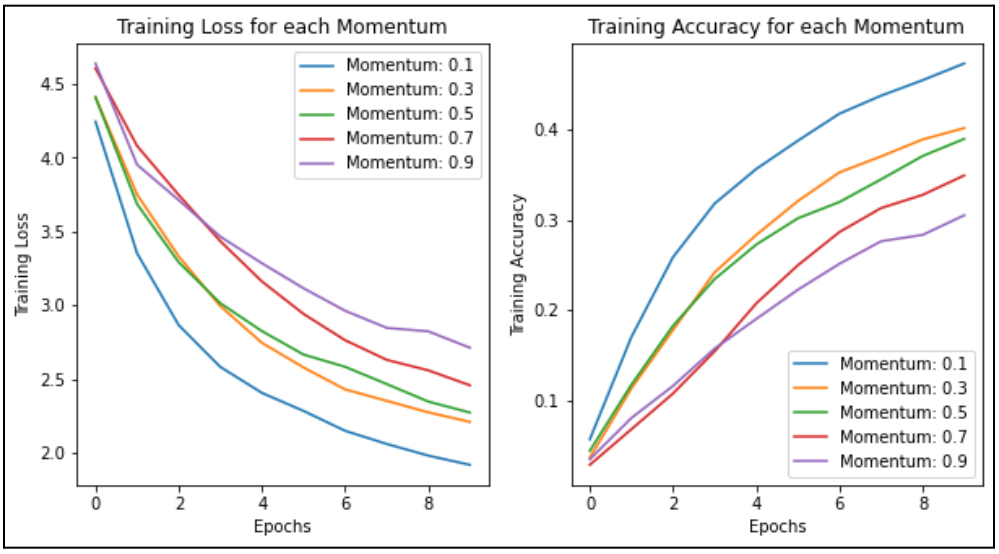


- From the obtained results, I inferred that the lr_decay parameter is usually set to a small value, which means that the learning rate is reduced by 10% or 1% after each update step, respectively. If lr_decay is set to 1, the learning rate remains constant throughout training.
- Then, I tuned the epsilon parameter for the AdaGrad optimizer.
- The epsilon parameter is added to the denominator to ensure that the division is well-defined and to prevent the learning rate from becoming too large when the sum of squared gradients is small. In other words, epsilon acts as a regularization term that helps stabilize the optimization process and prevent numerical instabilities.
- Iterated over the values eps = [1e-10, 1e-8, 1e-6, 1e-4, 1e-2] and obtained the following results.



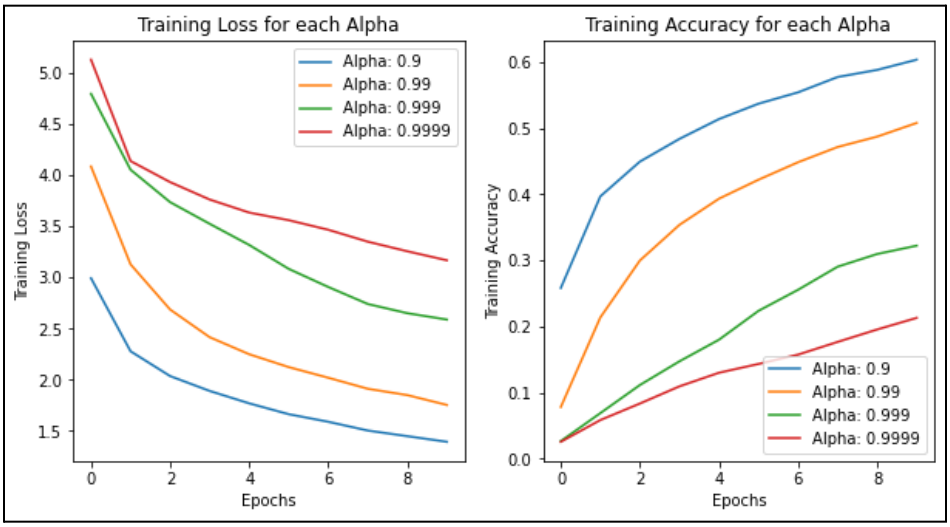
- The value of epsilon is usually set to a small positive constant, such as 1e-8 or 1e-6. The exact value of epsilon depends on the specific optimization problem and the scale of the gradients. If epsilon is too small, it may not have a noticeable effect on the optimization process. On the other hand, if epsilon is too large, it may overly stabilize the optimization process and slow down convergence.

