CSA 250 Deep Learning

Assignment 1

Python Dependencies and Requirements

- python (v3.7)
- tensorflow (v2.1)
- Other libraries used: argparse, numpy, os

Instructions to run

The ‘main.py’ is the interface to the program. It is programmed to run in two modes – train mode and test mode. The ‘main.py’, when executed without any arguments, enters into training the deep network. The ‘main.py’ when executed with the (optional argument) ‘--test-data <test_file>’ enters into test mode and produces the ‘Software1.0.txt’ and ‘Software2.0.txt’.

- Train mode : python main.py
- Test mode : python main.py –test-data <test_file>

  NB: <test_file> has to be in the same directory as ‘main.py’

Program details

This program is an implementation of FizzBuzz using traditional methods (Software1.0) and using deep learning (Software2.0). The FizzBuzz is a simple task of classifying numbers into four categories – fizz (numbers divisible 3), buzz (numbers divisible by 5), fizzbuzz (numbers divisible by 15) and other numbers as such. The program has three major components:

- ‘main.py’ : This python file acts as a client file for the user to interact
- sofware_1_0 : This is a python package consisting of files for implementation using traditional modulo arithmetic.
- sofware_2_0 : This is a python package consisting of files for implementation using modern deep learning.

‘main.py’

The ‘main.py’ is programmed to run in two modes – train mode and test mode. The ‘main.py’, when exectuted without any arguments, enters into training mode and when executed with the optional argument test mode.

‘software_1_o’

The software_1_0 package consists of files used for implementing FizzBuzz using modulo arithmetic. This package consists of two files.

- fizzbuzz.py
- accuracy.py

‘fizzbuzz.py’ module has a fizzbuzz() function that takes a filename (<test_file>) as input. The function then opens the input file, reads it line by line to obtain the number and classifies it into one of the four categories. It then writes the classes to a text file ‘Software1.txt’ in the same directory as that of ‘main.py’.

‘accuracy.py’ module has an accuracy() function that compares the ‘Software1.txt’ outputted by ‘fizzbuzz.py’ with ‘test_output.txt’ file present in the same directory as that of ‘main.py’. It finally calculates and prints the accuracy of classification.

‘software_2_o’

The software_2_0 package consists of files used for implementing FizzBuzz using deep learning. This package consists of five files.

- generate_train_data.py
- model_parameters.py
- model_train.py
- model_test.py
- model_accuracy.py

‘generate_train_data’ module has three functions of which generate_train_data() generates the training data set depending on the arguments passed such as TRAIN_BEGIN, TRAIN_END etc. Each number between TRAIN_BEGIN and TRAIN_END is converted to an numpy array with entries according to the binary sequence of that number. Also for each number, one hot encoded (according to the class) categorical numpy array is generated. These two arrays for the training data (train_X and train_Y).

‘model_parameters.py’ has model hyperparamter variables such as epochs, learning rate, batch size etc initialized with values.

‘model_train.py’ module has the model_train() function, which trains the deep network using the training data generated with tensorflow backend. After the model is trained, it is stored in a ‘./model’ directory.

‘model_test.py’ module has model_test() function that takes a filename (<test_file>) as input. The function opens and loads the saved model from ‘./model’ directory. It then opens the input file, reads it line by line to obtain the number and classifies it into one of the four categories depending on the output of the network model. It then writes the classes to a text file ‘Software2.txt’ in the same directory as that of ‘main.py’.

‘model_accuracy.py’ has a model_accuracy() function that compares the ‘Software2.txt’ outputted by ‘model_test.py’ with ‘test_output.txt’ file present in the same directory as that of ‘main.py’. It finally calculates and prints the accuracy of classification.

Model architecture

The model is defined using Keras API within Tensorflow. The architecture has two Dense layers with 512 units each. Relu activation is used for both layers for activating the outputs. A dropout layer is added after these dense layers to regularize and prevent overfitting. The loss function used was 'categorical_crossentropy', and the optimizer used was 'RMSprop()' with learning rate of 0.001. The output layer of the network is a dense layer with four categories, with softmax activation. The model was trained for 50 epochs with batch size of 32.

