# *TP2*:

## *Presents all obtained the results (graphs, accuracy, running time)*

### *Part I:*

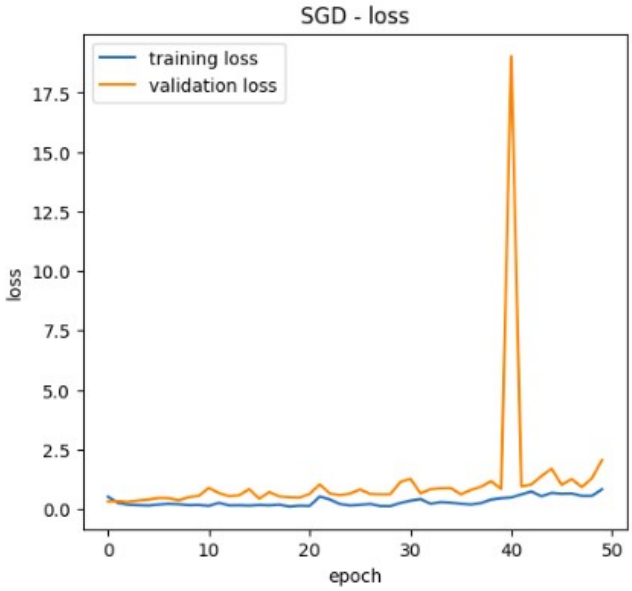
The used algorithms:
- SGD (Stochastic Gradient Descent): updates the model parameters using the gradient of the loss function, the size of the batch is 1
- batch SGD: updates the model parameters using the gradient of the loss function with respect to the entire training dataset, it has stable updates but computationally expensive.
- mini-batch SGD: updates the model parameters using the gradient of the loss function with 64 as batch size, its balances the stability of batch SGD and the efficiency of SGD.
- mini-batch SGD with decay: we use 10e-6 as learning rate, which decreases over time, the decease in learning rate can help in fine-tuning the model as it converges.
- SGD with decay and momentum: 10e-6 as learning rate which decreases over time and momentum to accelerate convergence, it helps in the relevant direction and dampens oscillations, this algorithm leads to faster convergence and reduced oscillations compared to standard SGD
- Adam (Adaptive Moment Estimation): combines the benefits of momentum and RMSProp, it adapts the learning rate for each parameter, with learning rate or 0.001 and batch size of 1 then 32, generally it converges faster than SGD and is less sensitive to hyperparameters.
- RmsProp (Root Mean Square Propagation): adapts the learning rate (0.001) by dividing it by an exponentially decaying average of squared gradients, we use it with batch size of 1 then 32, it’s good for non-stationary objectives and can handle sparse gradients well.

The execution time:

| The algorithm | Batch size | Iterations | Execution time |
|---|---|---|---|
| SGD | 1 | 6400 | 596 (10m) |
| Mini-batch | 64 | 100 | 35 (0.5m) |
| Batch | 6400 | 1 | 13 (0.2m) |
| Mini-batch with decay | 64 | 100 | 37 (0.6m) |
| SGD with decay/momentum | 1 | 6400 | 1081 (18m) |
| Adam with learning rate | 1 | 6400 | 1436 (24m) |
| RmsProp with learning rate | 1 | 6400 | 1488 (25m) |
| Mini-batch | 32 | 200 | 65 (1m) |
| Adam | 32 | 200 | 89 (1.5m) |
| RmsProp | 32 | 200 | 95 (1.6m) |