- I moved on to tune the different parameters for the RMSProp optimizer.
  - Tuned the momentum parameter.
  - RMSProp optimizer does not have a momentum parameter in the same way that other optimization algorithms like SGD with momentum or Adam do. Instead, it uses a moving average of the squared gradients to rescale the learning rate.
  - Iterated over the following values of momentum parameter (momentum factor).
  - momentum_ = [0.1, 0.3, 0.5, 0.7, 0.9]
  - Obtained the following results.

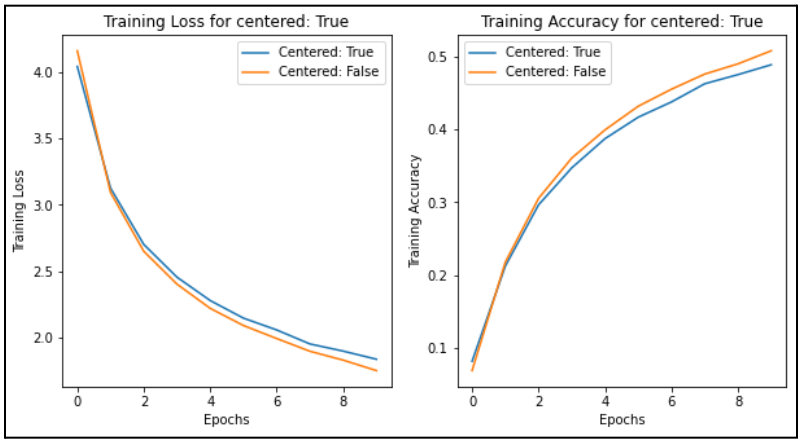


  - After that, I tuned the alpha parameter for the RMSProp optimizer.
  - Alpha is called the smoothing constant which controls the decay rate of the moving average of the squared gradients.

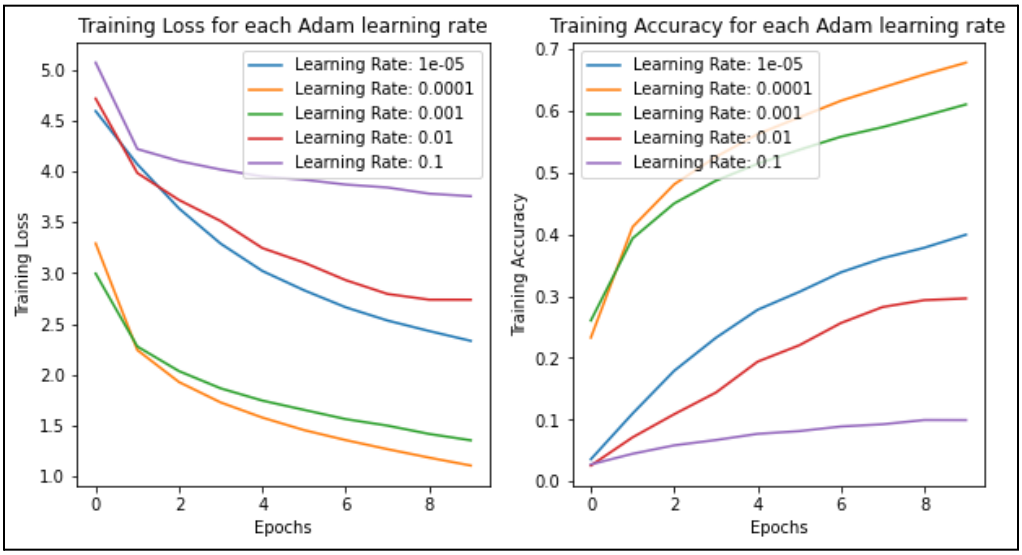
- Iterated over the following values of alpha parameter and obtained the following results.



- The value of alpha is usually set to a value between 0.8 and 0.999. A high value of alpha places more weight on recent gradients, which can help the optimizer adapt quickly to changes in the landscape of the loss function. However, a high value of alpha can also result in overfitting or instability. On the other hand, a low value of alpha places more weight on historical gradients, which can help the optimizer find a smooth trajectory towards the minimum, but may also slow down convergence.
- Then, I also compared the models when I tuned centered parameter for RMSProp Optimizer (If True, gradients are normalized by the estimated variance of the gradient; if False, by the uncentered second moment).
- Results were as follows:

