University at Buffalo

CSE 574: Introduction to Machine Learning
(Fall 2018)

Project 1.1: Software 1.0 Versus Software 2.0

Author: Ameen M Khan
Person Number: 50288968

Problem Statement The project is to compare two problem solving approaches to software development: the modulo arithmetic based approach (Software 1.0) and the machine learning approach (Software 2.0). It is also designed to quickly gain familiarity with Python and machine learning frameworks.

1 Introduction

Software 1.0 is implemented using logic based approach (if-else conditions). Software 2.0 is implemented using *Keras* on top of *Tensorflow*, a machine learning framework, and its performance is measured for different *hyperparameters*. The machine learning approach used in Software 2.0 is a **Sequential Neural Network**, which has the ability to learn from the training data provided to it and predict the outcome for data in terms of probabilities, using this already *learnt* model.

Anatomy of Neural Networks

The training of Neural Network depends on following objects:

1. Input Data (Training, Testing, Validation): We split our input data into training and testing, so that the model is never tested on the same data, so as to avoid model to overfit. Validation data is used to monitor training of the accuracy of the model.

2. Layers of neural nodes: These are stacks of layers of data processing units capable of performing tensor operations. The state of the neural networks is defined by *weights*, which are random initially in training phase, but are updated in the every subsequent pass over the data.
   For our problem statement, we are using densely connected layers, which are sufficient for processing 2D tensors.

   - Each hidden unit has an *activation function* associated to it, to provide non linearity to it. Without this, the layer would consist of only linear operations (dot(W, input) + b), meaning layer could perform linear transformations, leading to a much restricted hypothesis space which wouldn't be able to benefit from.

3. Loss Functions or Objective Function: This is the quantity that will minimized during training of the model.

4. Optimizers: This determines how the network of layer of neural nodes will be updated based on the loss functions.

2 Methodology

2.1 Preparing the data

The training and testing data is generated using the Software 1.0 approach. This is then processed and converted into tensors, so they are compatible for the neural networks to operate on them.

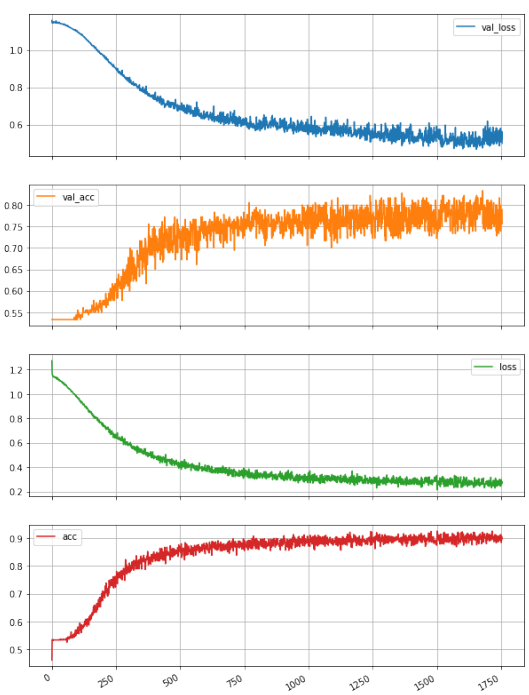
2.2 Setting up the architecture

I started off with the network made up of an input layer (encoded input data), one hidden layer (Dense), and one output layer consisting of 4 units (each corresponding to an output class).
From there I explored the various neural network architecture decisions like selection of number of hidden layers (1-2), number of hidden nodes in each layer(256-1024) and different activation functions in the hidden and output layers (relu, sigmoid and softmax).
Also, I applied *dropout* to the hidden layer(s), which consists of randomly dropping out (ignoring) some nodes in the layer, during training phase and is commonly used and effective regularization method for neural networks.
For our problem statement, which is a common multi-class classification, I chose to stick to the general guidelines and used *categorical crossentropy* loss. Finally, for choosing an optimizer I tried different optimizers available in the Keras API (RMSprop, SGD, Adam, Adadelta etc.) and different values for the learning rates.