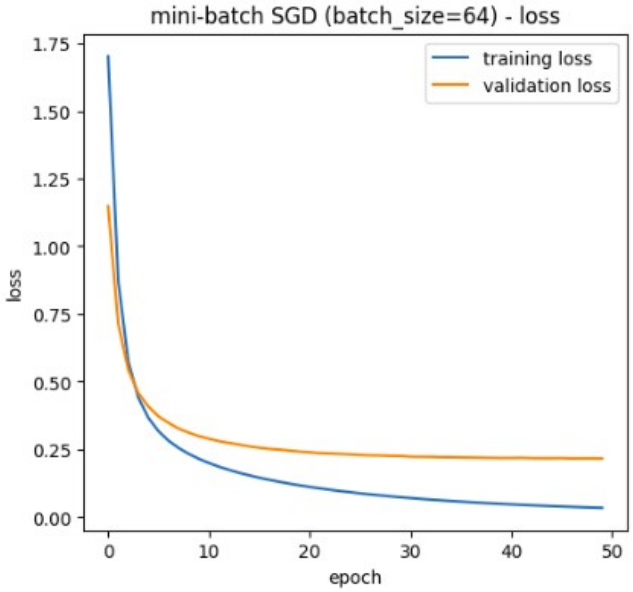
Accuracy and graphs:

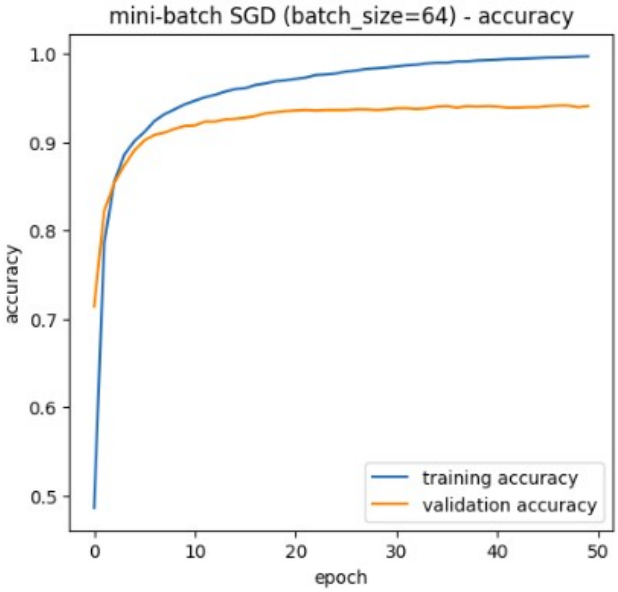
The below graphs show training/validation loss and accuracy with 50 epochs



