CPSC 340 Machine Learning Take-Home Final Exam (Spring 2020)

Question 1 Answer

Name, Student Number: Rishabh Singal, 18802141
ugrad ID: z7b0b

1 Introduction

Three sentences describing the MNIST classification problem.

Answer: MNIST contains 28x28 pixes images of handwritten, and sometimes obscure, digits from 0-9. The MNIST dataset contains 60,000 images in total, out of which 50,000 are used in this assignment for training while the other 10,000 are used for training. The goal of machine learning models is to try and classify each test image as accurately as possible.

2 Methods

2.1 KNN

Three to four sentences describing the particulars of your KNN implementation, highlighting the hyperparameter value choices you made and why.

Answer: Since KNN (based on euclidean distance) is a non-parametric model and since MNIST dataset is massive, I ran 5-fold cross validation algorithm with k-neighbour values of 1 to 10 on 5000 examples so that no value errors are thrown during the computation of each row's euclidean distance with other ones. Lowest cross-validation error was observed for k-value of 3 and this k-value was used to train the KNN model with 20,000 randomly chosen training examples on Google colab which yielded a test error of 3.2%.

2.2 linear regression

Three to four sentences describing the particulars of your linear regression implementation, highlighting the hyperparameter value choices you made and why.

Answer: I tried several linear regression models in the following order and determined the best hyperparameters for each model using 5-fold cross-validation on various combinations of inputs:

1. Least Squares Linear Regression with L2: This was a regular least squares model which predicted the class by rounding the output value to the nearest integer. As expected, it yielded poor results with the test error being 75.54%.
2. Robust Linear Regression: The weight vector for each class was computed by solving the least squares equation with robust L0 regularization. The prediction was the max index value in each row of output matrix. It yielded test error 16.72%.
3. Least Squares Classifier: This was the same as robust linear regression model except it used L2 regularization and achieved test error of 14.6%.

4. Kernel Linear Regression Polynomial: I computed the polynomial kernel using a small sub-set of training examples (5,000) because any larger and I exceeded my computer's RAM. Using p=3 and L2 $\lambda = 1$, I achieved a test error of 4.3%. I noticed that the presence of λ didn't affect the results much.
5. Kernel Linear Regression RBF (**Best** linear regression model): Finally, I used RBF kernel to train the model and achieved a very low test error of 3.62% with $\sigma = 10$ and L2 $\lambda = 0.01$.

2.3 SVM

Three to four sentences describing the particulars of your SVM implementation, highlighting the hyperparameter value choices you made and why.

Answer: I implemented multi-class SVM with both sum and max losses with L2-regularization. Stochastic gradient descent (with momentum) with a minibatch of 1 and epoch of 5 was used to fit the final model, although epoch of 0.1 was used during cross-validation to select the hyperparameters to select the L2 λ of 0.01 for sum loss and 0.1 for max loss. Sub-gradient approach was used to compute gradients. For this application, multi-clas SVM sum loss yielded better test error of 9.46% compared to 14.22% obtained using max loss.

2.4 MLP

Three to four sentences describing the particulars of your MLP implementation, highlighting the hyperparameter value choices you made and why.

Answer: My MLP implementation in this assignment is essentially a direct carryover from assignment 6 with a small difference except the fact that I'm using Xavier initialization (with a divide by 2 term added in the denominator) to initialize the layer weights and using SGD with momentum. I ran the model with gradient descent only once which achieved a test error of 4.65%. Also I ran the model with SGD with several inputs of epochs (10, 20, 30, 40) and minibatch sizes (100, 500, 1000, 2500) and observed that epoch of 40 with minibatch size of 2500 worked quite well combined with SGD with momentum yielded a low test-error of only 3.28%. The training error was only slightly lower at 2.774%.

2.5 CNN

Three to four sentences describing the particulars of your CNN implementation, highlighting the hyperparameter value choices you made and why.

Answer: In order to create the CNN from scratch, I received significant help from Mahan Fathi's Stanford CS231N Assignment 2 repo (`https://github.com/MahanFathi/CS231/tree/master/assignment2/cs231n`). However, I do completely understand how the backpropagated gradients are derived using computation graphs and how the forward passes of each layer works. Each CNN layer's (convolutional, relu, max-pool, fully-connected and softmax) forward pass takes input matrix X, filter W and bias B (if applicable) as inputs and produces output layer. Each CNN layer's backward pass obtains the gradient flowing from upstream the network as inputs and computes gradients with respect to input matrix X, filter W and bias B (if applicable). Convolution forward pass is done by sliding the filter across the width and height of each input image matrix (1x28x28). A class is defined which constructs a CNN consisting of convolutional - relu - max-pool - fully-connected - relu - fully connected - softmax and initializes the weight and bias matrices similarly to neural_net.py except the convolutional layer filter has a 3-dimensional shape. The dimension of each convolutional filter is 5x5 and 8 filters are used. The size of hidden fully-connected layer is 128. I fit the function as is normally done in CPSC340 by flattening the weights and passing them to the stochastic gradient descent function I wrote (findMinSGD).

However, my loss term kept exploding with different combinations of SGD learning rate, filter dimensions and number of filter and and I could not achieve a result due to time constraints. Perhaps, flattening the filter matrices for SGD isn't yielding good results in this model.

3 Results

Model	Their Error	Your Error (%)
KNN	0.52	3.2
linear regression	7.6	3.62
SVM	0.56	9.46
MLP	0.35	3.28
CNN	0.23	No Result

4 Discussion

Up to half a page describing why you believe your reported test errors are different than those provided (and "detailed" on the MNIST website).

Answer: Following are potential sources of differences for each model:

1. KNN: We're using Euclidean distance to compute the distances between training examples. However, a better measure might be to combine density- and euclidean-based methods to achieve clusters with stronger relationships which would lower the test error.
2. Linear Regression: My kernal RBF linear regression actually performs better than the provided value, but the non-kernel ones did not perform as well which makes sense provided that the data is not contained in a linear sub-space.
3. SVM: They're likely using better stochastic gradient methods such as Adam to fit the training data, whereas I'm just using SGD with momentum which is not as smart as Adam.
4. MLP: Initialization, activation technique, and number of hidden layers make a big difference in the performance of MLP models. However, since the provided MLP code in assignment 6 uses sigmoid activations, adding more than 1 layer will cause problem of vanishing gradients. Hence, better error can be achieved by using smarter initialization techniques for weight matrices (area of active research), changing sigmoid activation to relu, and using Adam SGD method for fitting data.
5. CNN: I did not actually achieve a result for CNN model unfortunately due to time restrictions, but it makes sense that the provided CNN test error is lower than provided MLP test error as CNN actually learns increasingly complex features of the input image such as lines (horiontal, vertical, diagonal), combinations of lines, and patterns as the number of convolutional layers are increased. If a dog image is input into a MLP trained on MNIST digits, it will still confidently predict a digit value, but CNN won't predict anything confidently because the patterns in the dog image do not match the patterns in the digit images.

5 References

1. `https://github.com/MahanFathi/CS231/tree/master/assignment2/cs231n`