HW 3

Arnab Dey
Student ID: 5563169
Email: dey00011@umn.edu

Solution 1.a

The value of n is 4 and the value of m is also 4. The value of k is 16.

Solution 1.b

The size of $\boldsymbol{W}_{conv}$ is 3×3. The size of $\boldsymbol{W}_{fc}$ is 16×10. The size of $\boldsymbol{b}$ is 1×10.

Solution 2

Summary: Here I have used Tensorflow to build a feed forward neural network. The network is learned using stochastic gradient descent (SGD) algorithm for weight update using a mini batch size of 32. SGD is used without momentum and with a learning rate of 0.01. I have also converted labels to *one-hot* representation to use *categorical crossentropy* as the loss for Tensorflow model. The input features are scaled to $[0, 1]$ as I have seen this improves the performance of the neural network a lot.

Convergence criteria used: I have declared convergence if in 3 consecutive epochs the decrease of loss is less than 0.001 or loss increases for 3 consecutive epochs. The maximum number of epochs has a limit of 100 though I have never seen hitting this limit before convergence is achieved.

Results: Fig. 1 shows the cumulative loss and accuracy with increasing nuber of epochs. From Fig. 1,

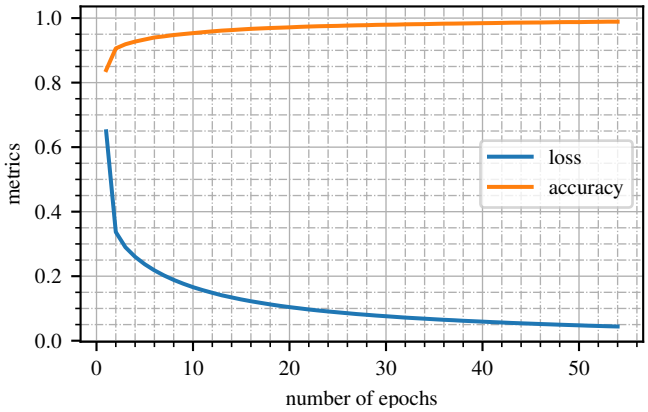


Figure 1: Q2: SGD cumulative training loss and accuracy with different epochs: Feedforward neural net

we can see that as we increase the number of epochs, the network learns better and thus the loss decreases monotonically and the accuracy increases. Table 1 shows the training loss and accuracy after convergence and test loss and accuracy.

Table 1: Q2: Train (after convergence) and test loss and accuracy

	Loss	Accuracy(%)
Train	0.044	98.88
Test	0.077	97.60

Solution 3.a

Summary: Here Tensorflow is used to build a convolutional neural network as asked in the homework. The network is learned using stochastic gradient descent (SGD) as the optimizer to update the weights without momentum and with a learning rate of 0.01. The parameters are tuned after iterating through multiple runs with different learning rates. The value which produced best accuracy is used. I have used batch normalization layer after convolution layer to normalize the output of the convolution layer. Batch normalization is actually standardization procedure over mini batches where the convolved ensemble feature mean is subtracted from each feature and divided by the standard deviation of corresponding feature. It helps in improving the accuracy. Once the output is normalized, I have used the ReLU activation layer. Another alternative to this could be to directly pass the *activation='relu'* argument to convolution layer in Tensorflow Keras.

For SGD, I have used learning rate of 0.01 with no momentum. For ADAGRAD, I have used learning rate of 0.1 with initial accumulator value of 0.01. For ADAM, I have used learning rate of 0.001 and ρ_1 and ρ_2 values of 0.9 and 0.999 respectively. I ran the code multiple times with different set of hyperparameters and the chose the values which produced best accuracy. For ADAM, it seemed to me that the default values gave me the best result. I have also converted labels to *one-hot* representation to use *categorical crossentropy* as the loss for Tensorflow model. The input features are scaled to $[0, 1]$ as I have seen this improves the performance of the neural network a lot.

Convergence criteria used: I have declared convergence if in 3 consecutive epochs the decrease of loss is less than 0.001 or loss increases for 3 consecutive epochs. The maximum number of epochs has a limit of 100 though I have never seen hitting this limit before convergence is achieved.