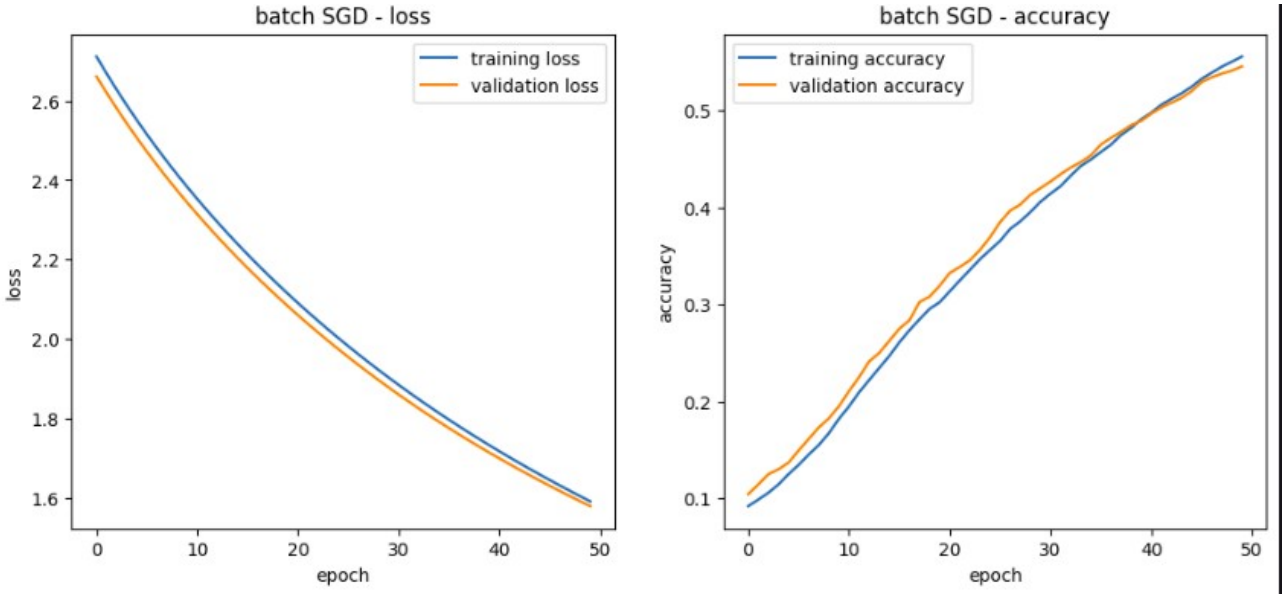


SGD with learning rate 0.01 starts with a loss around 1 but the it decreases to 0 then increases with unstable way, on the validation set we had loss arround 17.5 in the 40th epoch, the accuracy increases from 0.85 to 0.98 then deacreses to 0.7 when we achienve the 50th epoch.