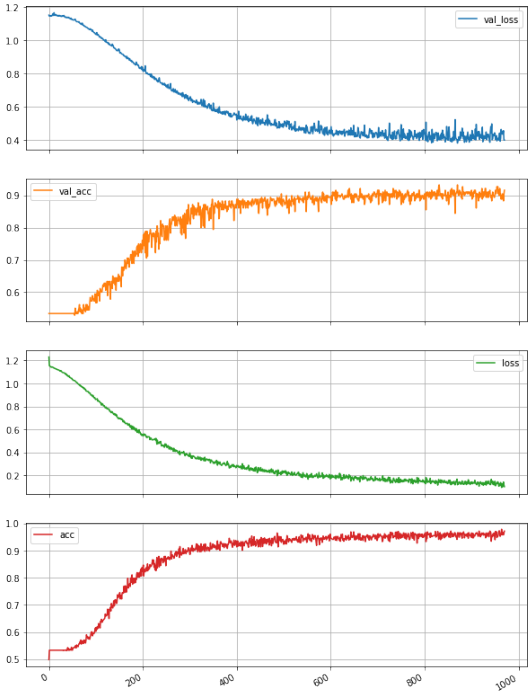
Besides the above stated parameters, I also tried different epoch settings for the finalized architecture to check for a better testing accuracy.

3 Experiments and Observations

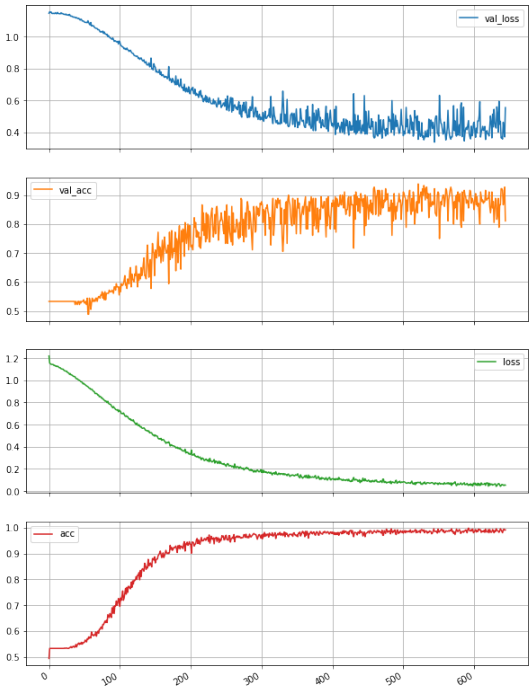
3.1 Neural Layers and Nodes



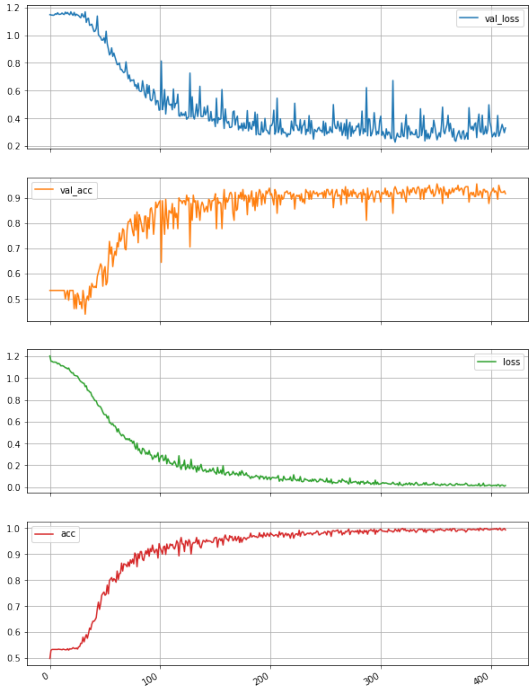
(a) Single hidden layer with 256 nodes



(b) Single hidden layer with 512 nodes.



