ASSIGNMENT 3 REPORT – MANIKANDAN LALITPRASAD (mxl190001)

Dataset 1- SGEMM GPU Kernel Performance

This data set measures the running time of different parameter combinations of parameterizable SGEMM GPU kernel. The dataset contains 241600 records of these feasible combinations. For each tested combination, 4 runs were performed and recorded (measured in milliseconds)

There are 14 parameters, the first 10 are ordinal and can only take up to 4 different powers of two values, and the 4 last variables are binary.

Attribute Information:

Independent variables:
1-2. MWG, NWG: per-matrix 2D tiling at workgroup level: {16, 32, 64, 128} (integer)
3. KWG: inner dimension of 2D tiling at workgroup level: {16, 32} (integer)
4-5. MDIMC, NDIMC: local workgroup size: {8, 16, 32} (integer)
6-7. MDIMA, NDIMB: local memory shape: {8, 16, 32} (integer)
8. KWI: kernel loop unrolling factor: {2, 8} (integer)
9-10. VWM, VWN: per-matrix vector widths for loading and storing: {1, 2, 4, 8} (integer)
11-12. STRM, STRN: enable stride for accessing off-chip memory within a single thread: {0, 1} (categorical)
13-14. SA, SB: per-matrix manual caching of the 2D workgroup tile: {0, 1} (categorical)

Output:
15-18. Run1, Run2, Run3, Run4: performance times in milliseconds for 4 independent runs using the same parameters. They range between 13.25 and 3397.08.
We take the average of these four runs as our target variable and added as a column in our data frame

OBJECTIVE:
To implement two learning algorithms using this dataset
1. Artificial Neural Network (ANN)
2. K Nearest Neighbors

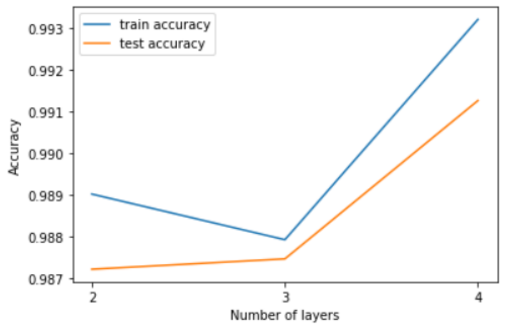
Algorithm1 - Artificial Neural Networks:

For neural networks, we will use Keras API on top of Tensor flow as this seems the fastest in terms of computing speed.
For all the experimentations we will be using GridSearchCV to pick the best parameter among a group of parameters. We will be doing the following experimentations:
- Number of layers
- Number of nodes
- Activation functions
- Optimizers
- Dropout Rates

Experimentation with number of layers:

Initially, we begin experimenting with the number of hidden layers from 2 to 4. The number of neurons is selected randomly with the following combinations.

- 2 hidden layers: 50-30
- 3 hidden layers: 50-40-30
- 4 hidden layers: 50-40-30-20



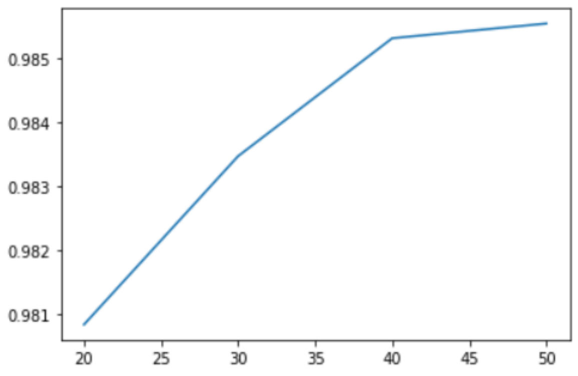
We can see that the training accuracies keep increasing and the test accuracy keeps increasing when the number of hidden layers is 4. Thus, we choose 3 hidden layers for this experiment.