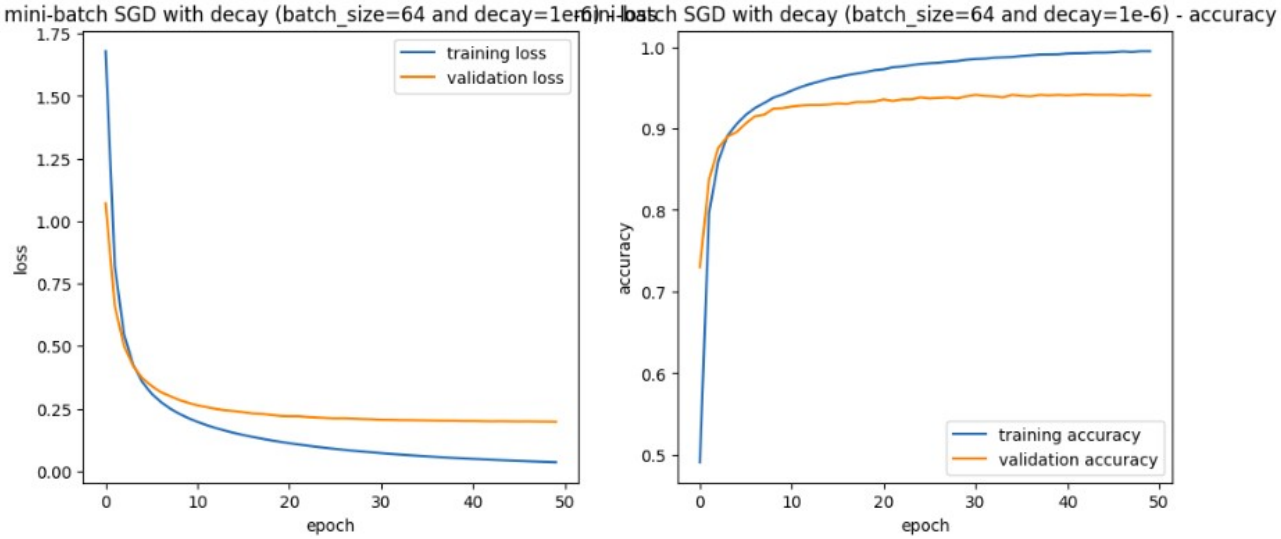


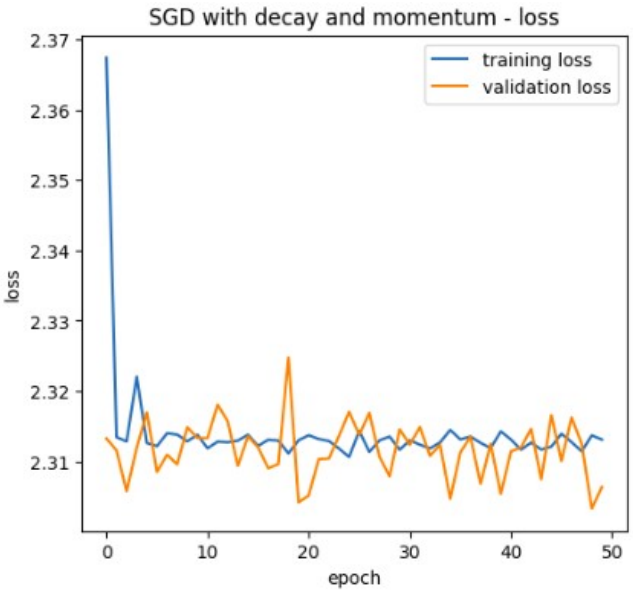

Mini-batch SGD (with batch size 64) has loss starting at 1.75 and going down to 0, the accuracy was from 0.5 to 0.9 on validation set and to ~1 for the training set, we had slower convergence highlighting noise from smaller batches.

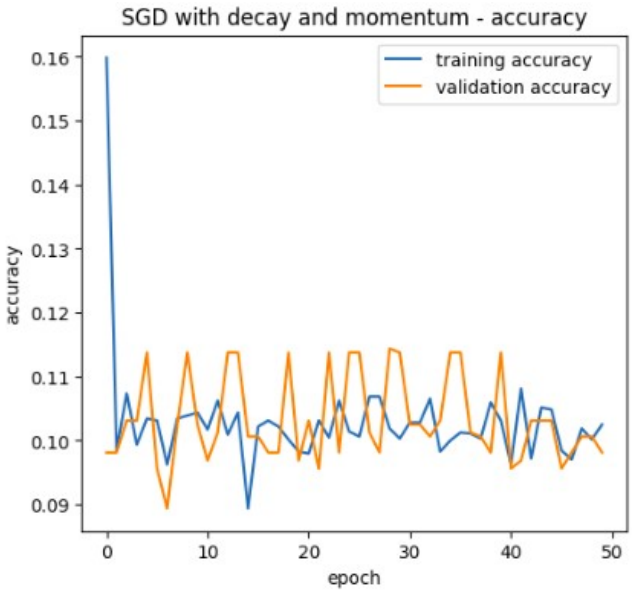



Batch sgd has loss starting from 2.7 decreasing to 1.6, the accuracy increases from 0.1 to ~0.55, it converges faster because we're using all the data in training step which lead to get global results on all the data.

