ECE542 / CSC591 - Neural Networks CNN for MNIST Classification

Suhas Keerthiraju
Dept. Electrical Engineering
NCSU
Raleigh, USA
skeerth@ncsu.edu

Shashank Shekhar
Dept. Computer Science
NCSU
Raleigh, USA
sshekha4@ncsu.edu

Mohit Manoj Vinchoo
Dept. Computer Science
NCSU
Raleigh, USA
mvincho@ncsu.edu

Abstract—Convolutional Neural Network(CNN) belongs to a class of deep neural network which makes use of convolution operation to preprocess the data and obtain a set of features. These features are later supplied to a MultiLayer Perceptron (MLP) which consists of an input layer, hidden layer(s) and an output layer. Every neuron in the hidden layer(s), the convolution layer(s) and the output layer uses an activation function. CNN utilizes a supervised learning technique called backpropagation for training. The performance of a CNN can be tuned by employing a number of hyperparameters and by using strategies such as dropout, adverserial training and many more. This report outlines the details of CNN designed to identify handwritten digits in MNIST dataset. The proposed CNN uses two convolution layers with ReLU activation, two pooling layers, a single dense/fully connected layer and an output layer with softmax activation. The network uses cross entropy loss as the cost function and achieves an accuracy of 99% on MNIST dataset.

Index Terms—CNN, convolution, pooling, ReLU, softmax, cross entropy loss

I. Introduction

CNN is comparable to the pattern in which the neurons in a human brain are interconnected. CNNs require minimmal amount of preprocessing compared to a regular feed-forward neural network. The goal of this project is to implement a CNN for handwritten digits identification and analyze the system performance.

II. Methodology

A. Network Structure

The network structure refers to the details of the input, convolution, pooling, hidden and output layers. This configuration determines the dimensionality of network parameters. As the network structure becomes more complex, the dimensions of network parameters grow proportionally. Configuration also covers the strategies employed to enhance the system performance. The network configuration used in the current project is detailed by Table I

The number of neurons in input layer depends on the input dimensionality. The MNIST dataset has images of dimensions 28*28 which are serailized into vectors of size 784. Thus, input layer consists of 784 neurons. The output layer consists of 10 neurons, each neuron corresponding to digits 0 to 9.The

TABLE I
CNN Network Configuration

Layer	Detail-1	Detail-2	Detail-3
Input Layer	784 neurons		
Convolution Layer #1	5*5 conv filter	28 filters	ReLU activation
Pooling Layer #1	2*2 pooling	Stride=2	Max pooling
Convolution Layer #2	5*5 conv filter	56 filters	ReLU activation
Pooling Layer #2	2*2 max pooling	Stride=2	Max pooling Hidden
Layer #1	1024 neurons	Dropout=0.5	ReLU activation
Output layer	10 neurons		Softmax activation

graphical representation of the network structure in AlexNet style is illustrated by Fig 1.

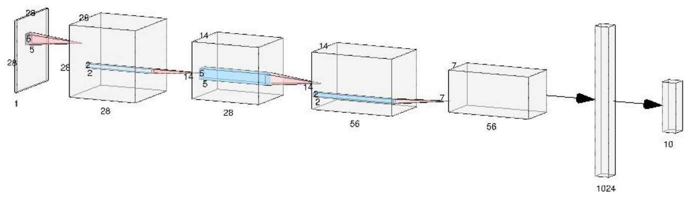


Fig. 1. CNN architecture

B. Network Parameters

The network consists of parameters namely weights and biases to be tuned by the process of backpropagation and stochastic gradient descent. These parameters are applicable only to the fully connected hidden layer(s) and the output layer. The dimensions of these parameters in various layers are mentioned in Table II

TABLE II
CNN Parameter Dimension for Dense Layers

Parameter	Dimension
b^1	1024*1
W^1	2744*1024
b^2	10*1
W^2	1024*10

C. Activation Function

Different activation functions are tried in the process of training. The details of these activation function are as follows

• The sigmoid activation function σ: R → R is given by the mathematical Equation 1 and Equation 2 gives the derivative of the sigmoid σ^0. Fig 2 gives the graphical view of σ and σ^0