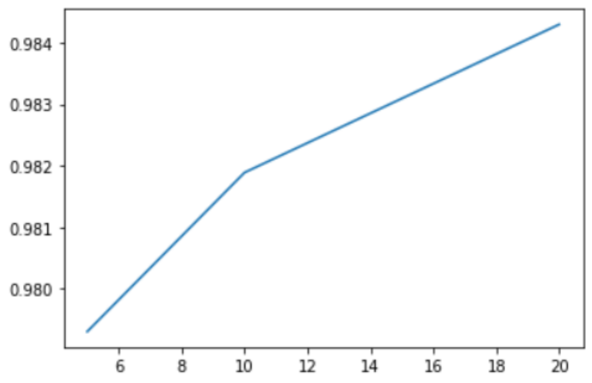
Training accuracy: 98.79%

Test accuracy: 98.74%

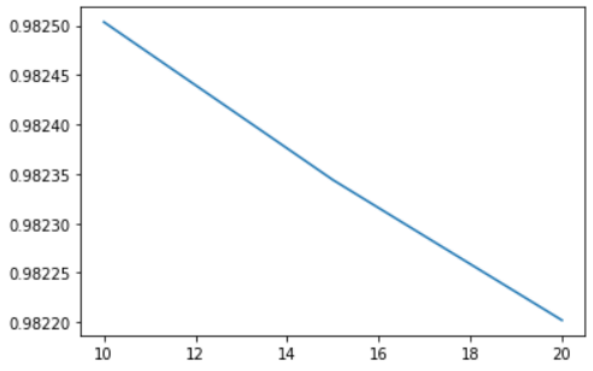
Experimentation with number of neurons:

We try different combination of neurons in each of the 3 different hidden layers.





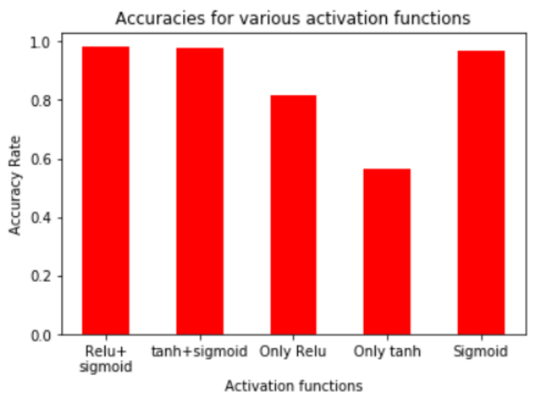
- For the first hidden layer we experiment with neurons from 20,30,40,50. We see that we get highest accuracy when neurons=50 at 98.55%. Thus we fix neurons=50 for the first layer.
- Next, we change neurons as 5, 10, 20 for the second layer. We see that we can achieve maximum accuracy when neurons=20 at 98.42%



- Finally, we change neurons as 10, 15, 20 in the last hidden layer. We achieve maximum accuracy when neurons=10 at 98.25%.

Thus, the best combination is **50-20-10**

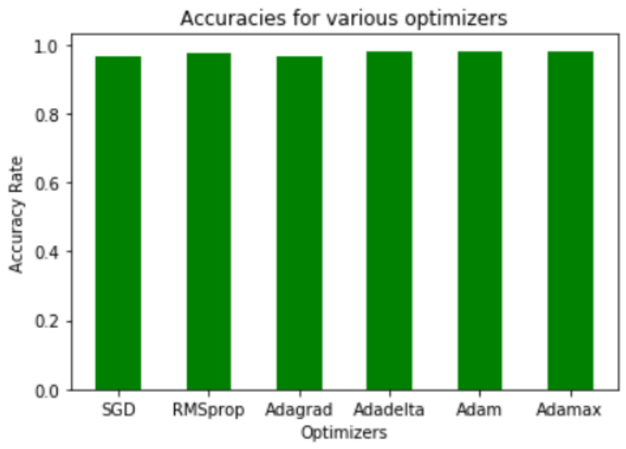
Experimentation with activation functions:



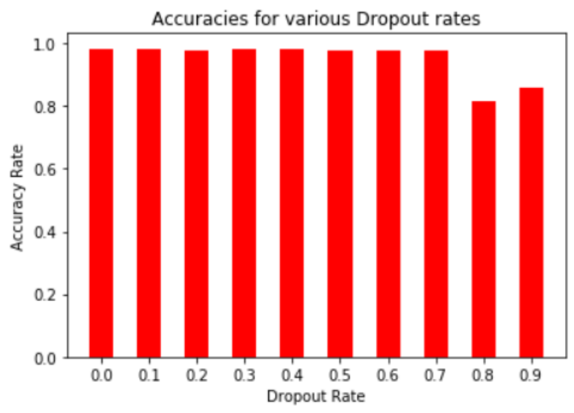
The different activation functions we use are **tanh, sigmoid and relu** as all these work very well with binary classifications. We normally go with sigmoid for the output layer but we also check for the output layer with tanh and relu functions. We can see that **relu for the hidden layers and sigmoid for the output layer performs best of all with 96.78% accuracy**.