Results: Fig. 2 shows the cumulative loss and accuracy with increasing nuber of epochs. From Fig. 2,

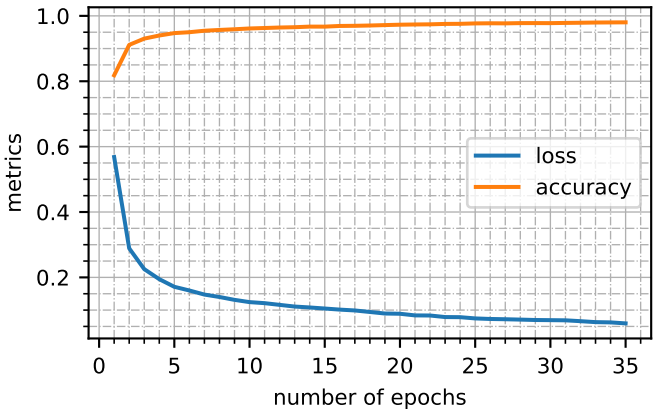


Figure 2: Q3: SGD cumulative training loss and accuracy with different epochs: CNN

we can see that as we increase the number of epochs, the network learns better and thus the loss decreases monotonically and the accuracy increases. Table 2 shows the training loss and accuracy after convergence and test loss and accuracy. If we compare the results shown in Table 1 with that of Table 2, we can see

Table 2: Q3: Train (after convergence) and test loss and accuracy

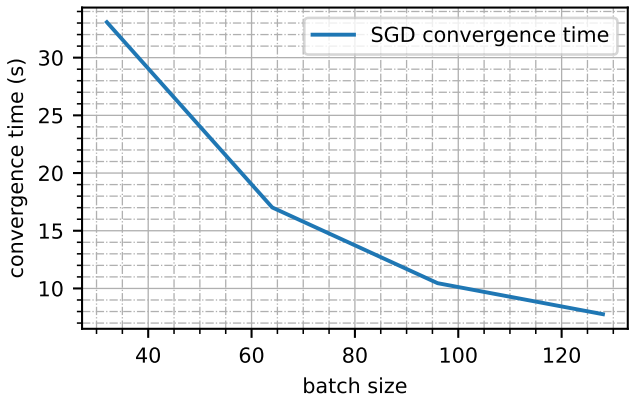
	Loss	Accuracy(%)
Train	0.060	98.04
Test	0.032	98.93

that though feed forward network gave a slightly better accuracy for training data, CNN gave better test accuracy.

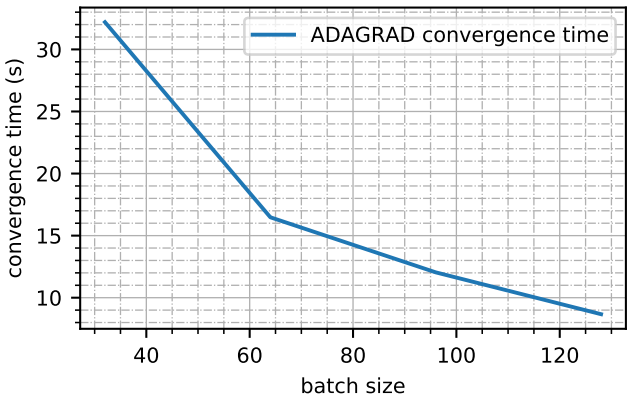
Solution 3.b

Convergence time calculation: I have calculated the convergence time in seconds using the Tensorflow callbacks which is essentially the time between the train end and train begin callbacks. I have measured the time using *time.time()* module of Python. Also, I have averaged the time over 10 independent runs to reduce the variability.

