**Abstract**

Image Classification is a well explored problem within the study of Deep Learning. The MNIST dataset has served as an important dataset used to benchmark the predictive powers of Neural Network Designs and such has served as inspiration for many of our decisions. In this report, we discuss the analytic methods and utilization of popular techniques used to maximize the performance on the provided dataset consisting of “normalized handwritten digits, automatically scanned from envelopes by the U.S. Postal Service”, that have been “deslanted and size normalized, resulting

in 16 x 16 grayscale images” (Le Cun et al., 1990).

All experiments for performance evaluation are in the context of Accuracy and Loss. For improved statistical significance we run each test of performance 10 times and report statistically relevant measures such as the average, max, min, and standard deviation when comparison of performance is discussed. For graphs we select a representative plot to visually display the results.

**Neural Network Design**

For our neural network we’ve made several design choices that have led to performance exceeding 90% accuracy for each of our networks.

*Number of neurons*

When choosing the size of layers to accurately model the problem we use empirically-derived rules-of-thumb. In particular, it’s well known that in order to reduce the chance of overfitting and chose a neural network of adequate size to model a problem, the number of neurons in the hidden layer should ideally lie within the range neuron size for the input and output layer. In this case [10, 256]. The size of output layer and input layer respectively.

*Activation Functions*

Activation functions allow us to model non-linear properties for modeling non-linear problems. The choice of activation functions is important because during the learning the backpropagation algorithm calculates the gradient of the activation function, so we can pass the maximum amount of the error though the network during back-propagation if we use ReLU which always has a derivative value of 1 or 0. over Sigmoid and Tanh because the max derivative of Sigmoid is 0.25, Tanh on the other hand has a max derivative of 1. This implies that during our backpropagation algorithm that ReLU consistently learns faster. In an unconstrained problem, ReLU in every layer proves to be empirically superior for learning. We introduce ReLU as the first activation function in each network as the first hidden layer. In this problem we explore the option of considering Sigmoid and Tanh in the hidden layers.

For the final layer SoftMax is chosen because we’d like our classifier to calculate a probability distribution used to calculate the loss and that determines the error that our backpropagation will exploit to correct the weights. The goal is that as the certainty of a classification increases, the corresponding class probabilities should decrease.

***Fully Connected Neural Network***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer # | Type | Neurons | Kernel Size | Activation | Input Shape |
| 1 | Input | 256 |  |  | 256 |
| 2 | Fully Connected | 128 |  | ReLU |  |
| 3 | Fully Connected | 128 |  | Sigmoid |  |
| 4 | Fully Connected | 128 |  | Tanh |  |
| 5 | Output | 10 |  | SoftMax |  |

***Locally Connected Neural Network (No Weights Shared)***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer # | Type | Neurons | Kernel Size | Activation | Input Shape |
| 1 | Input | 256 |  |  | 16 x 16 x 1 |
| 2 | Locally Connected 2D | 32 | 3 x 3 | ReLU |  |
| 3 | Locally Connected 2D | 64 | 3 x 3 | Tanh |  |
| 4 | Fully Connected | 128 |  | Sigmoid |  |
| 5 | Output | 10 |  | SoftMax |  |

***Locally Connected Neural Network (Convolutional Neural Network)***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer # | Type | Neurons | Kernel Size | Activation | Input Shape |
| 1 | Input | 256 |  |  | 16 x 16 x 1 |
| 2 | Convolutional 2D | 32 | 3 x 3 | ReLU |  |
| 3 | Convolutional 2D | 64 | 3 x 3 | Tanh |  |
| 4 | Fully Connected | 128 |  | Sigmoid |  |
| 5 | Output | 10 |  | SoftMax |  |