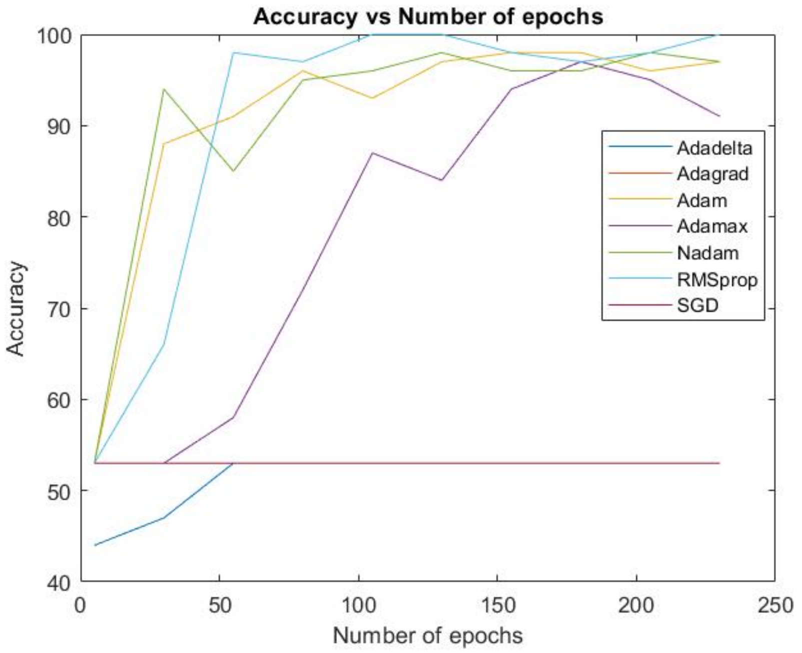
Experiments

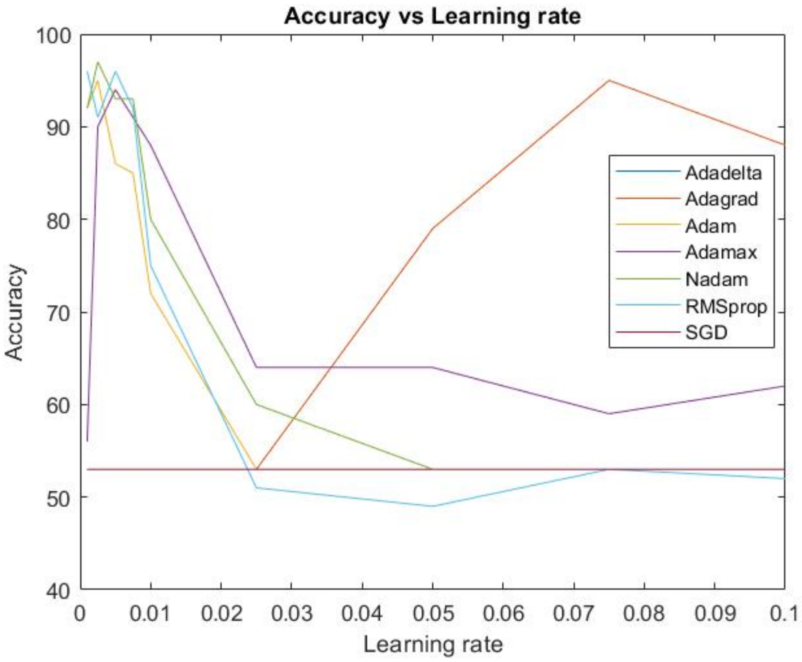
For the experiments, the following model was fixed to be the base model and was evaluated by varying some of the hyperparameters.

