Convolutional Neural Network for MNIST classification

Shalin Rathi
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA
sjrathi@ncsu.edu

Sanjana Kacholia
Department of Computer Science
North Carolina State University
Raleigh, USA
skachol@ncsu.edu

Shraddha Dhyade
Department of Computer Science
North Carolina State University
Raleigh, USA
sddhyade@ncsu.edu

I. INTRODUCTION

The objective of the project is to implement CNN model in Python for MNIST data classification.

II. STRUCTURE OF NETWORK

The network was built using Keras in Python.

- Input Layer
- Convolution 2D (16 layers)
- Batch Normalization
- Max Pooling with pool size = (2,2)
- Dropout layer (0.25)
- Convolution 2D (32 layers)
- Max Pooling with pool size = (2,2)
- Flattening
- Dense layer with units=128
- Dropout (0.5)
- Optimizer - Adam
- Output layer

III. EXPERIMENTS WITH HYPERPARAMETERS AND VISUALIZATION OF CROSS-VALIDATION FOR HYPER-PARAMETERS SELECTION

We did a grid search for parameters and experimented with following values -

- Batch size - 32, 64, 128
- Epoch - 10, 15, 20
- Learning rate - 1e-2, 1e-3, 1e-4
- Kernel Size - (2,2), (3,3), (4,4)

  The following combination gave the best results -
- Learning Rate = 1e-3
- Number of Epochs = 20
- Batch Size = 64
- Kernel Size = (4, 4)

The accuracy by varying one hyperparameter at a time and keeping other parameters as the best parameters found above, can be analysed as in figures 1-4.

After these hyperparameters, we tested with various activation functions and the results are in Table 1

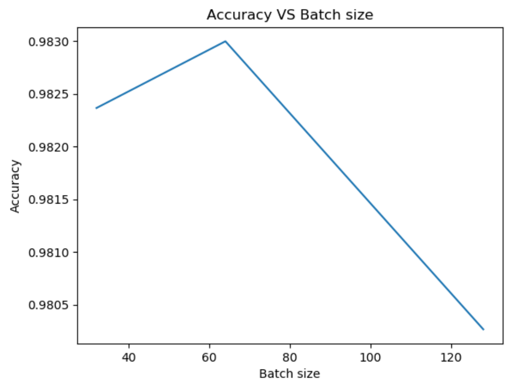

Fig. 1. Accuracy VS Batch size

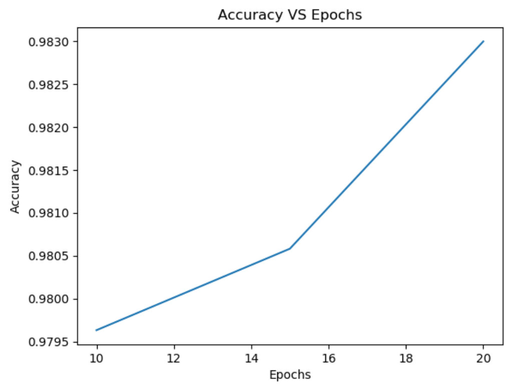

Fig. 2. Accuracy VS Epochs

TABLE I
RESULTS OF VARIOUS EXPERIMENTS

Activation	**Accuracy**	
	Train	***Valid***
ReLU	0.9889	0.9907
Sigmoid	0.9845	0.9845
tanh	0.9839	0.9855

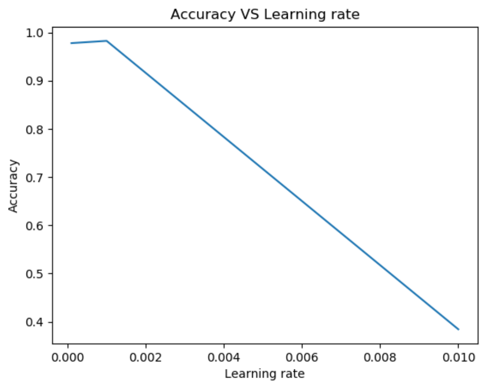

Fig. 3. Accuracy VS Learning rate

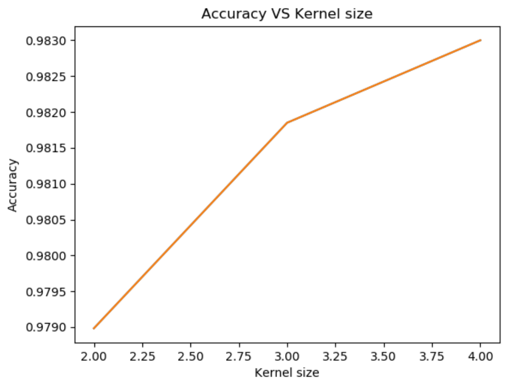

Fig. 4. Accuracy VS Kernel size

IV. Visualization of performance of best hyper-parameters for different activation functions

After selecting the hyper parameters from the grid search we do regression to find the best activation function. Figures 5-8 display the graphs for the training and validation data for ReLU, sigmoid and tanh activation functions using 1-fold cross validation. From these graphs it can be seen that ReLU activation gives the best accuracy and lowest loss out of the three activations.

