Programming Assignment 2

Hayleigh Sanders RIN #661195735

Introduction

The following project is an implementation of a three layer neural network to classify handwritten digits from 0 to 9. ReLu was used as the activation function and softmax was used as the output function. The neural network achieved a low classification error rate after 50 iterations.

Theory and Setups

Multilayer Neural Network

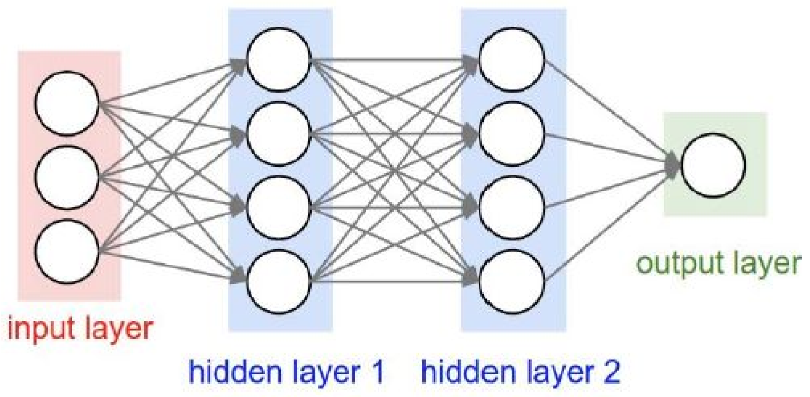


Image Credit: Andrej Karpathy, CS231n

A multilayer neural network consists of an input layer, a number of hidden layers and an output layer, with a weight matrix between each layer. The weights are initialized to random values and an initial prediction $\hat{y}[m]$ for each x[m] input is produced. This output is used to produce the output gradient, which is back propagated through each layer of the neural net to obtain the weight gradients for each layer. Based on these gradients, the weight matrices are updated and the process is repeated for a number of iterations. This project will implement a multilayer neural network with two hidden layers.

Loss Function Used For Training

The loss function utilized in this project is the negative log conditional likelihood (cross-entropy) loss function for multiclass regression given by:

$$L(y,\hat{y}) = -\frac{1}{M}\sum_{i=m}^{M}(y^m \log(\hat{y}^m) + (1-y^m)\log(1-\hat{y}^m))$$

The loss is averaged over a training set of size M samples. Y is the data label and Y-hat is the output produced by the neural network.

Forward Pass

Starting at the input layer, for each x[m] in the training data set, its corresponding output $\hat{y}[m]$ is computed from the current weights W = [W^1, W^2, W^3] and their biases W^1_0, W^2_0, W^3_0 with the following:

$$H^1[m] = ReLu\left((W^1)^T x[m] + W^1{}_0\right)$$
$$H^2[m] = ReLu\left((W^2)^T H^1[m] + W^2{}_0\right)$$
$$\hat{y}[m] = softmax\left((W^3)^T H^2[m] + W^3{}_0\right)$$

Where the ReLu function is the activation function given by,

$$ReLu(z) = max(0, z)$$

ReLu is used as the activation function instead of the sigmoid function because in a multi-layer network, because the amount of error back-propagated through the neural network becomes large, with the weight gradients becoming small, so a thresholding activation function is needed to avoid saturation.

And softmax is the output function given by,

$$\sigma_M(z) = \frac{exp(z)}{\sum_{k'=1}^{K} exp(z)}$$

This process is split into multiple steps:

Z1 = W^1X + $W_{0,1}$
H1 = Relu(Z1)

Z2 = W^2H1 + $W_{0,2}$
H2 = Relu(Z2)

$Z3 = W^3 H2 + W_{0,3}$
Y_hat = softmax(Z3)

Backward Pass

Starting at the output layer, the output gradient is computed and passed over the neural network layers. The weight gradients were found recursively by the following process:

Find the error at the output layer:
dZ3 = Y_hat - Y

Find the gradients at the output layer:
$dW^3 = (1/M)*dZ3*H2^T$
$dW_{0,3} = (1/M)*\text{sum}(dZ3)$