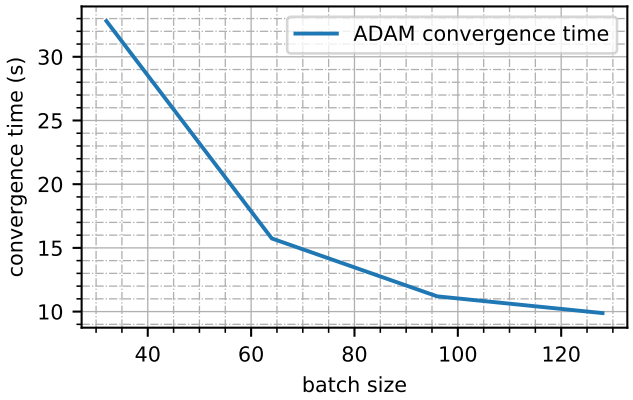
Results: Fig. 3a, Fig. 3b and Fig. 3c show the evolution of convergence time, averaged over 10 runs, with increasing batch size.



(a) SGD: Convergence time, averaged over 10 runs, with increasing batch size



(b) ADAGRAD: Convergence time, averaged over 10 runs, with increasing batch size



(c) ADAM: Convergence time, averaged over 10 runs, with increasing batch size

Figure 3: Q3: CNN: Convergence time, averaged over 10 runs, with SGD, ADAGRAD and ADAM optimizer with increasing batch size

From Fig. 3a, Fig. 3b, Fig. 3c and Fig. 4, we can see that as we increase the batch size the time taken to converge in seconds decreases which means learning becomes faster as we increase the batch size. However, this is opposite to the theoretical understanding we have on the effect of increasing batch size as we know if we increase the number of batch size, gradient computation time increases and thus learning time becomes slower. Thus, I believe, in this case, the CPU instructions or parallelization is playing a major role in deciding the convergence time. I have used *time.time()* module of Python to compute the convergence time. I have also tested the code on Google Colab with GPU. Both CPU and GPU generated same trend of decreasing convergence time with increasing batch size. Therefore, I believe, here the convergence time (in seconds) is primarily governed by how the code execution is being handled by the CPU or GPU. I cross checked it with Professor and we agreed that it could be a possible reason of seeing this trend. Another possibility could be to plot iteration number instead of time in seconds on y-axis, however, we agreed to exclude that detail for this homework.

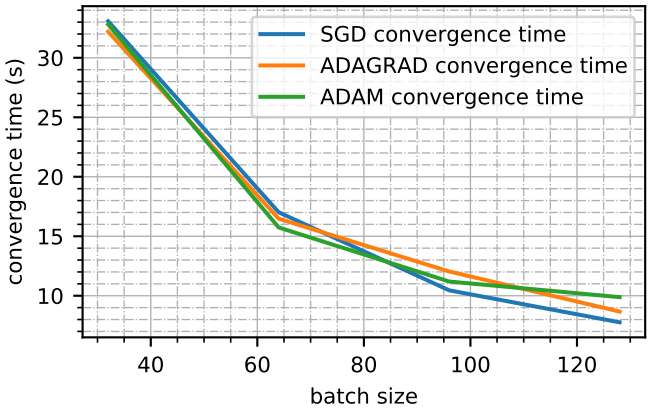
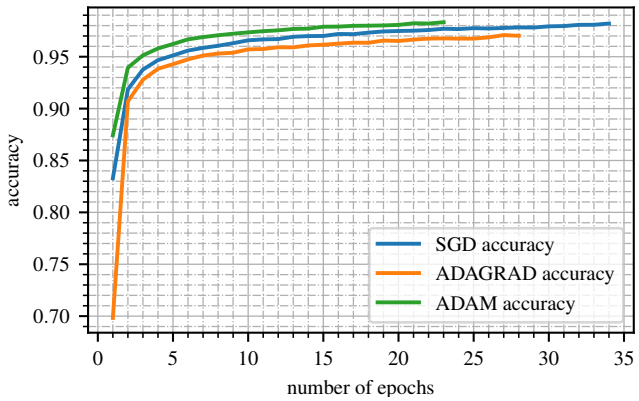
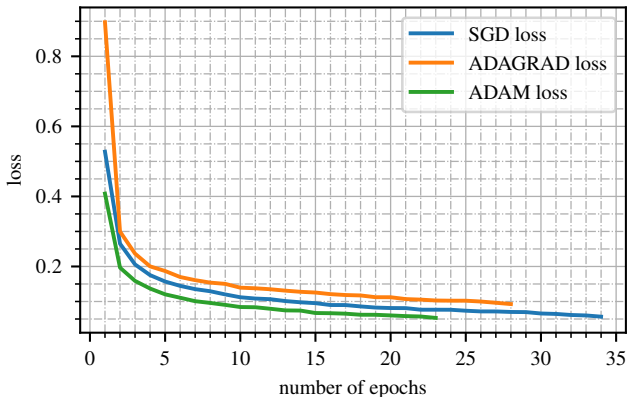