**Neural Network Performance**

*Fully Connected Neural Network*

*Loss Statistics*

count 10.000000

mean 0.282876

std 0.008211

min 0.273215

25% 0.276861

50% 0.280547

75% 0.287718

max 0.298098

*Accuracy Statistics*

count 10.000000

mean 0.937867

std 0.002934

min 0.933732

25% 0.935102

50% 0.938216

75% 0.939711

max 0.942202

*Locally Connected Neural Network*

*Loss Statistics*

count 10.000000

mean 0.206017

std 0.022262

min 0.184036

25% 0.190155

50% 0.197511

75% 0.212936

max 0.245063

*Accuracy Statistics*

count 10.000000

mean 0.950075

std 0.007014

min 0.939213

25% 0.945441

50% 0.951420

75% 0.956403

max 0.957150

*Convolutional Neural Network*

*Loss Statistics*

count 10.000000

mean 0.172386

std 0.010064

min 0.162192

25% 0.163783

50% 0.170172

75% 0.178379

max 0.190055

*Accuracy Statistics*

count 10.000000

mean 0.951769

std 0.004240

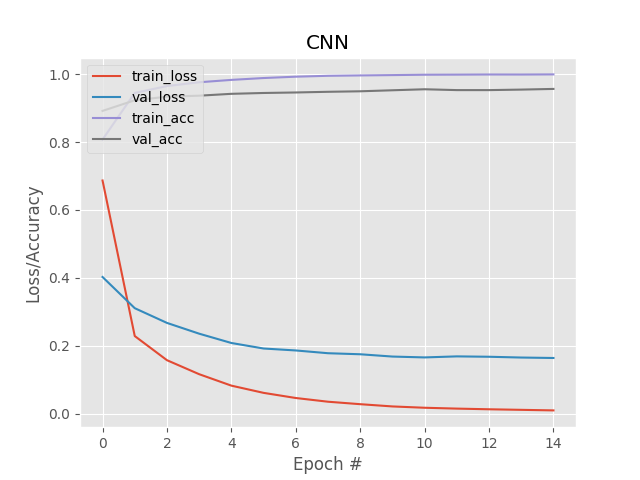
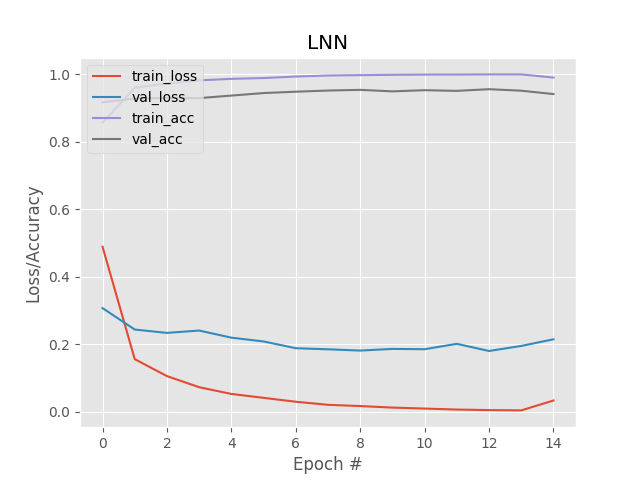
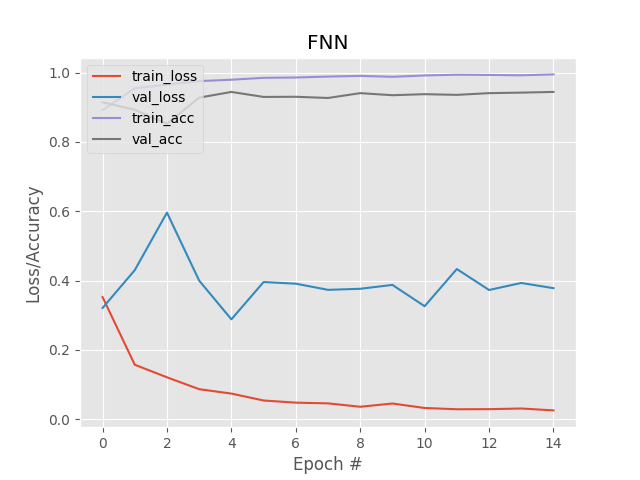
min 0.942701

25% 0.949801

50% 0.952167

75% 0.953662

max 0.957648

****

**For each of the networks, we analyzed how parameters initialization affects performance, then do experiments to verify our hypothesis and demonstrate the three cases based on your listed for each hyperparameter.**

***Parameter Initialization***

*Slow learning*

The magnitude of gradient change is subject to the size of the weights. Depending on the activation function small weights can make the gradient small. In training our neural networks, we utilize Adam which is a classical stochastic gradient descent. Therefore, we hypothesize that very small initial weights lead to subsequently slow learning as the magnitude of change at each iteration is very small.

FNN: weights are initialized to 0.001. Bias defaulted initialized as 0.

LNN: weights are initialized to 0.001. Bias defaulted initialized as 0.

# CNN: weights are initialized to 0.001. Bias defaulted initialized as 0.

*Effective Learning*

Glorot has been proven empirically to be the best parameter initialization for avoiding saturated regions by have a mean and unit variance. We utilize Sigmoid, and tanh in two hidden layers of our neural networks. They are shown to have effective gradients between the range of -1,1. So Glorot allows us to start in that range to learn as effectively as possible.

FNN: Weight Initialized using Glorot uniform distribution. Bias defaulted initialized as 0.

LNN: Weight Initialized using Glorot uniform distribution. Bias defaulted initialized as 0.

# CNN: Weight Initialized using Glorot uniform distribution. Bias defaulted initialized as 0.

*Too Fast Learning*

Large weights are sensitive to noise. They exasperate errors in computation. When noise is backpropagated it creates different values in our network then that of small weights. Initializing with high weight affects the learning in the beginning when the error is highest.

FNN: Weights in the first layer Initialized to 1.2. Bias defaulted initialized as 0.

LNN: Weights in the first layer Initialized to 1.2. Bias defaulted initialized as 0.

CNN: Weights in the first layer Initialized to 1.2. Bias defaulted initialized as 0.