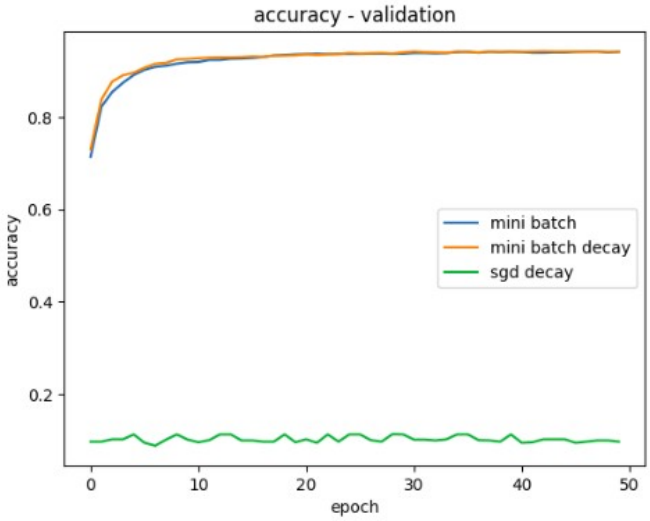


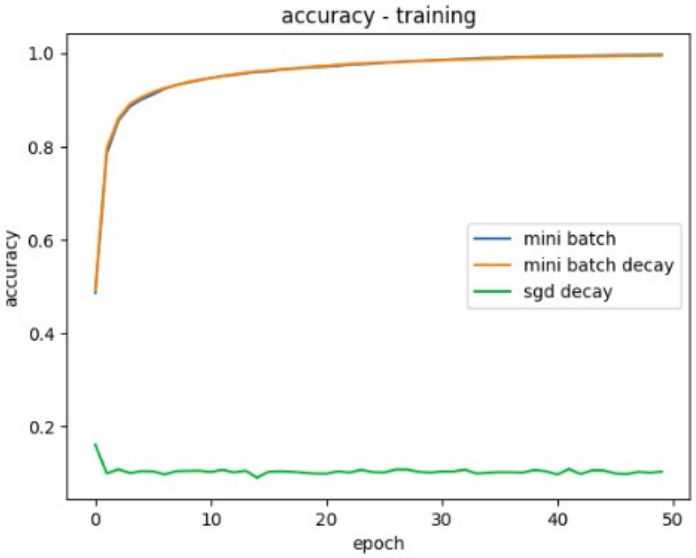

Mini-batch SGD (with batch size 64) with decay=10e-6 reached results as same as mini-batch SGD without decay, which wasn't expected because the decay helps in convergence.



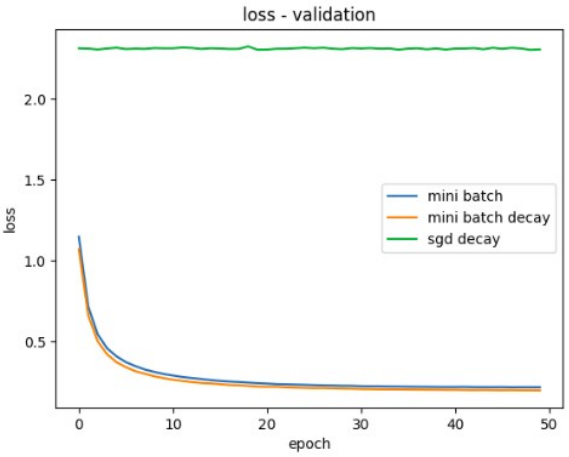



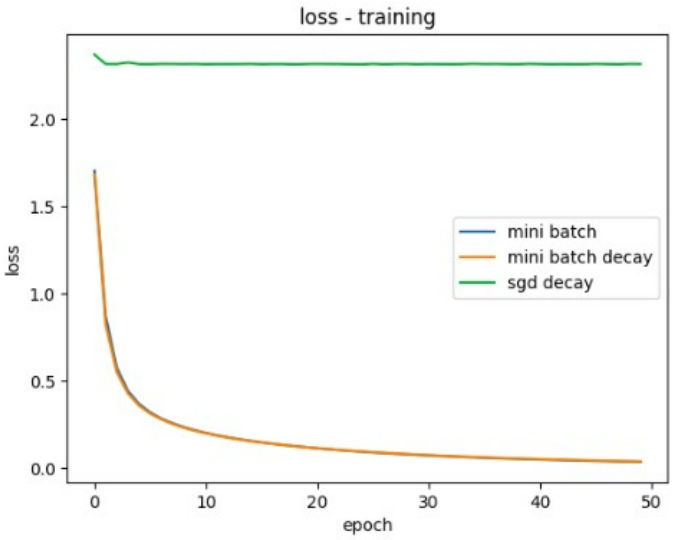

SGD with momentum and decay has loss 2.37~2.31 which is higher comparing to other algorithms, it has lower accuracy 0.09~0.16, also we have no convergence due the processing of each sample alone.



