

MiniProject 3: Classification of Image Data

David Schrier	260827890
Theo Janson	260868223
Jacob McConnell	260706620

Abstract

This assignment explores the use of Multi-Layer Perceptron architecture and Neural Networks for multi-class classification on the MNIST dataset. We compare different activation functions, network structures, and regularization methods to classify images of numerical "handwriting". We learned the hidden layers allow for better classification, that ReLu was the most effective activation function and that training with smaller batches while slower was most effective.

Keywords: Neutral Networks, Multi-Layer Perceptrons, Gradient Descent, Activation Function, Backpropagation

1. Introduction

In this project we compared several different MLPs to see how well they would work at classifying handwritten numerals to their correct digit. These handwritten digits are part of the well-known MNIST dataset. This dataset's first use was in the seminar computer vision paper published in 1998 "Gradient-Based Learning Applied to Document Recognition" by Yoshua Bengio, Yann LeCun and others [1]. They created this dataset out of previous NIST (National Institute of Science and Technology) data sets of hand written numerals. They did this to create a data set of handwriting that was independently distributed by quality so that they could train and compare machine learning models in a way that let them draw meaningful conclusions [1]. Like this paper we compared different MLP architectures using this data set to find out which would work best. To do so we created our own MultiLayer Perceptron class from scratch. We then used this class to construct MLPs with different structures and activation functions. The first step we took in this project was normalizing our data and reshaping it so that the models could be consistently trained with simple and standard inputs with values that did not break the model. We found that the ReLu activation function worked best compared to other activation functions, that we got better results with more hidden layers (but seemingly diminishing returns) and with smaller batches when we trained.

2. Model Descriptions

This section will detail both machine learning methods and their background.

2.1 MultiLayer Perceptrons

A multi-layer perceptron, or deep neural network was implemented. Each layer was fully connected. Various architectures were tested by varying the number of layers, and neurons in each layer. Weights to each layer were initialized using normal random distributions

divided by the square root of the number of inputs to the layer. This allows the standard deviation of the layer's weights to be fairly small, so that each number is close to 0, but not too close. This prevents exploding and vanishing gradients during forward propagation. Xavier initialization was also explored. This technique has similar benefits as provides relatively low upper bounds to the magnitude of the weight's initialization. A vector of inputs, or features is propagated through the network, and weighted by each hidden layer's weights and a layer bias.

2.2 Activation Function

Each hidden layer is appended by an activation function, which squishes the output of the hidden layer such that it becomes bounded. Hyperbolic tangent and Sigmoid activation functions bound the output between -1 and 1, and Rectified Linear Unit sets all negative outputs to 0. These activation introduce non linearity on the output of the layers to allow for the mapping of complex decision boundaries on the data.

2.3 Softmax Cross Entropy

Before computing the loss between between the network output and the true labels, the output is passed through a Softmax layer which transforms the final layer's output such that they add to 1. This output and the one hot representation of the true label can then be interpreted as probabilities and compared. Cross Entropy Loss is an indication of distance between the predicted and true distributions.

2.4 Mini-Batch Gradient Descent

To train the network, the weights of each layer need to be updated to correct for the Cross Entropy loss. The gradient of the weights of each layer is calculated by propagating the gradient of each subsequent layer backwards through the network by multiplying the gradient of each layer, starting with the gradient of the loss. This process implements the chain rule on the gradient of the hidden layer's weights

The weights for each layer are then updated according to gradient descent. The convergence of gradient descent is largely determined by the learning rate λ . L1 and L2 regularization techniques were tested, so that large weights in the hidden layers would decay more quickly: Mini-batch gradient descent allows for a number of training examples to propagate through the network and update the weights after each batch pass.

3. Datasets

The image data used was the MNIST database, which as 3Blue1Brown eloquently described, is the 'Hello World of neural networks. The dataset contains 60,000 training images and 10,000 testing images of handwritten digits. The black and white images are represented by 28x28 matrices of pixels. The darkest pixels have values of 0 and the brightest ones have values of 255. The pixels were flattened into a vector and normalized to small values to produce well behaved forward propagation through the network.

4. Results

Various tests were conducted. Table 1 lists results required by the project description. These involve varying the number of hidden layers, testing different activation functions, and exploring the effect of regularization and of data normalization. Tables 2-5 explore the effects of varying L1 and L2 regularization, mini-batch size and the gradient descent learning rate. Finally, a series of special tests were done. The first case tested a seven-layer deep neural network with 25 neurons on each layer. The second tested a 500 neuron wide, single layer network. The third explored the effect of Xavier initialization, rather than simple random initialization. Finally, we compared the performance of the multilayer perceptron to a Convolutional Neural Network (CNN) with an architecture implemented using PyTorch, as can be seen below:

CNN Architecture	
	Convolutional Layer: 32 3x3 kernels
	ReLU Activation
	Convolutional Layer: 64 3x3 kernels
	ReLU Activation
	Max Pooling: 2x2 kernels
	Convolutional Layer: 64 3x3 kernels
	ReLU Activation
	Convolutional Layer: 64 3x3 kernels
	ReLU Activation
	Flattenning layer
	Linear Layer

Test Index	Hidden Layer	Activation	Reg.	Reg. Constant	Train Acc.	Test Acc.
1	None	None	None	0	92.5%	92.1%
2	128	ReLU	None	0	99.0%	97.9%
3	128, 128	ReLU	None	0	99.7%	98.2%
4	128, 128	Sigmoid	None	0	96.7%	96.1%
5	128, 128	Tanh	None	0	99.6%	97.7%
6	128, 128	ReLU	L2	0.1	97.9%	97.3%
Un-Normalized	128, 128	ReLU	None	0	9.8%	9.7%