Experimentation with optimization functions:

We experiment with different optimization functions such as **SGD, RMSprop, Adagrad, Adadelta, Adam and Adamax**. All the optimizers perform almost similar. But the highest accuracy is achieved by **Adam at 98.21%.**



Experimentation with Dropout rates:



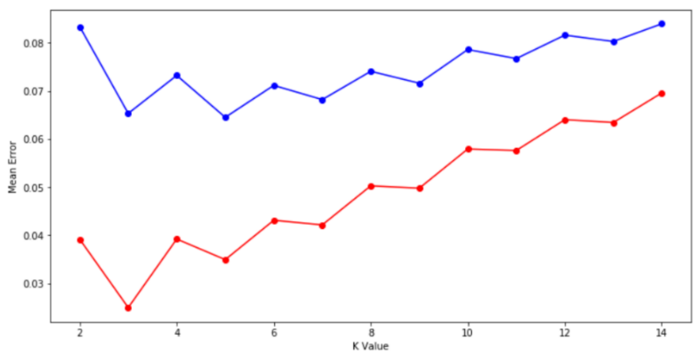
We do not want to over fit the data. Hence dropout is an option to reduce overfitting and hence we try dropout rates from 0.0 to 0.9. We can see that accuracies are almost similar with minute differences. But the best accuracy is achieved when **dropout=0.0 at 98.26%**

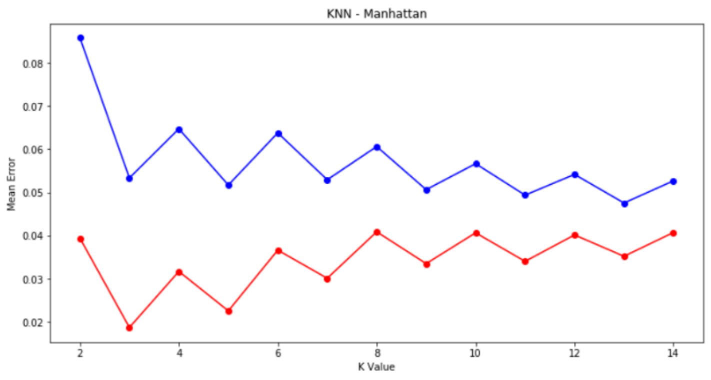
Finally, after choosing all the optimum hyper parameters for this model we train the model and test it with our test data. We get the below accuracies.

Train accuracy: 98.37%
Test accuracy: 98.27%

Algorithm 2: K Nearest Neighbors (KNN):
We use Sci-Kit package for KNN
Experimentation with distance metrics and number of neighbors:
We are experimenting with Euclidean and Manhattan metrics as these are the widely used metrics. The number of neighbors is experimented from 2 to 15.

Though both give similar performance the Manhattan metric gives slightly better accuracy for all the neighbor values. Also, we can see that after K=8, the model seems to generalize well for the test data as well. Till K=7, we can see a clear evidence of over fitting as training error is low and test error is high. Hence, we choose **K=8 and Manhattan distance** as our parameter. We achieve maximum accuracy of **93.93%.**





Comparison between the 2 algorithms:

For this dataset, Neural Networks gives 98.27% accuracy and KNN gives 93.93%. Neural network performs well for this dataset because the choice of neurons and number of hidden layers might be good to handle the data. Hence ANN is better equipped to handle more features than KNN.

Comparison of all algorithms:
In the previous assignment, the best accuracy was given by SVM RBF Kernel with 90%. But KNN beats the performance of SVM. The reason for this is because KNN is not model based. It does not lose any detail and compares every training sample to give the prediction. Hence testing performance will be good with KNN. Thus, we conclude that KNN gives the best accuracy on this dataset.

Ranking of the algorithms:
What I could have done?

- The performance of neural networks is baffling me as it expected it to do the best of all. I could have used lesser number of neurons like 10 -15 with just one or two hidden layers.
- I could have used Grid search CV with a combination of neurons and hidden layers instead of choosing neurons after fixing hidden layers. Maybe this would have resulted in a better accuracy.
- Feature selection could have been done to select the 10 most important features as this would have helped to increase the accuracy of KNN to a large extent. Also, the accuracy for ANN might have also improved.

Dataset 2- Australia Weather

The objective is to predict whether rain will come tomorrow or not based on the location's weather conditions in Australia. The dataset provides us with various variables such as Wind speed, humidity, temperature, pressure etc. Occurrence of rain is indicated by 1 and non-occurrence is indicated by 0. I have chosen all the continuous variables for my analysis.