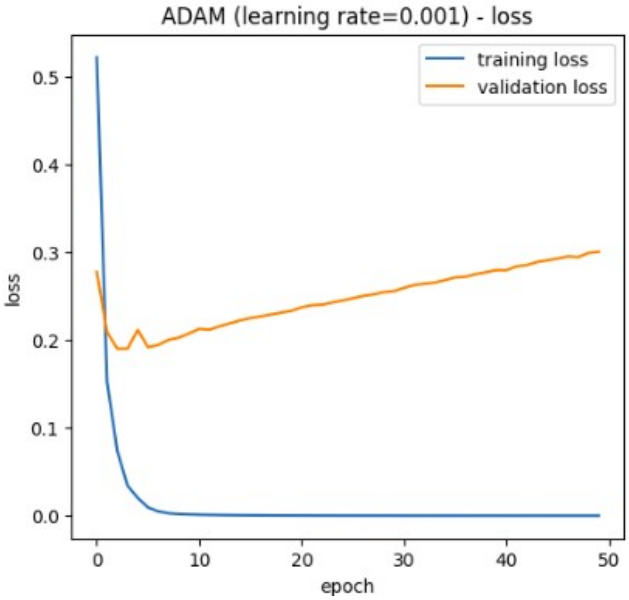



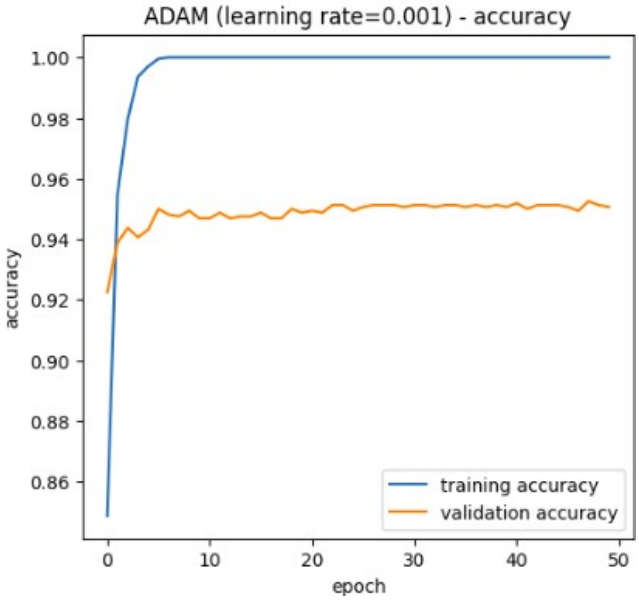

Comparing the accuracy result for the three algorithms SGD with decay, mini-batch SGD with and without decay, we get that both mini-batch algorithms acheive almost same result, 0.7~0.9 on validation set, for the SGD with decay it has values around 0, so it has the lower accuracy, maybe due the processing of each individual so it handles noises and update the weight based on them.