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$

$$\sigma^{0}(x) = \sigma(x)(1 - \sigma(x)) \qquad (2)$$

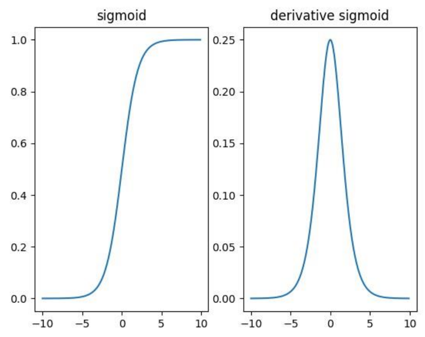


Fig. 2. Sigmoid and its derivative

• The mathematical representation of softmax activation function is given by Equation 3

$$\text{softmax}(x) = \frac{e^{x_i}}{\sum_i e^{x_i}} \qquad (3)$$

• Equation 4 gives the mathematical representation of tanh activation function and Fig 3 is the visual representation of the same

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \qquad (4)$$

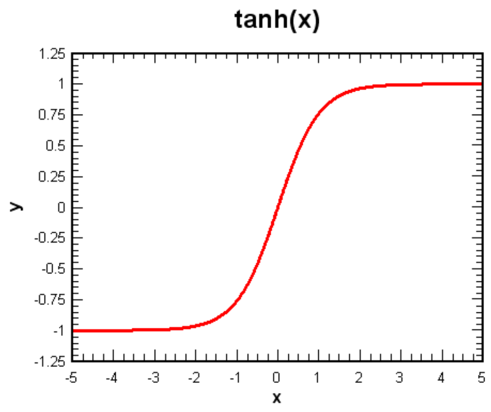


Fig. 3. tanh activation

• Equation 5 gives the mathematical representation of ReLU activation function and Fig 4 is the visual representation of the same

$$\text{ReLU}(x) = \max(0, x) \qquad (5)$$

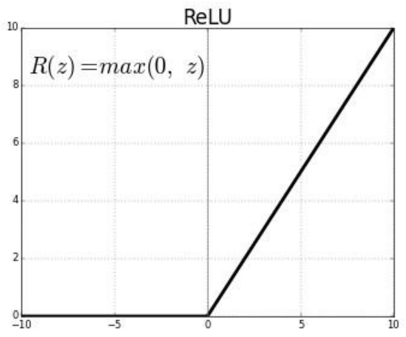


Fig. 4. ReLU activation

D. Cost Function

The proposed network uses a cross entropy loss as the cost function to track the deviation of network outcome from the desired outcome. Calculating the gradient of cost function is the starting phase for backpropagation. The details of the gradient is as follows

$\nabla_a L(\sigma(a))$ where $L = -(1-y).\ln(1-\hat{y}) - y.\ln(\hat{y})$
and σ(a) is the sigmoid function.

$$\nabla_a L(\sigma(a)) = -y + \sigma(a)$$

E. Hyperparameters

The hyper parameters to be tuned for improvements in network performance include batch size, number of epochs, learning rate, dropout probability, number of hidden layers, number of neurons in each hidden layer, type of activation function, kernel size and type, pooling function and many more. But, for experimentation purpose, only 4 hyperparameters i.e. batch size, number of epochs, learning rate and dropout probability are considered. The range of hyperparameters explored were limited mainly due to hardware capacity. The range of these values can be explored further with hardware with higher capacity. The hyper parameter values for optimum system performance is given Table III

TABLE III
RANGE OF HYPERPARAMETERS UNDER CONSIDERATION

Hyperparameter	Range
Mini batch size	[100, 1000]
Epochs	[10, 100]
Learning rate	[0.001, 1]
Dropout probability	(0, 1)

F. Dropout

Dropout refers to ignoring set of neurons in a network which is chosen at random during the training phase. At each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p. This results in a reduced network with incoming and outgoing edges to a dropped-out node are also removed. This is a form of regularization to prevent overfitting by penalizing the loss function. Dropout works as a regularizer by reducing the interdependency between the neurons by randomly eliminating the neurons. Dropout increases the number of epochs required for the network to converge, but at the same time reduces the time required to complete each epoch. So, roughly speaking the total training time increases by a factor of two. Also, since the neurons are dropped at random, the final resulting parameters are equivalent to the aggregate of number of models. During the testing phase the final weights are multiplied by the probability that the neuron in present. Fig 5 gives the visual representation of the working of dropout during training and testing phase. Also, it indicates how dropout works as a regularizer.