(c) Single hidden layer with 1024 nodes



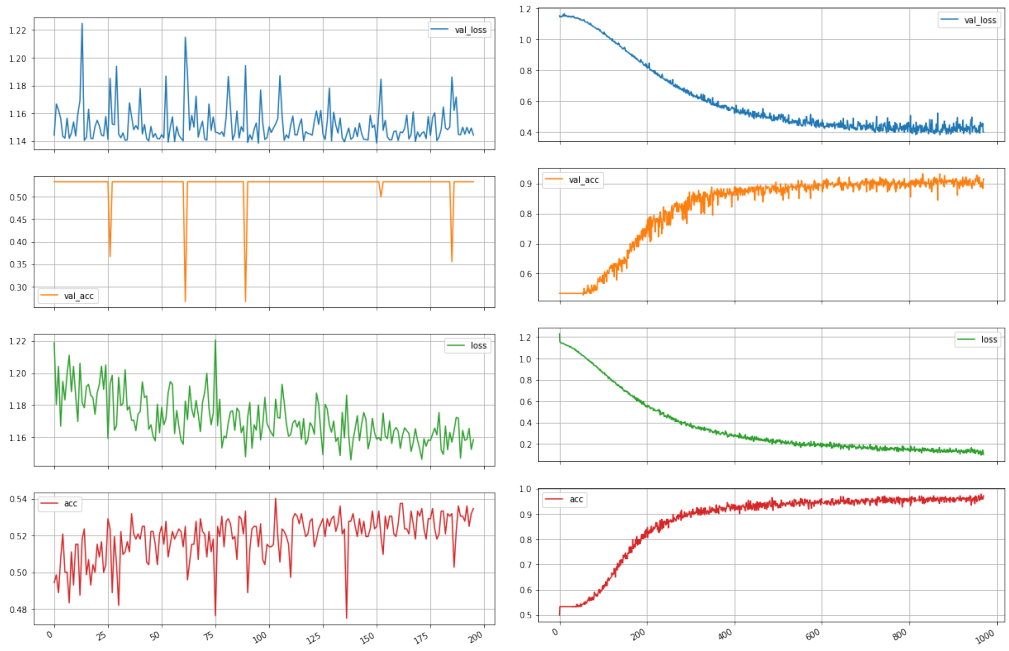
(d) 2 hidden layers,512 nodes and 256 respectively

The graphs in the figure are for different neural network hidden layer-nodes architectures. As I increased the number of nodes in the hidden layer, the early stopping kept on decreasing, from 1750 to 600 when increased from 256 to 1024 nodes. I conducted multiple trial for each changed hyperparameters, and present only the mean value of each.

Layer	Nodes	Testing Accuracy
Single Hidden Layer	256 Nodes	85
Single Hidden Layer	512 Nodes	91
Single Hidden Layer	1024	93
Two Hidden Layer	512, 256	94

3.2 Activation Functions

When the activation function in hidden layer(s) is updated from relu to sigmoid, the testing accuracy of the model dips down 53%. This is probably because of *vanishing gradient problem*, which isn't the case for ReLU.



(a) Sigmoid Activation (b) Relu Activation

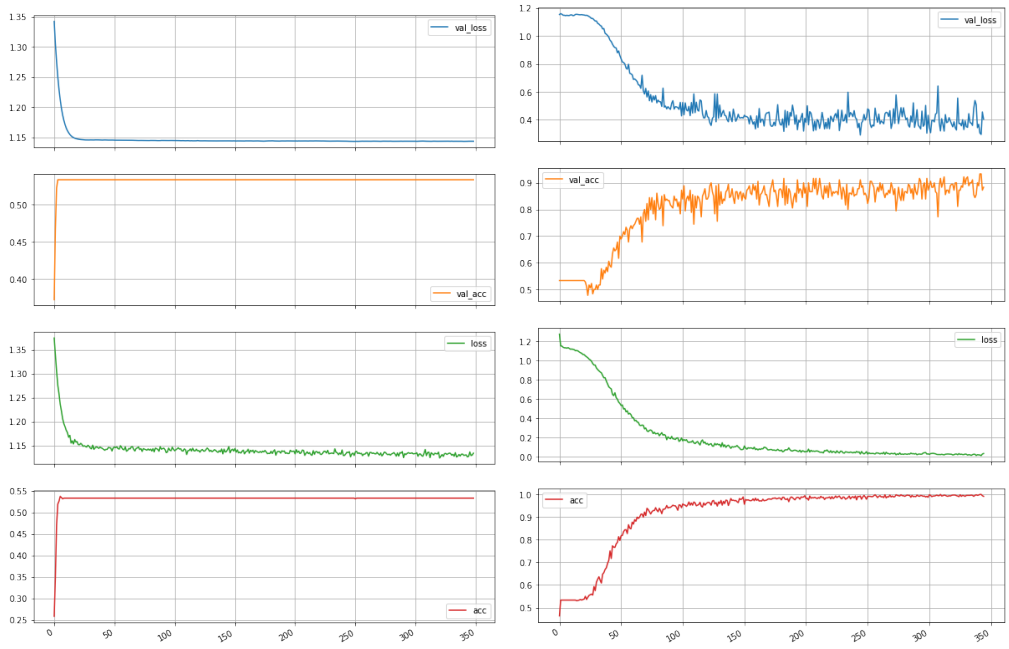
Figure 2: Graphs for different activation function in single hidden layer with 512 nodes