Table 1: Required Results: Learning Rate = 0.1, Mini-Batch Size = 50, Epochs = 15

Test Index	Hidden Layer	Reg.	Reg. Constant	Activation	Train Acc.	Test Acc.
7	64, 64	L2	0.001	ReLU	97.4%	97.0%
8	64, 64	L2	0.1	ReLU	96.1%	95.9%
9	64, 64	L2	10	ReLU	11.4%	11.6%
10	64, 64	L1	0.001	ReLU	97.1%	97.1%
11	64, 64	L1	0.1	ReLU	91.3%	91.4%
12	64, 64	L1	10	ReLU	11.6%	11.4%

Table 2: Regularization Results: Learning Rate = 0.1, Mini-Batch Size = 50, Epochs = 5

Test Index	Hidden Layer	Reg.	MiniBatch	Activation	Train Acc.	Test Acc.
13	64, 64	None	10	ReLU	98.5%	96.8%
14	64, 64	None	100	ReLU	97.2%	96.6%
15	64, 64	None	1000	ReLU	90.0%	91.5%

Table 3: MiniBatch Results: Learning Rate = 0.1, Epochs = 5

Test Index	Hidden Layer	Reg.	MiniBatch Size	Train Acc.	Test Acc.
16	64, 64	10	50	9.7%	9.8%
17	64, 64	1	50	92.2%	91.5%
18	64, 64	0.1	50	98.4%	97.1%
19	64, 64	0.01	50	92.3%	93.2%

Table 4: Learning Rate Results: Regularization = None, Reg. Constant = 0, Activation = ReLU, Epochs = 5

Test Index	Hidden Layer	Reg.	MiniBatch Size	Train Acc.	Test Acc.
16	64, 64	10	50	9.7%	9.8%
17	64, 64	1	50	92.2%	91.5%
18	64, 64	0.1	50	98.4%	97.1%
19	64, 64	0.01	50	92.3%	93.2%

Table 5: Learning Rate Results: Regularization = None, Reg. Constant = 0, Activation = ReLU, Epochs = 5

Model	Hidden Layer	MiniBatch Size	Train Acc.	Test Acc.
Deep Network	25, 25, 25, 25, 25, 25, 25	50	96.2%	95.6%
Wide Network	500	50	97.4%	91.5%
Xavier Initialization	64 64	50	97.6%	96.3%
CNN	N/A	N/A	N/A	99.3%

Table 6: Special Test Results: Regularization = None, Reg. Constant = 0, Learning Rate = 0.1, Activation = ReLU, Epochs = 5

5. Discussion and Conclusion

The first test was done on a perceptron network with no hidden layers. This architecture does not include non-linearity introducing activation functions, and is essentially linear regression. Linear regression performed surprisingly well. Adding a single hidden layer and activation to the network considerably increased performance due to the non-linearity introduced. Adding a second layer further increases performance. Early layers can be interpreted as colour gradient detectors, and the subsequent layers added are used to detect more complex shapes and structures. The data is relatively simple, and each digit is pretty distinct from other digits, so adding a second layer does not considerably increase accuracy. The increase is due to better differentiation between '0' and '6'.

The effect of various activations was explored. ReLU activation produced the best accuracy on both the training and test sets. Sigmoid and Tanh activations performed ever slightly worse, but still produce accurate classifiers. Sigmoid and tanh activations may have produced slightly worse performance because of the loss of information by squishing large weights together. Another interesting note is that when testing different structures of MLPs with sigmoid activation functions we found that it worked better with a single hidden layer rather than two hidden layers. We aren't sure why this is the case.

We also tested the MLP with and without regularization and using different regularization constants. We got better results without regularization and when we used regularization we got better results with a smaller regularization hyper parameter. This might indicate that we had very good data to train our networks on and that we didn't have any large weights that needed to be penalized. L2 regularization (ridge regression) performed slightly better than L1 regularization (lasso regression). This may be because L2 penalizes coefficients evenly while L1 penalizes more heavily, resulting in stronger regularization. A test was done using unnormalized image data. Many pixels in such an image are represented by value on the order of 255. These large values explode the gradient as the weights are multiplied out of proportion, producing a useless network.

Tests were done with varying mini batch sizes. The results show a clear trend as smaller batch sizes produce more accurate results, but this also increases the training time. As the batch size decrease, the training method resembles online learning, as each sample updates the weights.

We also tested the same MLP with different learning rates. The best learning rate we tried was 0.1. We think that for learning rates than 0.1 the steps in training the model are too small and the weights do not converge effectively to an optima. For larger steps the steps are too large. Gradient descent overshoots the optima and the model outputs essentially random results.

We ran simple tests comparing a wide, single layer network with 100 neurons, and a deep 7 layer network with 25 neurons. The former showed better performance as it detects complex structures in the data more efficiently. We also compared the random initialization to Xavier initialization which produced essentially the same results. Finally, our MLP to a CNN which was previously implemented for the course ECSE 415 - Intro to Computer Vision. This model had great accuracy, but took very long to train due to the many convolutional and max pooling layers. CNNs are more suited for forming predictions on image data, so one might expect stronger performance.

6. Statement of Contribution

Theo wrote most of the implementation of the MLP, the minibatch tester, the cost functions, and the activation functions and contributed to the report. David contributed towards the report, ran most of the tests and their respective data visualization and plots required in the project. Jacob helped debug the MLP code so that certain required tests were possible, carried out some of the tests and contributed to the report.

References

- [1] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.