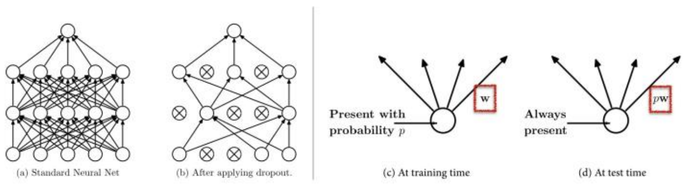


Fig. 5. Dropout as a regularizer

III. Training and Validation

To check if the fitted model is appropriate i.e. to make sure that the model is neither under fitted nor over fitted, it is important to plot the learning curve with respect to training and validation sets. In the current scenario, the training set consisted of 60000 images. This set was divided into training and validation set with ratio 70:30. Fig 6 represents the variation of training and validation loss with respect to epoch number. The decreasing trend in the curve indicates that the model improves over the epochs and thus loss decreases. Fig 7 indicates the training and validation accuarcy over diferent epoch numbers. As the number of epochs increases, the accuracy increases and tends to become almost invariant. Also, validation accuracy closely follows training accuracy indicating that the fit for the model is optimal.

Fig 8 visualizes the variation of training and validation loss with respect to the batch size. The increasing trend in the loss curve indicates that the model tends to degenerate in its performance over increasing batch size and thus the loss increases. Fig 9 indicates the training and validation accuarcy over diferent epoch numbers. As the batch size increases, the accuracy decreases. After a batch size of 60, the validation curve closely follows training curve which indicates the region of optimal value.

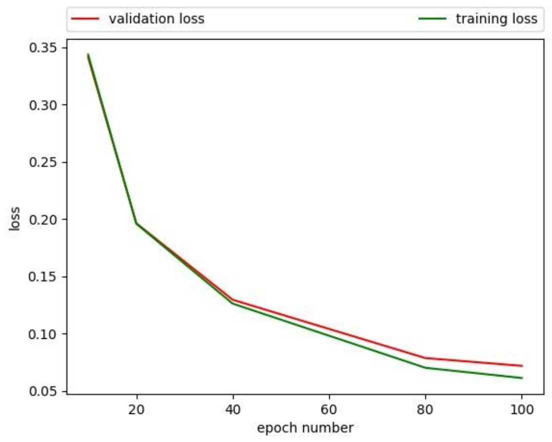
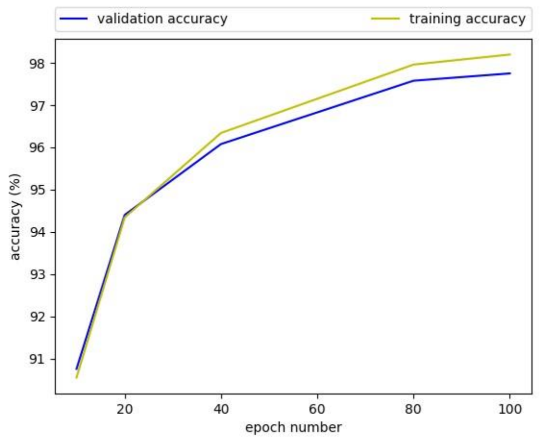
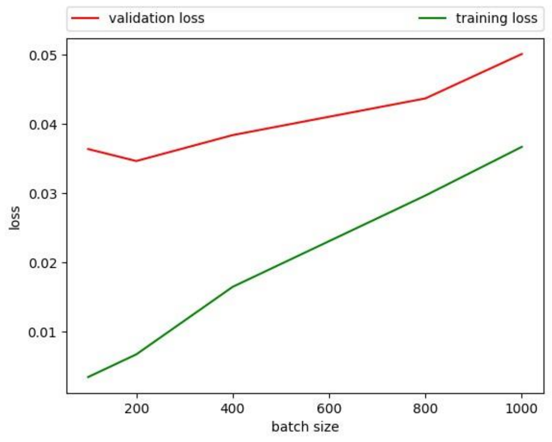


Fig. 6. Loss variation with epochs Fig.



