Mohamed ABDUL GAFOOR

- We ran all the experiments in the **Google Colab** and the configuration is as follow. We have used the Google **GPU not CPU!**

CPU	GPU	TPU
Intel Xeon Processor with two cores @ 2.30 GHz and 13 GB RAM	Up to Tesla K80 with 12 GB of GDDR5 VRAM, Intel Xeon Processor with two cores @ 2.20 GHz and 13 GB RAM	Cloud TPU with 180 teraflops of computation, Intel Xeon Processor with two cores @ 2.30 GHz and 13 GB RAM

- We found average number of reading out of 5 calculations.

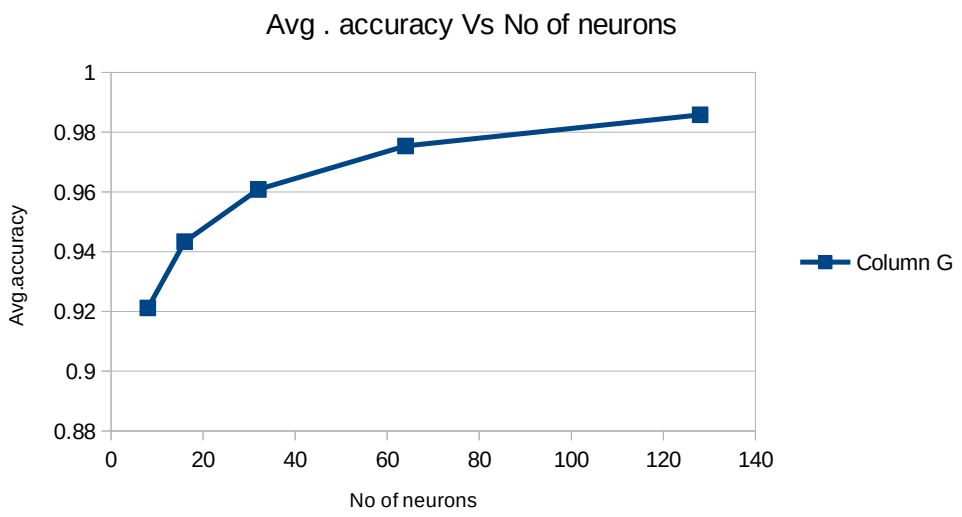
Exercise 1: Modify the number of neurons on the hidden layer as specified in Table 1. What can be observed regarding the system accuracy? How about the convergence time necessary for the system to train a model?

A screen shot from an excel sheet to show the average of 5 readings.

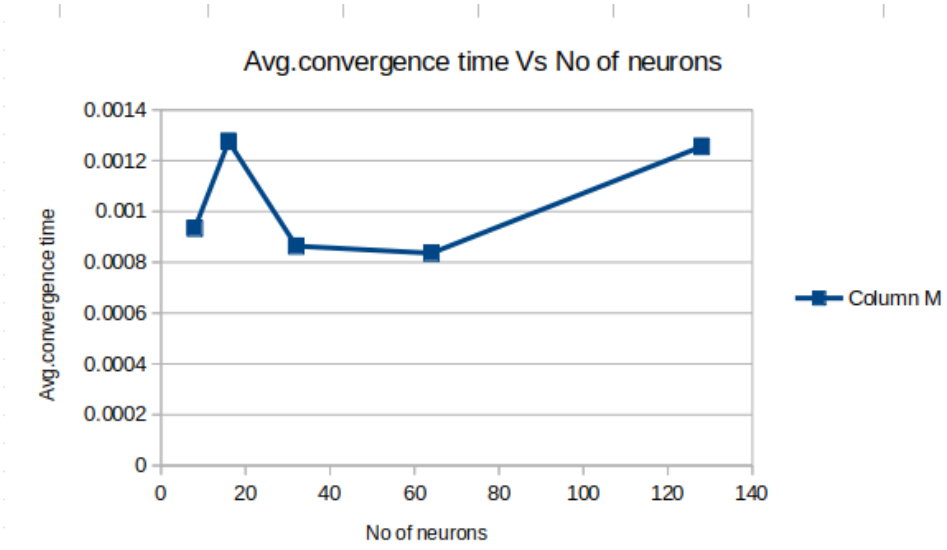
No of Neurons	Accuracy_1	Accuracy_2	Accuracy_3	Accuracy_4	Accuracy_5	Avg. Accuracy	time_1	Time_2	Time_3	Time_4	Time_5	Avg. time
8	0.9225	0.9211	0.9177	0.9231	0.9213	0.92114	0.00016	0.00048	0.00127	0.00236	0.0004	0.000934
16	0.943	0.9431	0.9447	0.9414	0.9445	0.94334	0.00189	0.00075	0.00064	0.00026	0.00284	0.001276
32	0.9622	0.9586	0.9612	0.9627	0.9594	0.96082	0.00212	0.00035	0.0008	0.00042	0.00063	0.000864
64	0.9748	0.9751	0.9761	0.9757	0.9752	0.97538	0.00022	0.00026	0.00231	0.0002	0.00119	0.000836
128	0.986	0.9861	0.9859	0.9846	0.9864	0.9858	0.00146	0.00062	0.00209	0.00049	0.00162	0.001256