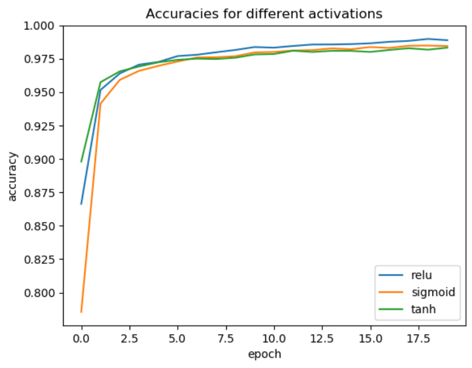

Fig. 5. Training accuracy for different activation functions

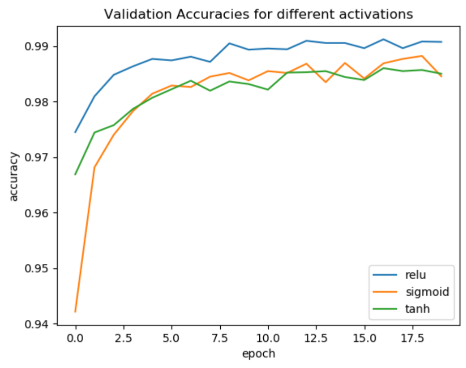

Fig. 6. Validation accuracy for different activation functions.

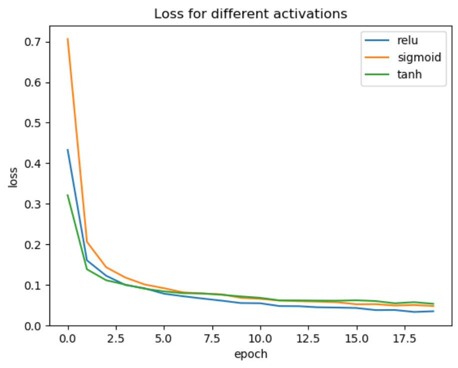

Fig. 7. Training loss for different activation functions

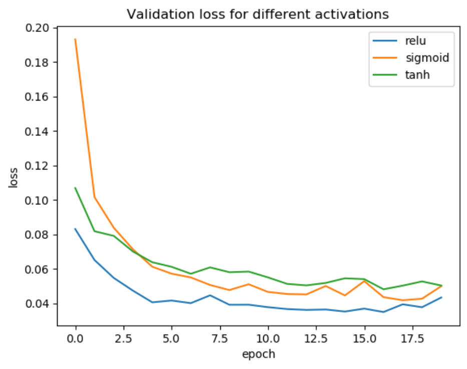

Fig. 8. Validation loss for different activation functions.

V. Learning Curve

Figures 9 and 10, show the accuracy and loss for the final model after hyper parameters tuning and activation function selection.

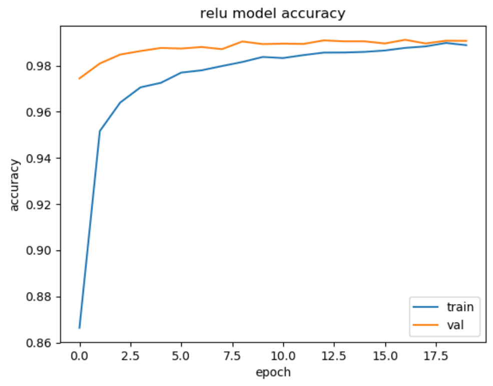

Fig. 9. Final Accuracy curve

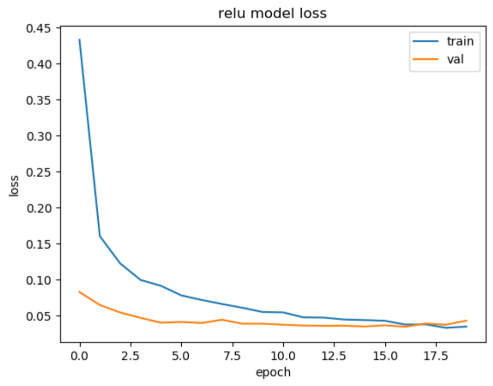

Fig. 10. Final Loss curve

VI. Accuracy on Testing Set

The final model with ’ReLU’ activation function gives an accuracy of 0.9922 on testing set with a loss of 0.0259.

VII. Analysis of Results

Based on the accuracy graphs, it can be seen that it takes at least 17 epochs to achieve convergence. Also, it can be seen from the validation plots, at epoch 20 the accuracy goes down and loss goes up. Hence we selected 20 epochs to train over model, to avoid over fitting. With sigmoid and tanh activations, we faced a vanishing gradient problem, which was overcome by adding a regularization layer of batch normalization. We also faced overfitting while training our model, hence dropout layers were introduced, one between 2 convolution layers and another after the dense layer. The batch size 0f 64 is quite low for a training set of 60,000 samples, but gives the best performance from the range of all the hyper parameters in the grid search, when other parameters are fixed.