7. Accuracy trend with epochs Fig. 8.



Loss variation with batch size

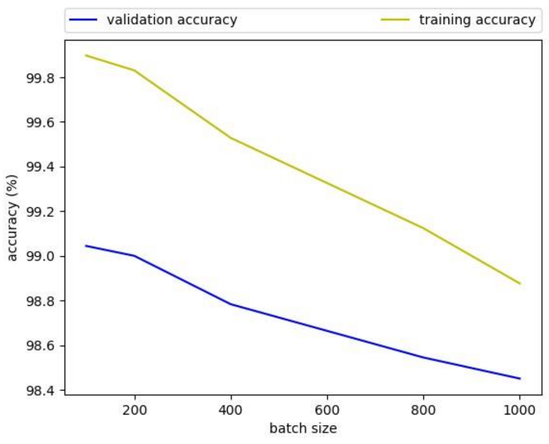


Fig. 9. Accuracy trend with batch size

Fig 10 represents the variation of training and validation loss with respect to learning rate. The decreasing trend in the curve indicates that the model improves over the epochs and thus loss decreases. Fig 11 indicates the training and validation accuarcy over diferent epoch numbers. As the learning rate increases, the accuracy increases and reaches a maximum. Further increase in learning rate results in negative trend for accuracy and loss. The maximum accuracy corresponds to the optimal value for learning rate.

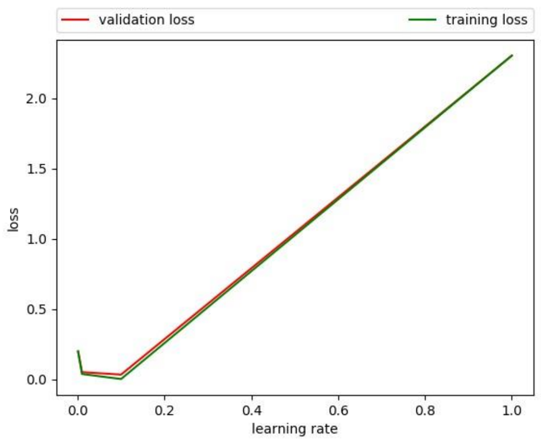


Fig. 10. Loss variation with learning rate

Fig 12 represents the variation of training and validation loss with respect to dropout probability. As the dropout probability increases, the validation reaches a minimum and again starts increasing. The minimum loss corresponds to the optimal value of dropout probability. Also, similar behavior can be deduced out of the accuracy curve in Fig 13.

IV. Results and Discussion

A. Optimum Hyperparameters

The range of values for hyperparameters have already been mentioned in ”Hyperparameters” section. The criteria