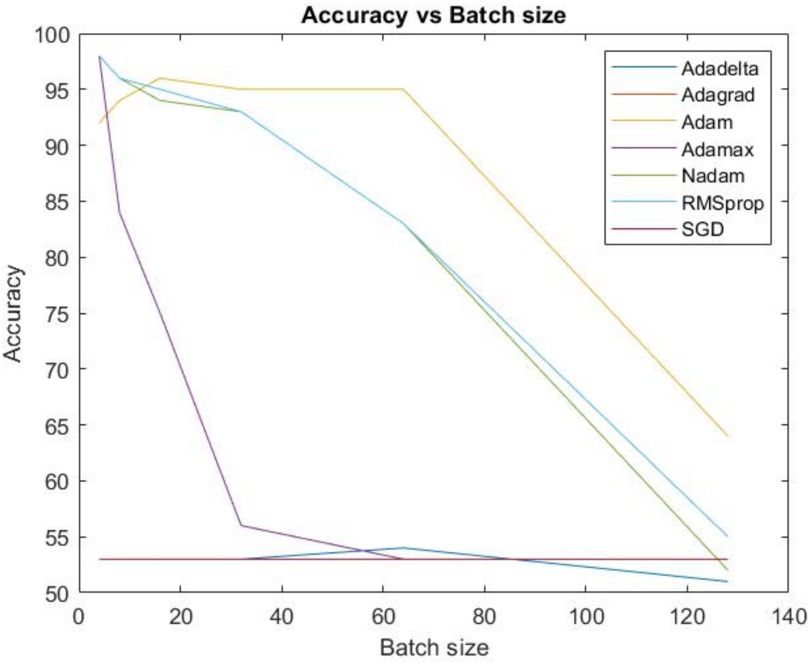
No. of dense layers	2
No. of units in dense layer	512
No. of dropout layers	2
Dropout rate	0.2
Activation for hidden layers	ReLu
Activation for output layers	Softmax
Loss function	Categorical cross entropy
Training epochs	50
Optimiser	RMSprop
Learning rate	0.001
Batch size	32

The major experiments conducted was to analyse the performance of the network in terms of its accuracy when the number of epochs, learning rate and batch size was changed. This was evaluated for all major optimisers available and was plotted (figures in next page).

The number of hidden layers, dropout layers were varied as well. But, the performance accuracy was erratic and did not follow any pattern. Hence they are not presented in this report. The above base model was obtained as a result of such explorations with model architecture.


Accuracy vs Number of epochs
Accuracy
Number of epochs
100
90
80
70
60
50
40
0
50
100
150
200
250
Adadelta
Adagrad
Adam
Adamax
Nadam
RMSprop
SGD


Accuracy vs Learning rate
Accuracy
Learning rate
100
90
80
70
60
50
40
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
Adadelta
Adagrad
Adam
Adamax
Nadam
RMSprop
SGD


Accuracy vs Batch size
Accuracy
Batch size
100
95
90
85
80
75
70
65
60
55
50
0
20
40
60
80
100
120
140
Adadelta
Adagrad
Adam
Adamax
Nadam
RMSprop
SGD

Results

The FizzBuzz task was implemented using the traditional modulo arithmetic method as well as using the deep learning method. The modulo arithmetic method (Sotfware1.0) produced an accuracy of 100%. On the other hand, the deep learning method (Software2.0) produced an accuracy of 98% on the test data (with the base model).

The Software1.0 implementation is bound to produce 100% accuracy due to the nature of its logic. The Software2.0 implementation uses a deep network that learns the different characteristics of the four classes from the training data and then uses it to predict the classes for test data. The performance of the network varies with different parameters, different optimizers and different structures. The structure used in this project was obtained by trying many of such combinations.

From the plot between accuracy and the number of epochs, we can infer that as the number of epochs increases, accuracy increases initially, but it saturates after a particular point. As the epoch count increases the time taken for training increases as well, and the chances for overfitting the data is quite high.

From the plot between accuracy and learning rate, we can observe that as the learning rate increases, there is a steep decrease in the accuracy. This could be attributed to the skipping of learning the essential features due to a large learning rate. If the learning rate is low, it takes too much of time to converge and if it is high, the network may fail to lean weights properly. In the plot, there is an abnormal curve (corresponding to that of Adagrad) where accuracy increases with learning rate for which I do not have any valid explanation as of now.

From the plot between accuracy and batch size, we can observe that as the batch size increases, there is a decrease in the accuracy for all optimizers.