3.3 Optimizers

I used RMSprop optimizer, as suggested by the provided notebook in all previous trials. The default learning rate as defined in Keras is *0.001*

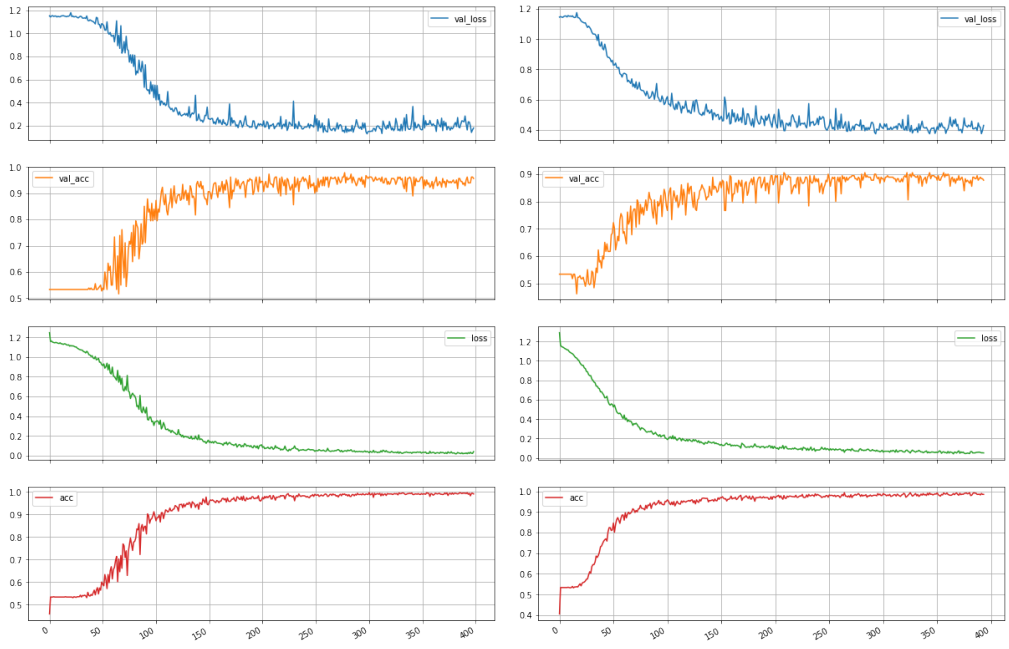
The two hidden layer (512, 256) architecture is tried with the SGD optimizer, and has testing accuracy of 54. I further tried the same architecture with the Adam, Adadelta and

Adagrad optimizers with their default learning rates, which have testing accuracies of 92, 98 and 93 respectively.