Backpropagate through the second layer:
$dH2 = W^{3T}*dZ3$
dZ2 = dH2*dReLu(Z2)

Find the gradients at the second layer:
$dW^2 = (1/M)dZ2*H1^T$
$dW_{0,2} = (1/M)*\text{sum}(dZ2)$

Backpropagate through to the first layer:
$dH1 = W^{2T}*dZ2$
dZ1 = dH1*dReLu(Z1)

Find the gradients at the first layer:
$dW^1 = (1/M)*dZ1*X^T$
$dW_{0,1} = (1/M)*\text{sum}(dZ1)$

dReLu(z) is the derivative of the ReLu function given by,

$$dReLu(z) = \begin{cases} 0 \; if \;\; z < 0 \\ 1 \; if \;\; z > 0 \\ undefined \; if \;\; z = 0 \end{cases}$$

M is the total number of samples in the data set.

The weights and their biases for each layer are then updated using the gradients found in the backward pass process with the following:

$W^1 = W^1 - \eta dW^1$
$W_{0,1} = W_{0,1} - \eta dW_{0,1}$

$W^2 = W^2 - \eta dW^2$
$W_{0,2} = W_{0,2} - \eta dW_{0,2}$

$W^3 = W^3 - \eta dW^3$
$W_{0,3} = W_{0,3} - \eta dW_{0,3}$

Where η is the learning rate. It was decreased after each iteration so that the weights converge more quickly.

Stochastic Gradient Descent

Stochastic gradient descent was used for the backpropagation process. The forward pass, backward pass and gradient updating steps were performed in batches of 50 data points, with each batch being a subset of the data points in the data set after the set was shuffled randomly. The stochastic gradient descent process is given as,

$$W_k^{t+1} = W_k^{t+1} - \eta \left[\frac{1}{S} \sum_{x[m],y[m] \in D} \frac{\partial L(x[m]y[m]W)}{\partial W} \right]$$

Where D is each batch.

Neural Network Architecture

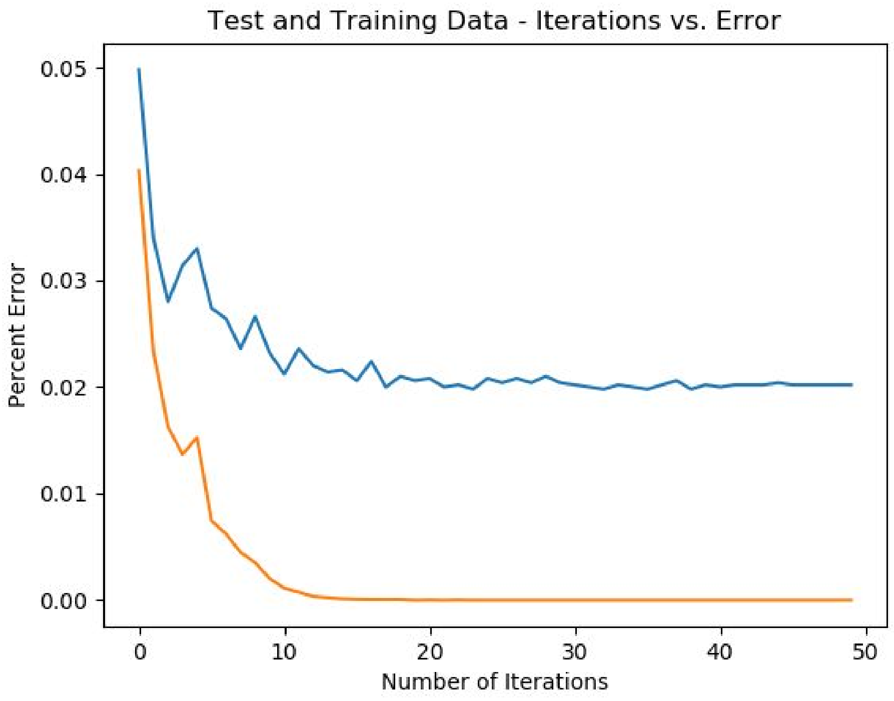
A multilayer neural network was constructed to classify images of handwritten digits from 0 to 9.The neural network has an input layer, two hidden layers and an output layer. The two hidden layers each have 100 nodes and the output layer has 10 nodes, corresponding to the predicted digit class output from 0 to 9. ReLu was used as the activation function between the hidden layers. Softmax was used as the output function because it produces a 1x10 vector of probabilities for each class for each input vector. Each data point was a 784x1 vector corresponding to a flattened 28x28 pixel image, which was normalized by dividing each pixel by 255.