Figure 4: Q3: CNN: Convergence time comparison, averaged over 10 runs, with SGD, ADAGRAD and ADAM optimizer with increasing batch size

Details gathered from a single run: Fig. 5 shows the loss and accuracy for SGD, ADAGRAD and ADAM optimizers for a single run with batch size of 32. The figures reveal that of all these three optimizers, ADAM performs best. Fig. 6 shows the loss and accuracy for SGD, ADAGRAD and ADAM optimizers for



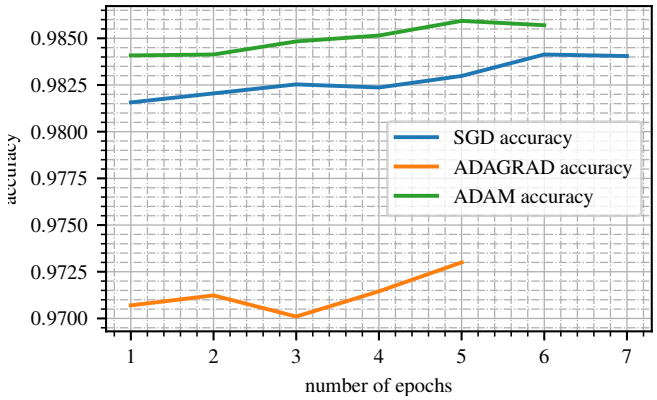
(a) Q3: CNN: Accuracy of different optimizer with 32 batch size



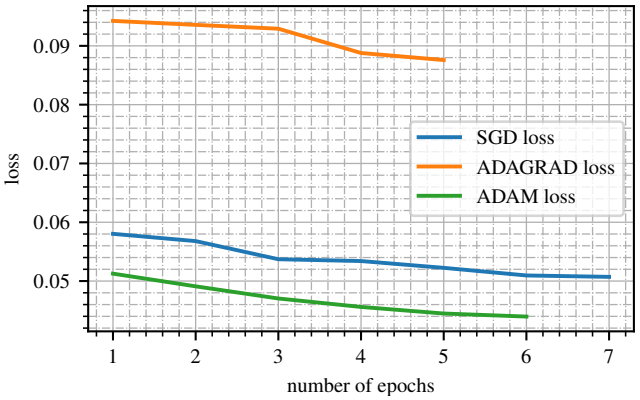
(b) Q3: CNN: loss of different optimizer with 32 batch size

Figure 5: Q3: CNN: Accuracy and loss with SGD, ADAGRAD and ADAM optimizer with increasing epoch with batch size of 32

a single run with batch size of 64. The figures reveal that of all these three optimizers, ADAM performs best. Fig. 7 shows the loss and accuracy for SGD, ADAGRAD and ADAM optimizers for a single run with batch size of 96. The figures reveal that of all these three optimizers, ADAM performs best. Fig. 8 shows the loss and accuracy for SGD, ADAGRAD and ADAM optimizers for a single run with batch size of 128. The figures reveal that of all these three optimizers, ADAM performs best.