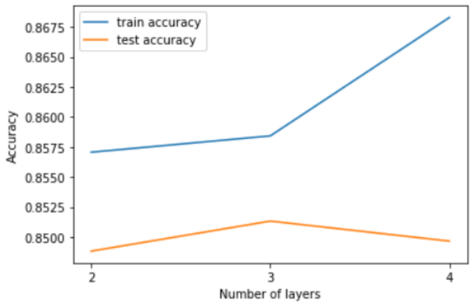
Independent Variables: Mintemp, Maxtemp, Rainfall, WindGustSpeed, WindSpeed9am, WindSpeed3pm, Humidity9am, Humidity3pm, Pressure9am, Pressure3pm, Cloud9am, Cloud3pm, Temp9am, Temp3pm

Dependent variable: Rain
I did random sampling of 20000 observations because the original sample contains 1.4 lakhs. 20000 is a good sample for training the model. Min-max normalization is done on the selected features.

Algorithm 1 - Artificial Neural Networks:
Experimentation with number of layers:
Similar to the previous dataset, initially, we begin be experimenting with the number of hidden layers from 2 to 4. The number of neurons are selected randomly with the following combinations.

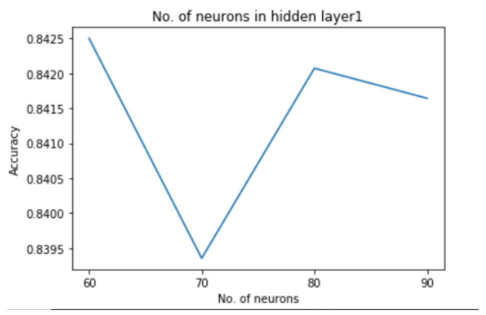
- 2 hidden layers: 50-40
- 3 hidden layers: 50-40-20
- 4 hidden layers: 70-50-40-30

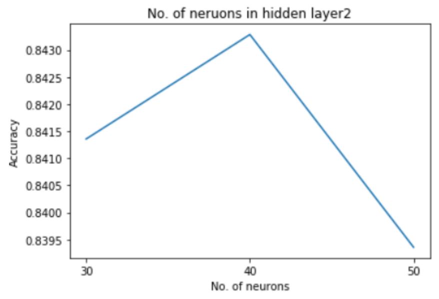
We plot the test and train accuracies to understand the variations in the errors.
We can see that the test accuracy dips when number of layers is 4. The training accuracy is still high which means the model has over fit the data. Thus **3 hidden layers** seems to be a good bias variance tradeoff and we choose that.

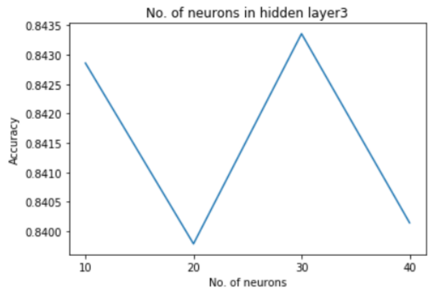


Training accuracy: 86%
Test accuracy: 85.1%

Experimentation with number of neurons:
We try different combination of neurons in each of the 3 different hidden layers.
For the first hidden layer we experiment with neurons from 60-90. We see that we get highest accuracy when neurons=60 at 84.25%. Thus we fix neurons=60 for the first layer.



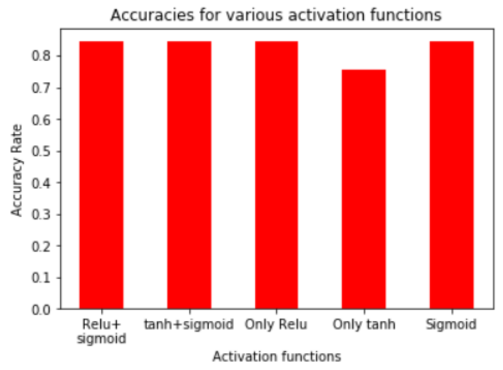




- Next, we change neurons from 30 to 50 for the second layer. We see that we can achieve maximum accuracy when neurons=40 at 84.32%
- Finally, we change neurons from 10-40 in the last hidden layer. We achieve maximum accuracy when neurons=30 at 84.33%.