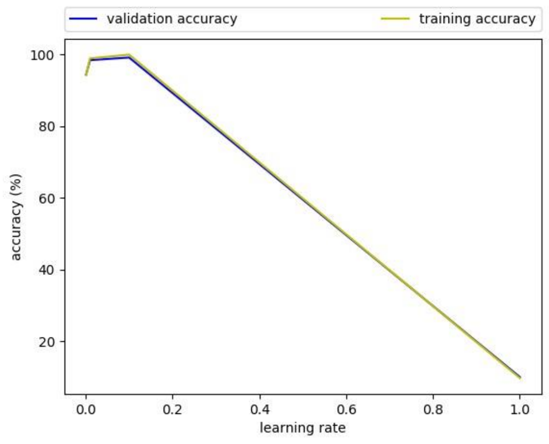


Fig. 11. Accuracy trend with learning rate

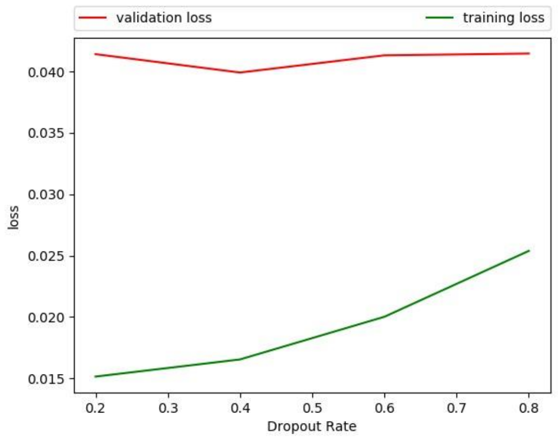


Fig. 12. Loss variation with dropout probability

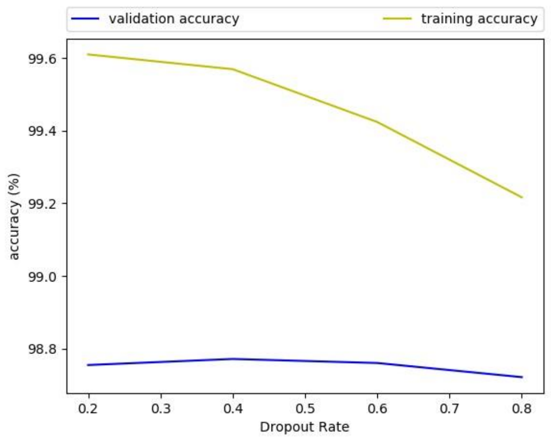


Fig. 13. Accuracy trend with dropout probability

for choosing the optimal value was solely based on the hyperparameter value that minimizes the loss function. In case of not observing this kind of behaviour(number of epochs for instance), the optimal value was choosen as the value where the validation curve starts to slightly deviate from the training curve(comparable to the concept of early stopping). The optimal value of these hyperparameters is specified in Table IV

TABLE IV
HYPERPARAMETERS FOR OPTIMUM PERFORMANCE

Hyperparameter	Optimal Value
Mini batch size	200
Epochs	40
Learning rate	0.1
Dropout probability	0.4

The best performing activation function is ReLU. The intution behind this is that tanh and sigmoid activation functions suffer from vanishing gradient effect and also the weight values are always limited since the maximum value that their differential version can attain is limited. In contrast, ReLU activation needs only handful non-zero values to continue the training process and their is no upper limit for the value this function can attain.

To support this assumption, Fig 14 provides the comparison of training accuracy across different activation functions and Fig 15 provides the validation accuracy achieved by each activation function under consideration. Table V provides the quantitative comparison of training loss and validation loss of different activation functions. But, it can be observed that the effect of convolution makes ReLU and tanh activation functions behave almost the same way. As mentioned earlier, due to vanishing gradient effect, the performance of sigmoid activation function is degraded.

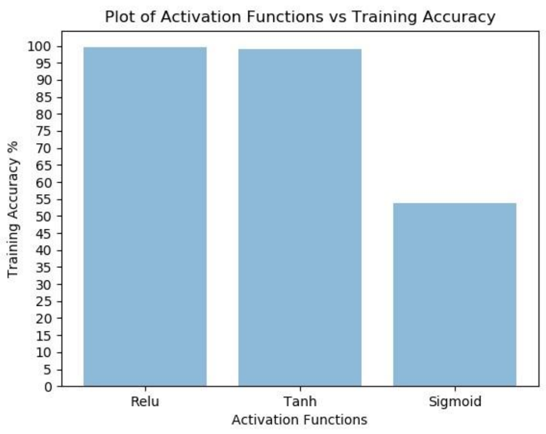


Fig. 14. Training accuracy trend with activation function

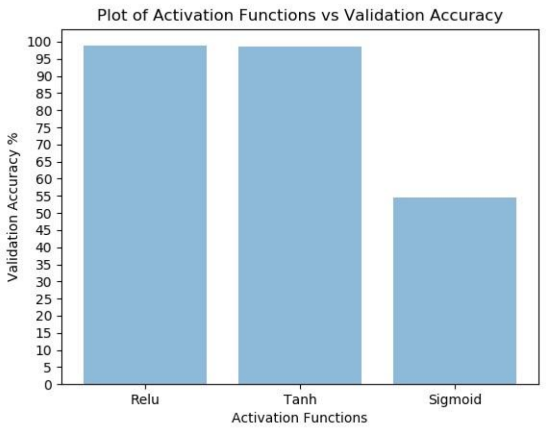


Fig. 15. Validation accuracy trend with activation function

TABLE V
LOSS FOR DIFFERENT ACTIVATION FUNCTIONS

Activation function	Training loss	Validation loss
ReLU	0.0158	0.0394
tanh	0.0305	0.0469
Sigmoid	2.1278	2.1257

B. Test Results

The final accuracy on testing set of 10000 samples with the aforementioned nework configuration, optimum hyperparameter values and ReLU activation function function is 99%.