PROJECT REPORT

CSE574
INTRO TO MACHINE LEARNING

PROJECT 1.1

BY
YASH AHUJA
50245092
yashahuj@buffalo.edu

1. Introduction

This is a machine learning class project. The problem taken into consideration is the classic Fizz-Buzz problem. The point of this project is to get familiarized with how machine learning works when compared to normal programming and also to get comfortable with python itself.

2. Problem and model description

The idea is to compare the testing between software 1.0 and software 2.0.
Software 1.0 uses logical python functions and Software 2.0 uses machine learning where you have training and testing sets.
For my project report I have used Keras as my machine learning framework. The necessary things done are:

1. Process data
2. Define the model
3. Run the model
4. Plot results

Brief description on data processing:

- The goal of this is to clean the dataset in order to get an accurate training set
- Binary encoding to increase the number of features from 3 to 10

Brief model description:

- Keras is used as the machine leaning framework
- The simplest model, a sequential model, is used here
- Two dense layers are used in our case-one is for the input and the other is for the output
- Drop-out, activation function, and optimization method are taken as the hyperparameters in order to improve the resulting accuracy

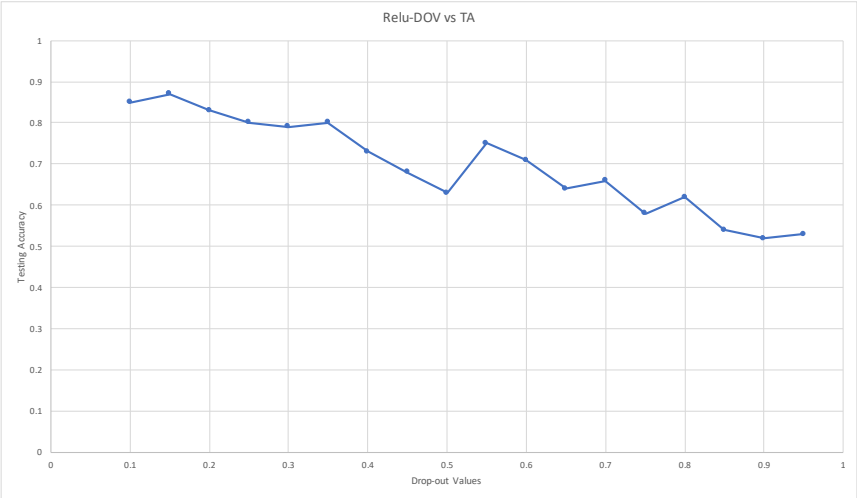
Brief description on plotting of results

- Plotting a graph by varying drop-out rates
- Plotting a graph by varying number of nodes
- Constructing a table using different activation functions like relu, elu, selu, tanh, sigmoid, etc
- Constructing a table using different optimization methods like sgd, rmsprop, adamax, adam, adadelta, etc

3. Results

GRAPHS

1. Varying Drop-out Rates



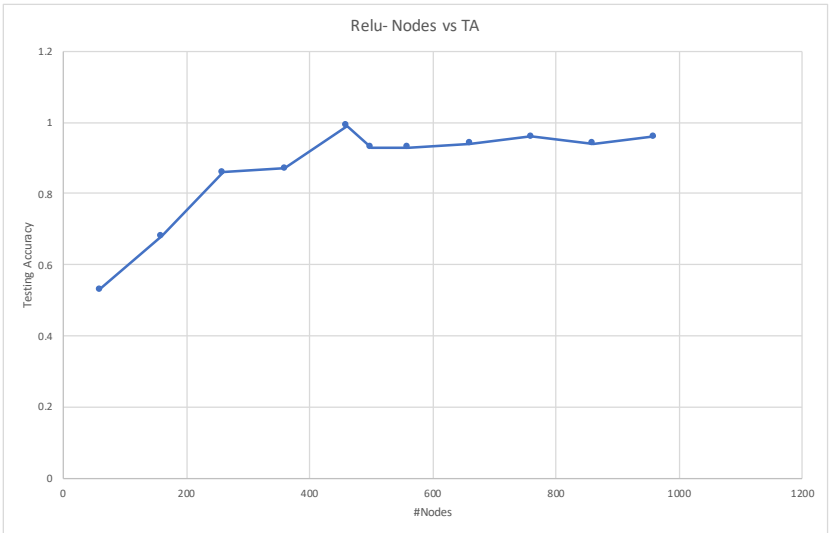
Note: The other parameters taken are default. As follows:

- Activation Function: *relu*
- Optimizer: *rmsprop*
- Number of nodes: *256*

Comments:

As seen from the above graph, I have varied the drop-out rates and found the testing accuracy for each case. We can see that there is a gradual decrease in the accuracy as we increase the drop-out rate. This is because when you increase the drop-out value beyond a certain threshold, a lot of the input features become zero (hence the overall model will have less features and not fit properly), reducing the capacity and learning rate of the model which in turn decreases the accuracy.

2. Varying the number of nodes



Note: The other parameters taken are default. As follows:

- Activation Function: *relu*
- Optimizer: *rmsprop*
- Number of nodes: *256*

Comments:

As seen from the above graph, the testing accuracy increases as the number of nodes increases. After around 500 nodes, the testing accuracy saturates. When there are fewer nodes, the model gets difficult to predict as the network does not have enough nodes to give a good accuracy. This is because for the network to predict the model accurately, it needs to have enough or more nodes to get a solid pattern to learn from.