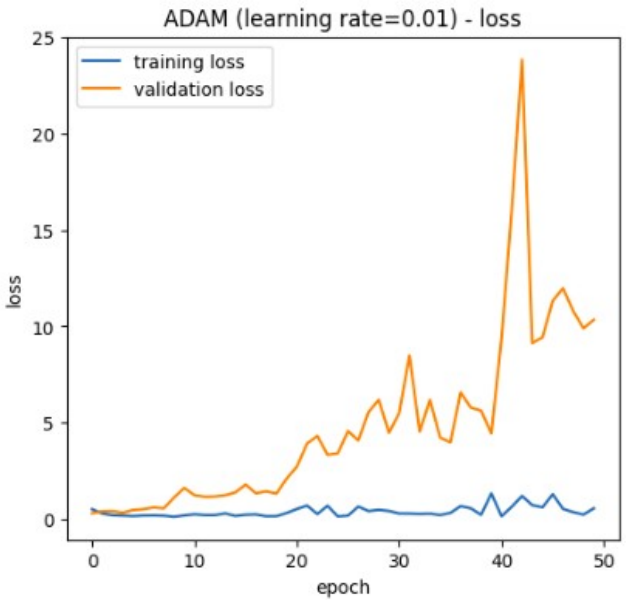



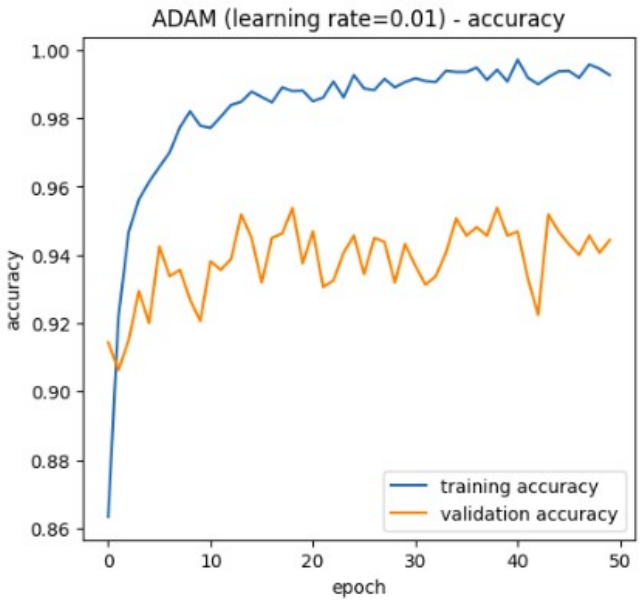

When it comes to the loss, SGD with decay has the higher loss (it's expected because it acheives the lower accuracy), the same note as the previous for mini-batch with/without decay, they have almost the same result, we have the mini-batch with decay algorithm converges faster but with small difference with mini-batch around epoch number 5.



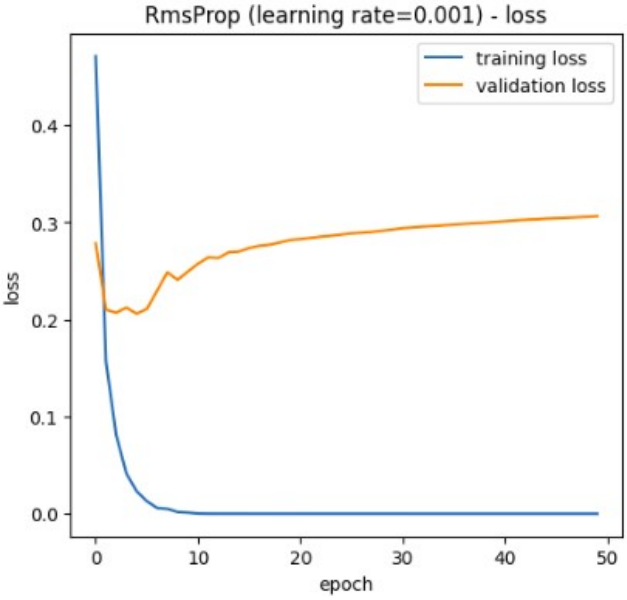



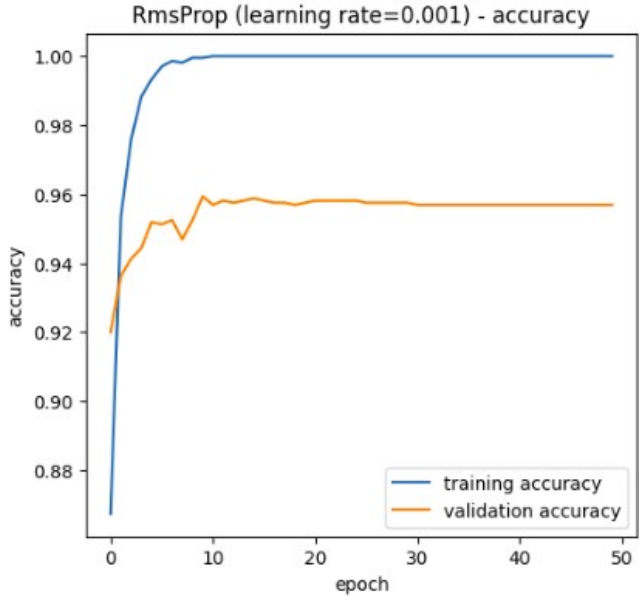

Adam with learning rate of 0.001, its training/validation loss are both going down from 0.5 to ~0 while the accuracy increases from 0.86 to ~1, for the validation set it hasn't as much as good result because the accuracy doesn't pass 0.95 and the loss suddenly inceases which may indicate to overfitting on the training set.