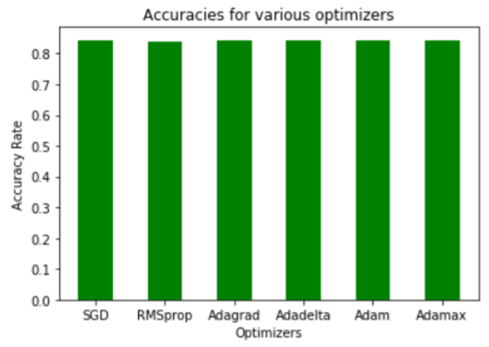
Thus, the best combination is **60-40-30**

Experimentation with activation functions:



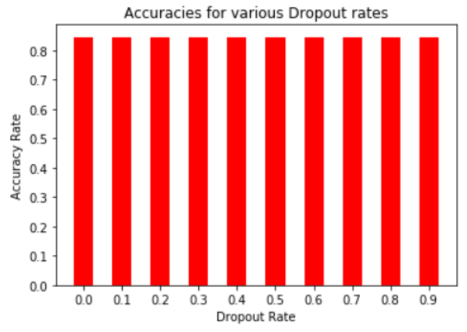
The different activation functions we use are **tanh, sigmoid and relu** as all these work very well with binary classifications. We normally go with sigmoid for the output layer but we also check for the output layer with tanh and relu functions. We can see that **tanh for the hidden layers and sigmoid for the output layer performs best of all with 84.31% accuracy**.

Experimentation with optimization functions:



We experiment with different optimization functions such as **SGD, RMSprop, Adagrad, Adadelta, Adam and Adamax**. All the optimizers perform almost similar. But the highest accuracy is achieved by **Adagrad at 84.38%.**

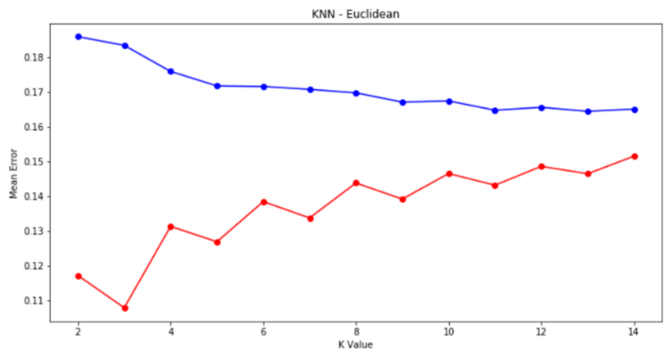
Experimentation with Dropout rates:

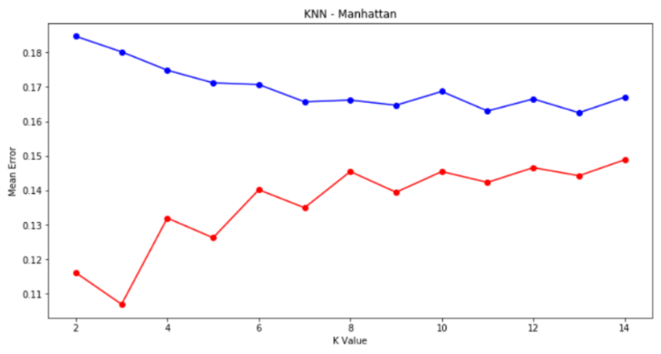


We do not want to over fit the data. Hence dropout is an option to reduce overfitting and hence we try dropout rates from 0.0 to 0.9. We can see that accuracies are almost similar with minute differences. But the best accuracy is achieved when **dropout=0.1 at 84.52%**
Finally, after choosing all the optimum hyper parameters for this model we train the model and test it with our test data. We get the below accuracies.
Train accuracy: 84.75%
Test accuracy: 84.88%

Algorithm 2: K Nearest Neighbors (KNN):

Experimentation with distance metrics and number of neighbors:
We are experimenting with Euclidean and Manhattan metrics as these are the widely used metrics. The number of neighbors are experimented from 2 to 15.
Though both give similar performance the Manhattan metric gives slightly better accuracy for all the neighbor values. Also, we can see that after K=8, the model seems to generalize well for the test data as well. Till K=7, we can see a clear evidence of over fitting as training error is low and test error is high. Hence we choose **K=8 and Manhattan distance** as our parameter. We achieve maximum accuracy of **83.38%.**





Comparison between the 2 algorithms:

For this dataset, Neural Networks gives 84.88% accuracy and KNN gives 83.38%. Neural network performs well for this dataset because the choice of neurons and number of hidden layers might be good to handle the data. Also, the number of features is 16 which is more than the features we used in the first dataset. Hence ANN is better equipped to handle more features than KNN.

Comparison of all algorithms:
For the previous assignment, the highest accuracy was given by Gradient Boosting at 84.82%. **But ANN outperforms it by 0.06% making it the best performer for this dataset**. KNN's poor performance might be due to curse of dimensionality because we use 16 features which is generally high. Hence KNN does not do well.

Ranking of the algorithms:
What I could have done?
Feature selection could have been done to select the 10 most important features as this would have helped to increase the accuracy of KNN to a large extent. Also, the accuracy for ANN might have also improved.