***Learning rate***

*Slow learning*

*Learning rate has also an effect on our neural networks ability to learn from gradient descent. The corrections to be made are scaled by the learning rate. So a very low learning rate makes more iterations necessary to improve the performance of our neural network.*

FNN: Learning rate set to 0.00005.

LNN: Learning rate set to 0.00005.

CNN: Learning rate set to 0.00005.

*Effective learning*

For this we notice that our Neural network with default parameters performs well. The default learning rate for keras is 0.01. We note that the designers of keras are experts within the field of machine learning and must has picked an empirically sound number which is verified in our experimentation.

FNN: Learning rate set to 0.001.

LNN: Learning rate set to 0.001.

CNN: Learning rate set to 0.001.

*Too fast learning*

Learning that is too fast we stuck to multiples of the default keras learning rate. Finding that 0.1 is too fast. We test for a closer bound and find that 0.05 leads to unstable performance. A high learning rate impacts our stochastic descent and makes it difficult to converge by overstepping past local minima.

FNN: Learning rate set to 0.05.

LNN: Learning rate set to 0.05.

CNN: Learning rate set to 0.05.

***Batch Size***

Batch normalization works best with a batch size adequately large enough to sample the real distribution of the entire set. A reduced sample size for the calculations provides a less accurate representations of the mean and standard deviation of the batch in correspondence to the actual mean and standard deviation of the entire dataset.

An effective Batch size in this case is identified as 128. Our results notably degrade with a batch size of 2 and worsen by a large margin with a batch size of 1 or exceedingly large batch sizes. Small Batch sizes lose effectiveness because they converge to flat minimizers possibly due to noise in gradient descent, while large Batch sizes converge to sharp minimizers.

FNN: Effective Batch Size is 128. Ineffective Batch Size is 1.

LNN: Effective Batch Size is 32. Ineffective Batch Size is 1.

CNN: Effective Batch Size is 128. Ineffective Batch Size is 1.

**(4*) Momentum.* Commonly used momentum coefficient values are 0.5, 0.9, and 0.99. Using the best parameter initialization strategy, the best learning rate, and the best batch size you have found so far, experiment with the three different momentum values on the three networks you have and document the results. Explain the differences you have observed on the three neural networks you have.**

**We detail the analysis and regularization techniques aimed at improving generalization performance on the Validation set.**

*Ensemble network*

Our ensemble consists of 6 convolutions neural networks generated from 3 varying architectures. The first architecture is our CNN design presented in task 1. The Second CNN architecture sets a higher kernel size to possible capture different features / more broad features of the images. The final architecture design includes more neurons in the first hidden layers, with the goal of mapping more features.

The three architectures are used to build, then compile and train 2 CNN’s each. For our ensemble algorithm each CNN predicts the class probabilities for each image in the validation set. The probabilities are predicted for each CNN then summed and averaged. The ensemble then guesses the class corresponding to the highest probability of the summed probabilities.  
  
The ensemble net receives an accuracy max that exceeds 96%. The average accuracy of our ensembled neural network not only exceeds the average results of each of the CNN’s individually but also exceeds the max value for any individual CNN for any given run.

Ensemble Statistics

count 10.000000

mean 0.959492

std 0.001309

min 0.957648

25% 0.958396

50% 0.959392

75% 0.960140

max 0.961634

Individual Convolutional Statistics

0 1 2 3 4 5

count 10.000000 10.000000 10.000000 10.000000 10.000000 10.000000

mean 0.949128 0.949726 0.952965 0.951719 0.950573 0.952566

std 0.005415 0.001971 0.004281 0.002994 0.004127 0.003466

min 0.941206 0.947185 0.944694 0.946188 0.944694 0.945690

25% 0.943822 0.947808 0.950922 0.949552 0.947932 0.951420

50% 0.950673 0.950174 0.951918 0.952167 0.950922 0.953164

75% 0.952915 0.951046 0.956901 0.954410 0.952292 0.954534

max 0.956153 0.953164 0.958146 0.955157 0.958641 0.958146

*Dropout*

With dropout we could not achieve superior validation accuracy. In fact, it seems that the accuracy of our model degrades as we increase the probability of dropout. We believe this is because our models generalize well because our model is “relatively small” with respect to the dataset so regularization isn’t necessary and doesn’t yield higher accuracy because overfitting isn’t so prevalent. We can verify this conclusion also by looking at the initial neural networks graphs, we see that the loss does not increase as the amount of epochs increase once it reaches its convergence region.

*L1 regularization*

Our previous conclusion regarding regularization by analyzing dropout for the individual networks seems to be supported because L1 regularization also could not achieve superior validation accuracy. In fact, it also seems that the accuracy of our model degrades as we increase the effects of L1 regularization penalty. Noting that the model has not reached convergence as quickly with regularization than without we increase the epochs (training time) and found that the network indeed even after convergence does not yield a higher generalization performance. It does seem that the network has reduced its overfitting of the training set by a marginal amount ~ 1%, but at the cost of lower overall validation accuracy of ~3%.