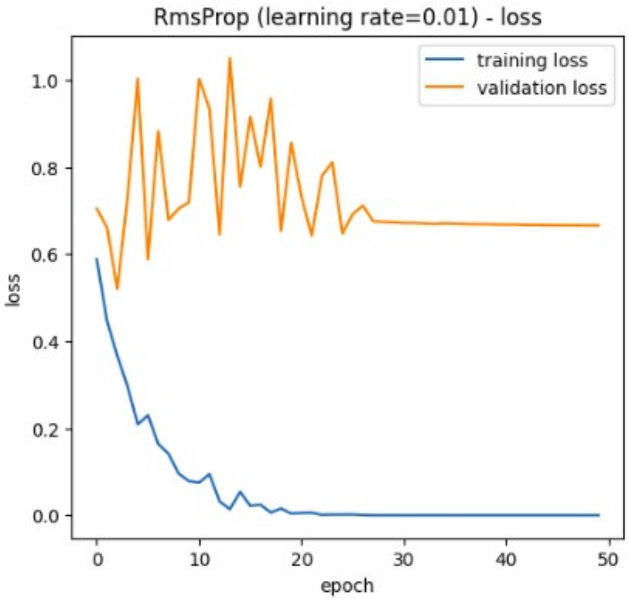



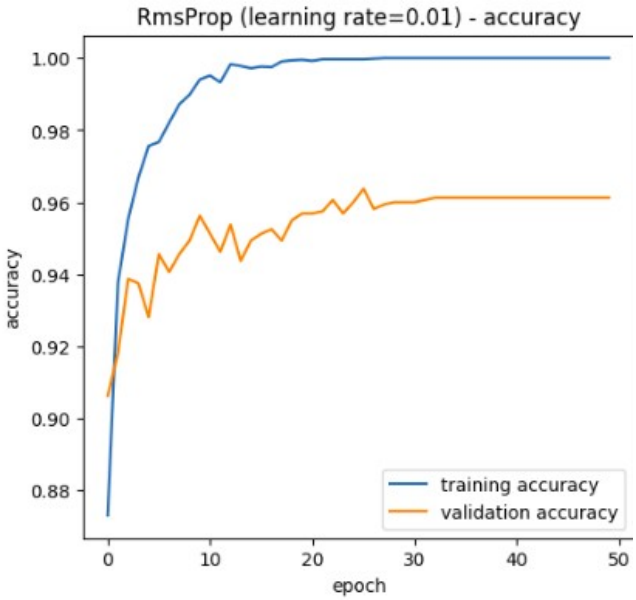

We almost have the same result for Adam with batch size = 32, except that we have oscilliation due the use of mini-batch with clear overfitting on training set where the loss was around 0 but it reaches 25 for the validation set.




RMSProp with learning rate 0.001 shows loss decreasing from around 0.4 to near 0.0 over the 50 epochs with validation loss around 0.25 which suddenly increases, it may indicates to overfitting, the accuracy starts at around 0.88 and goes up to ~1.




RmsProp with batch size = 32 shows clear overfitting, the both loss start around 0.65, the training one decreases until ~0 but the validation one was going up and down from 0.6 to 1 then has stable value 0.7, which indicates to oscillisation problem.

So, comparing all those models, the mini-batch with decay = 10e-6 and mini batch = 64 was the best, it has low execution time and good performance.

***Part II:***