The table and the following figure below show how the accuracy of the network increases with the number of neurons in the hidden layer.

No of neurons	8	16	32	64	128
Average system accuracy	0.92114	0.94334	0.96082	0.97538	0.9858
Avg. convergence time	0.000934	0.001276	0.000864	0.000836	0.001256



The convergence time is fluctuating throughout the experiment. The below figure shows the variation of the convergence time with number of neurons. We can observe that even though, it decreases at the beginning, it started to increases after 64 neurons in the hidden layer.



Mohamed ABDUL GAFOOR

Exercise 2: Modify the batch size used to train the model as specified in Table 2. Select a value of 8 neurons for the hidden layer. What can be observed regarding the system accuracy?

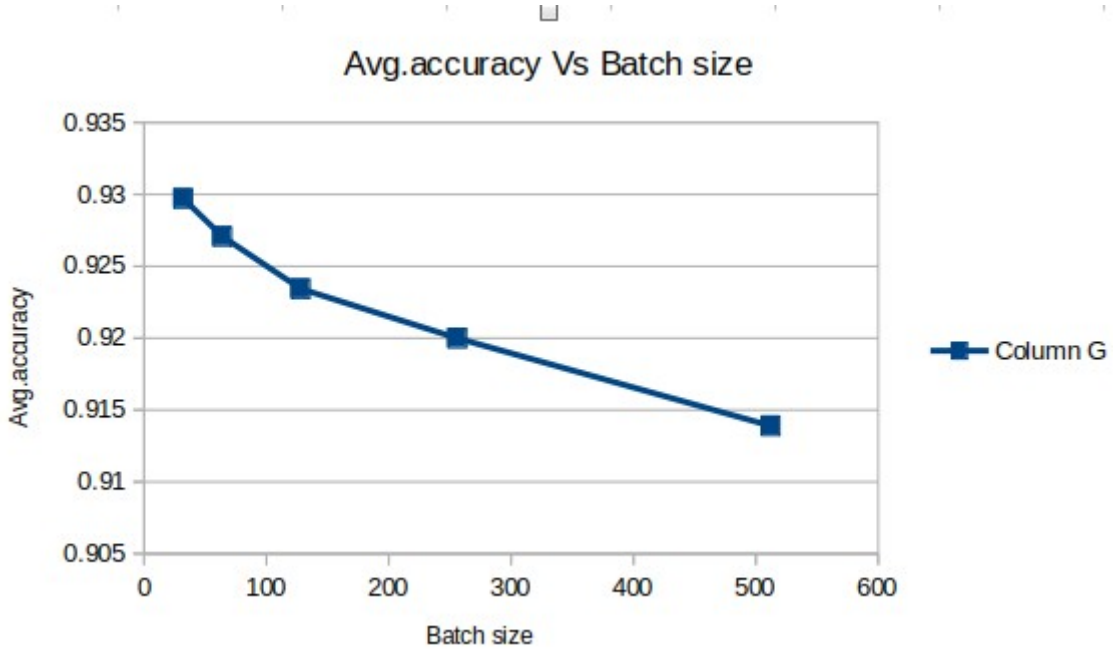
The screen shot from the excel. (8 neurons in the hidden layer).

Batch size	Accuracy_1	Accuracy_2	Accuracy_3	Accuracy_4	Accuracy_5	Avg.accuracy
32	0.9281	0.9283	0.9322	0.9301	0.9299	0.92972
64	0.9231	0.9289	0.9297	0.9277	0.926	0.92708
128	0.9217	0.9233	0.9241	0.9238	0.9243	0.92344
256	0.9195	0.9201	0.9191	0.9236	0.9177	0.92
512	0.9142	0.9152	0.9113	0.9138	0.915	0.9139

This experiment was performed for a fixed neuron size

Batch size	32	64	128	256	512
System accuracy	0.92972	0.92708	0.92344	0.92	0.9139

The following figure shows the accuracy decreases if we increase the batch size.