(a) SGD Optimizer

(b) Adam Optimizer



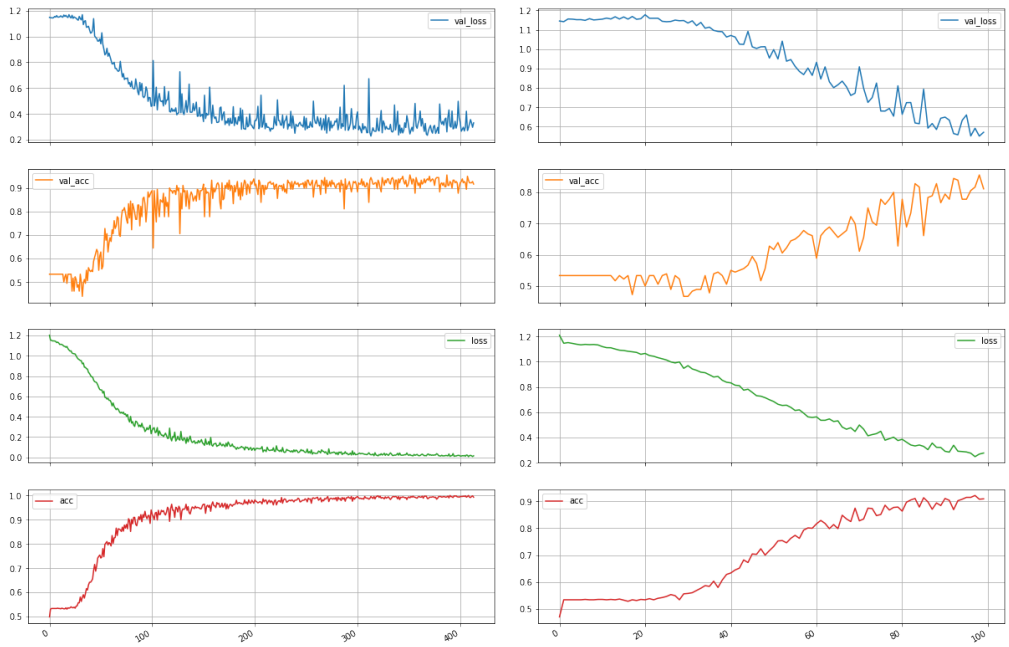
(c) Adadelta Optimizer

(d) Adagrad Optimizer

Figure 3: Graphs for different optimizers in 2 hidden layer architecture 512, 256 nodes.

3.4 Epochs

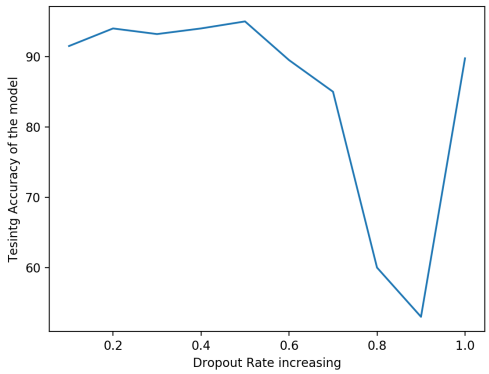
Number of epochs are irrelevant if they are greater than the point at which the model training encounter *early stopping*. When number of epochs is decreased to a point much lower than that of early stopping has decreased testing accuracy of 83.



(a) Normal Epochs with early stopping (b) Low Epochs

Figure 4: Comparison of architecture performance with reduced epochs with normal epoch configuration.

3.5 Dropout Rate



(a) Change in testing accuracy with increase of dropout rates

The dropout rate is the fraction of nodes which are ignored during the testing phase. In my model, the dropout rate of 0.2 (meaning 20percent of the nodes in the layer) has been applied to the two hidden layers. In the previous figure, the trials were conducted on the 2 layer (512,256) network using RMSprop optimizer.

4 Conclusion

The final architecture of the neural network, which performed the best after all the above trials:

- **Layers**: 2 hidden layers. Hidden Layer 1 of 512 nodes, Hidden Layer 2 of 256 nodes. Output layer of 4 nodes, each corresponding to a target class of the multi-class classification problem.
- **Activation Functions**: The hidden layers are activated by ReLU. The output layer is activated by softmax.
- **Loss Functions**: The loss functions used is *categorical crossentropy*, which is suggested for this type of problem.
- **Optimizer**: Adadelta Optimizer gave the best results, and with consistency over all the trials conducted for each variations.
- **Epochs**: The epochs set is at 1000, but for these settings the model early stops at around 520 epochs.
- **Dropout Rate**: The dropout rates between 0.2 and 0.5 produced the best results.

The final testing accuracy of the model, is averaging at 97.

5 References

1. Deep Learning with Python, Manning Publication
2. Hands-On Machine Learning with Scikit-Learn and TensorFlow, OReilly Publication
3. https://github.com/donnemartin/data-science-ipython-notebooks