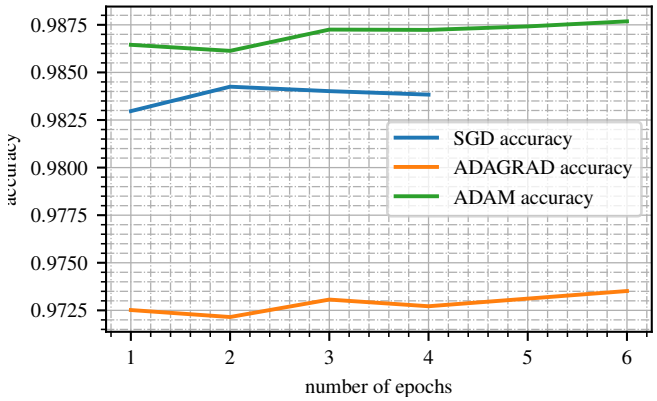


(a) Q3: CNN: Accuracy of different optimizer with 64 batch size

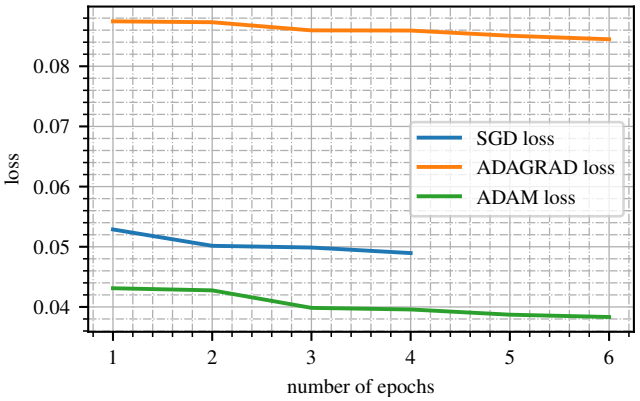


(b) Q3: CNN: loss of different optimizer with 64 batch size

Figure 6: Q3: CNN: Accuracy and loss with SGD, ADAGRAD and ADAM optimizer with increasing epoch with batch size of 64

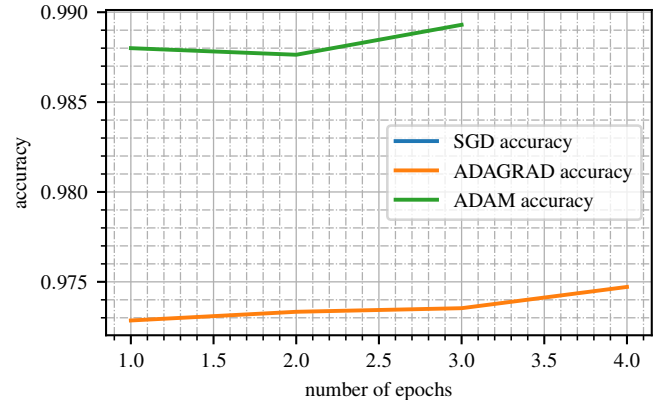


(a) Q3: CNN: Accuracy of different optimizer with 96 batch size

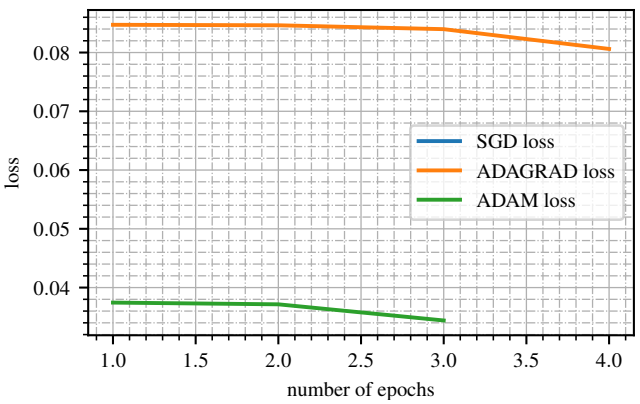


(b) Q3: CNN: loss of different optimizer with 96 batch size

Figure 7: Q3: CNN: Accuracy and loss with SGD, ADAGRAD and ADAM optimizer with increasing epoch with batch size of 96



(a) Q3: CNN: Accuracy of different optimizer with 128 batch size



(b) Q3: CNN: loss of different optimizer with 128 batch size

Figure 8: Q3: CNN: Accuracy and loss with SGD, ADAGRAD and ADAM optimizer with increasing epoch with batch size of 128