Mohamed ABDUL GAFOOR

Exercise 3: Modify the metric used to determine the system performance from accuracy to mean square error (mse). Has this parameter any influence over the training process?

We have changed the from “accuracy” to “mse”;

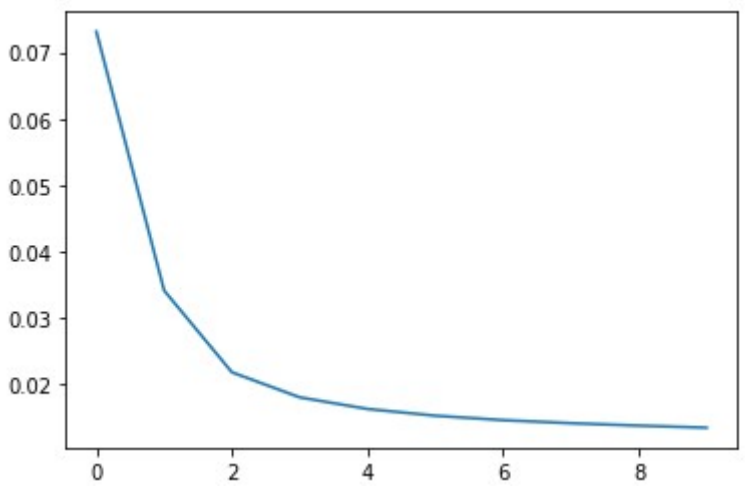
```
#TODO - Application 1 - Step 6 - Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['mse'])
```

We performed the calculation for 8 neurons in the hidden layer, batch size of 512 and for 10 epoch. The following figure shows how the mean square error decreases for each epoch. We didn't find any significant changes. For example the convergence time for **metrics = [“accuracy”]** is more or less equal to the **metrics = [“mse”]** for identical configuration.

The screen shot of the excel sheet to find average.

	mse	convergence time
Trial_1	0.0137	0.00036
Trial_2	0.0133	0.00099
Trial_3	0.0132	4.00E-05
Trial_4	0.013	0.00203
Trial_5	0.0133	0.00208
Average	0.0133	0.0011

The below figure shows how the “mse” decreases with the neuron number.



Mohamed ABDUL GAFOOR

Exercise 4: Using the default parameters specified at Application 1 (8 neurons in hidden layer, 10 epochs and a batch size of 200 images), train the model and save the associated weights in a file (`model.save_weights`), so it can be load anytime to perform further tests.

The model has been saved like below;

```
#model save
model.save_weights('my_model_weights.h5')
```

Exercise 5: Write a novel Pyhton script that loads the saved weights (`model.load_weights`) at Exercise 5 and make a prediction on the first 5 images of the testing dataset (`mnist.test_images()`).

It was implementation as below;

```
def load_fun(X_train, Y_train, X_test, Y_test):

  #TODO - Application 1 - Step 2 - Transform the images to 1D vectors of floats (28x28 pixels  to  784 elements)
  num_pixels = X_train.shape[1] * X_train.shape[2]
  X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
  X_test = X_test.reshape((X_test.shape[0], num_pixels)).astype('float32')

  #TODO - Application 1 - Step 3 - Normalize the input values
  X_train = X_train / 255
  X_test = X_test / 255

  #TODO - Application 1 - Step 4 - Transform the classes labels into a binary matrix
  Y_train = np_utils.to_categorical(Y_train)
  Y_test = np_utils.to_categorical(Y_test)
  num_classes = Y_test.shape[1]

  model = baseline_model(num_pixels, num_classes)

  model.load_weights("my_model_weights.h5")

  Y_pred = model.predict(X_test)
  Y_pred = np.argmax(Y_pred, axis=1)

  return Y_pred
```

If we load the function , we got the results as follow;

```
Y_pred = load_fun(X_train, Y_train, X_test, Y_test)
```

```
print("Y_pred: ", Y_pred)
print("Y_test: ", Y_test)
```

```
Y_pred:  [7 2 1 ... 4 5 6]
Y_test:  [7 2 1 ... 4 5 6]
```

The below figure show the first 6 images from test data-set, which is exactly matching with the predicted value.

```
%matplotlib inline
fig=plt.figure(figsize=(8, 8))
col = 3
row = 2
for i in range(1, col*row +1):
    img = X_test[i-1]
    fig.add_subplot(row, col, i)
    plt.imshow(img)
plt.show()
```