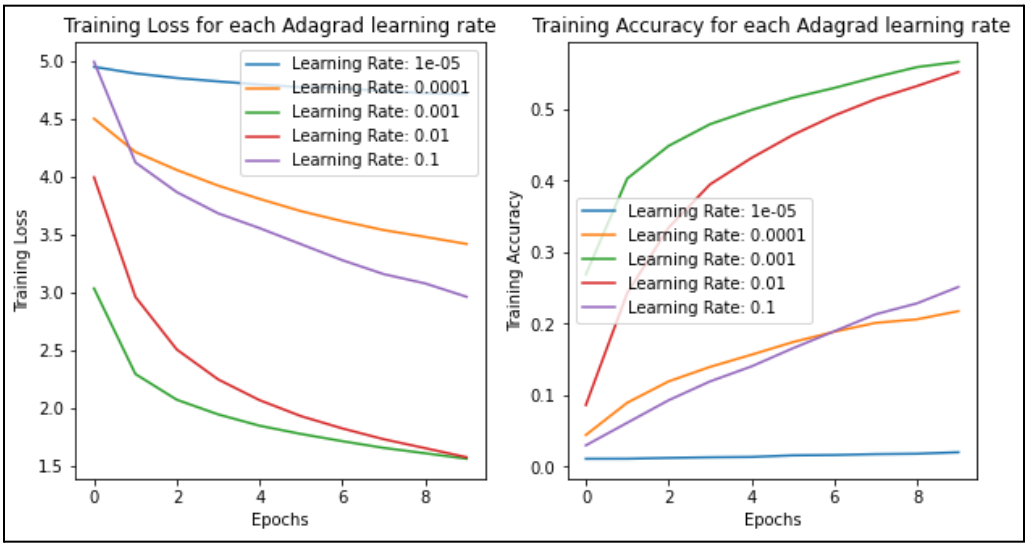


- Finally, I tuned all the three optimizers for the learning rates.
- The learning rate is a hyperparameter in optimization algorithms that controls the step size taken towards the minimum of the loss function during training. Specifically, the learning rate determines the size of the updates made to the model's weights in each iteration of the training process. A larger learning rate leads to larger weight updates, while a smaller learning rate leads to smaller weight updates.
- Iterated over the following values of learning rates for all the three optimizers:
- lr_list = [1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
    - **Adam optimizer**

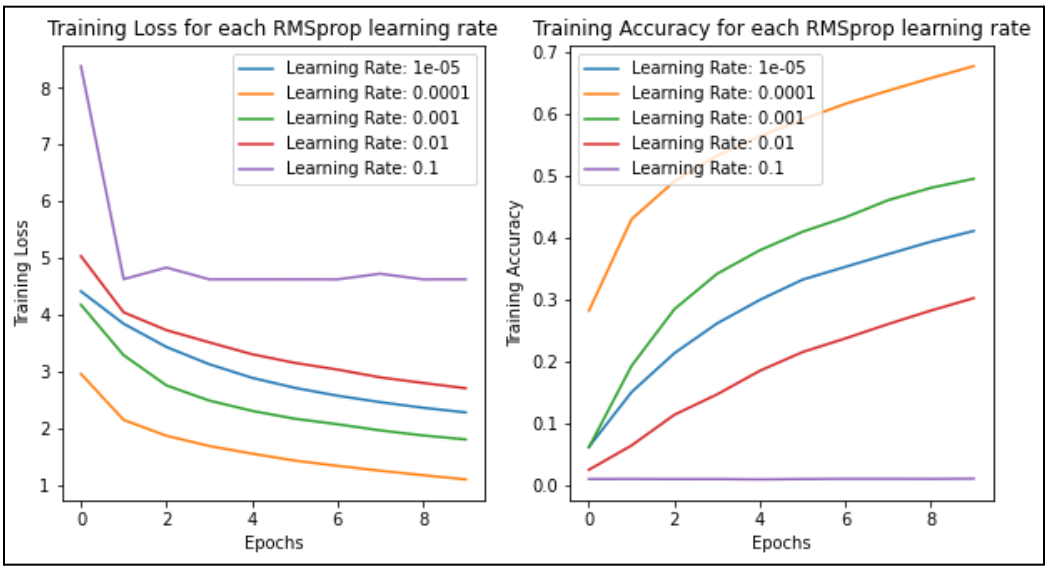


    - We observe that, for lr=0.0001, training accuracy has shot up unexpectedly and the model is performing better than the other cases. Here, it is important to note that the optimal learning rate depends on the specific optimization problem and the model architecture, and may vary from case to case. It is always a good practice to experiment with different learning rates and select the one that works best for your specific problem.

- **AdaGrad optimizer**



- **RMSProp Optimizer**



- In case of RMSProp, we observe that learning rate = 1e-4 is so much better than the other cases. This might be due to the same reasons as in Adam optimizer.

END OF LAB ASSIGNMENT 3