Model Hyperparameters

The hyperparameters were the learning rate η, minibatch size, number of nodes for each hidden layer, and the number of iterations (epochs) the weights were trained for. Each hyperparameter was tuned by either increasing or decreasing it a small amount and checking if the total classification error decreased. The values that minimized classification error were chosen. The learning rate was set to .5, the minibatch size for stochastic gradient descent was set to 50, the number of nodes for each hidden layer was 100, and the number of epochs was 50.

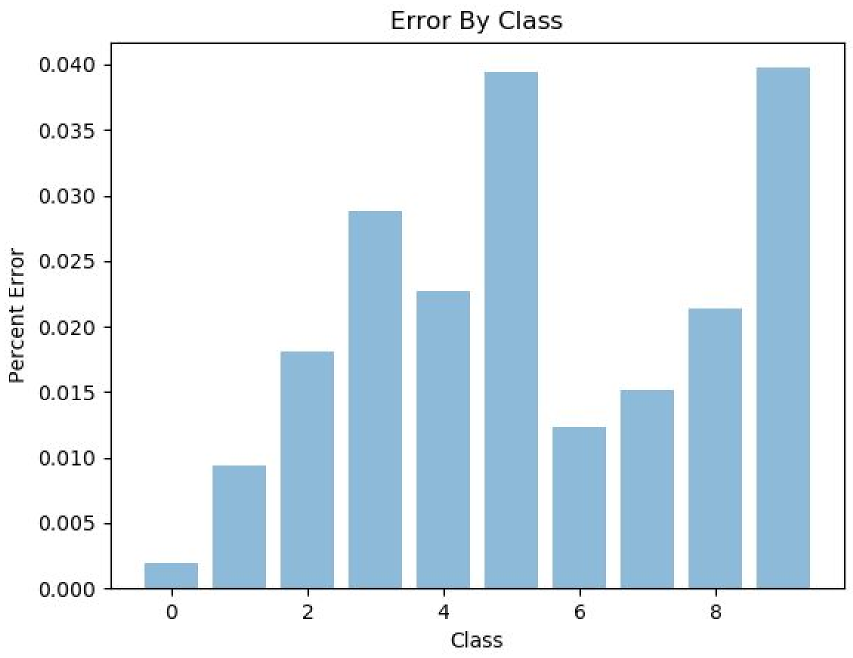
Experimental Results

The neural network was trained with the backpropagation method with stochastic gradient descent described above, which was repeated for 50 iterations (epochs). The weights were initialized to small Gaussian random values and updated with each backpropagation iteration. At each epoch, the output Y was computed with the forward propagation process and its accuracy was determined by summing the total number of accurately classified digits over the total number of samples in the data set. This process was performed for the training and testing data sets, and the error is plotted below for each epoch. The error was found by using 1 - accuracy.



The testing data error is shown in blue, and the training data error is shown in orange. Both training and testing error decreased as the iterations increased, meaning the weights were successfully updated.

The classification error for each digit is shown below. The error for each digit was computed from 1 - (correctly classified)/(total occurrences of digit).



5 and 9 appeared to have the most classification error, with 0 having the least.

Conclusion

The neural network appeared to successfully classify the digits with a low error rate.