Skeleton for Question 1 Answer

1 Introduction (3 *points*)

Three sentences describing the MNIST classification problem.

Answer: Each training example (X) is the flattened pixel value of a hand-written digit picture. Each y is the corresponding digit (0-9) in the picture. The classification task is to recognize the digit in a new hand-written picture. There are 50000 training data, 10000 validation data and 10000 test data.

2 Methods (40 *points*)

2.1 KNN (8 *points*)

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

Answer: For the implementation, I used the code from previous assignment. I speed up the algorithm by replacing the nested for-loop in cosine distance with numpy array operations. I used validation set to choose the hyperparameter k. I also used the validation set to compare between the cosine distance and the euclidean distance and found that cosine distance performs better generally. I chose $k = 4$ by testing on the validation set. Choosing the hyperparameter that gives the lowest validation error will probably result in over-fit on the validation set though.

2.2 linear regression (8 *points*)

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

Answer: For the implementation, I used the multi-class softmax loss function from previous assignment. Compared with one-vs-all logistic loss which train each class' W independently, softmax encourages calibrations among all columns. I used the log-sum-exp trick from the Piazza post (https://piazza.com/class/kenhpni6f94x4?cid=804) to deal with the overflow issues for both the exponential and the logarithm functions. For the hyperparameters, I used the validation set to tune the max iterations. I chose max iteration = 300, which gives the lowest validation error.

2.3 SVM (8 *points*)

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

Answer: For the implementation, I used the multi-class SVM. The main changes are in funObj function. I used a nested for-loop to add up the loss function and the gradient at the same time. For the calculation of the gradient, I used the sub-gradient method from the Piazza post (https://piazza.com/class/kenhpni6f94x4?cid=770). And then I used the validation set to tune the hyper parameters lambda and max iterations. As always, hyperparameters that give low validation errors tend to have larger Eapprox. In the end, using the method from the previous questions, I chose lambda = 0.01 and max iteration = 500.

2.4 MLP (8 *points*)

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

Answer: For the implementation, I used the neural net from a6 which has one layer and uses sigmoid activation function. From a6 I found that sigmoid performs the best among some commonly used activation functions by using scikit learn library so I'll keep using it here. I used the validation set to tune the hyperparameters lambda, maximum number of iterations and hidden layer size. Generally, validation error decreases with larger max iteration and hidden layer size. In the end, I chose lambda = 0.025 , max iteration = 1000 and hidden layer size = 150.

2.5 CNN (8 *points*)

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

Answer: A CNN generally has the following components: several convolution layers, with specified filters with different sizes and strides. There are also different methods to deal with the edges of the images, such as filling with zeros. And then there are usually one or more max pooling layers that capture the most important features per sub-images of a certain size. We can then flatten out the images into one array. Now we can use regular neural networks to classify the images. In my opinion, this is a composite of the methods we've implemented above: A CNN consists of MLPs, and each layer of the neural network has a classification model, like SVM.
There are many hyperparamters that we can tune. I would use the validation set as before and observe the how the error rate changes. But training one CNN model takes too long so I just used the default ones.
Citation: https://towardsdatascience.com/convolutional-neural-networks-from-the-ground-up-c67bb41454e1

3 Results (10 *points*)

Model	Their Error	Your Error (%)
KNN	0.52	2.87
linear regression	7.6	7.43
SVM	0.56	8.12
MLP	0.35	1.84
CNN	0.23	1.84

4 Discussion (7 *points*)

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: First of all, I did not perform any pre-processing on the image, such as normalization, reducing noise etc. So there could be messy parts of the images influences the learning.

Also, the distance function, loss function, regularization term and the number of layers I used in my implementation might be different from the ones that give the error rates on the website. I might be also missing some functions that can improve the performance, such as kernel for SVM. For CNN, there are more choices could be done, such as the filters, strides, filter sizes, pooling layer, activation function, classification functions, the number of the layers and all the hyper-parameters of those models.

I believe doing a cross-validation with a larger range for the hyperparameters could also help with finding the best ones. Also, I'd like to try using a different method to choose hyperparameters. However, since the dataset is pretty large, it takes too much time to iterate over all possible combinations.