TABLES

1. Activation Function

	Activation Function	Testing accuracy
1.	***relu***	0.84
2.	Sigmoid	0.53
3.	tanh	0.53
4.	selu	0.53
5.	softmax	0.53
6.	softsign	0.53
7.	elu	0.53
8.	softplus	0.53

Note: The other parameters taken are default. As follows:

- Drop-out rate: *0.2*
- Optimizer: *rmsprop*
- Number of nodes: *256*

Comments:

As seen from the above table, the activation function '*relu*' gives us the best accuracy. Why *relu* is advantageous and gives a better accuracy over the other activation functions is that it does not activate all the neurons at the same time. This is because the input that is negative is converted to zero which deactivates neurons in that region, making it more efficient.

2. Optimization Method

	Optimization Method	Testing Accuracy
1.	RMSprop	0.83
2.	SGD	0.53
3.	adagrad	0.85
4.	***adadelta***	0.87
5.	adam	0.79
6.	adamax	0.86

Note: The other parameters taken are default. As follows:

- Activation Function: *relu*
- Drop-out rate: *0.2*
- Number of nodes: *256*

Comments:

Our data set is a sparse one which is quite evident as 'relu' activation function is giving a drastically better performance. For sparse datasets, adaptive learning-rate methods are used. This is also evident from the above table as you can see 'SGD' has a poor performance and the adaptive learning-rate methods like adagrad, adadelta, adam, and adamax have good performances. In these methods, the learning rate does not need to be adjusted and the results will still be good with default values.

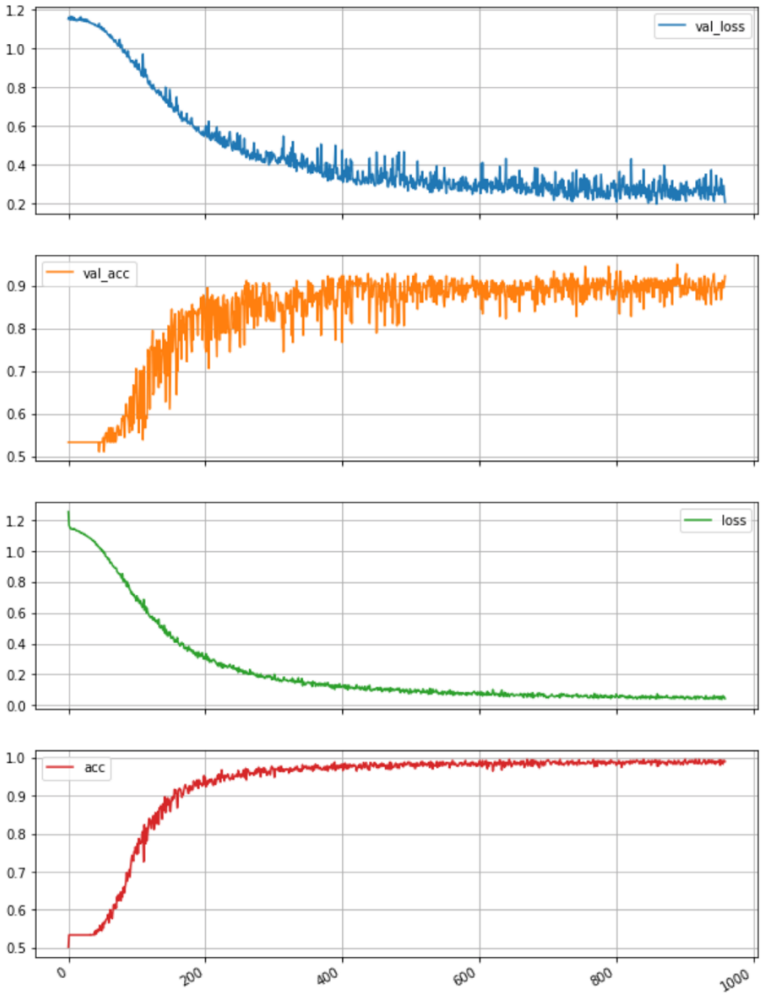
In my performance adadelta gave me the best accuracy with negligible difference to adagrad and adamax. Adadelta is the extension of adagrad overcoming the decaying learning rate problem.

3. Optimal Solution with optimal parameters

	Parameters	Optimum Results
1.	Drop-out Rate	0.2
2.	Number of nodes	960
3.	Activation Function	relu
4.	Optimization Method	adadelta

The testing accuracy result obtained is **100%**

The following is a plot obtained from matplotlib in python:



Description of the matplotlib plot
For the plot above the following results were observed:
#Epochs – 960/10000 due to early stopping
Validation loss (val_loss) – 20.91%
Validation Accuracy (val_accuracy) – 92.22%
Training loss (loss) – 4.34%
Training accuracy (acc) – 99.03%

We can see that beyond around 200 epochs, in the cases of the accuracies as well as the losses, saturation is observed. That is, the loss is low and the accuracy is high after around 200 epochs. When comparing validation accuracy and training accuracy, the validation accuracy has more noise. The same thing is observed for the losses.

4. Conclusion

It can be observed that Software 2.0 runs more efficiently than Software 1.0. With this project, we learn the use of Keras framework and also the intricacies of it, like the use and different types of models, the why's and the how's of the hyperparameters, and how important each element of the model is.