CS 615 - Deep Learning

Assignment 5 -MLPs
Winter 2023

Introduction

In this assignment we will explore the design and use of multi-layer perceptrons for multi-class classification.

Allowable Libraries/Functions

As per usual, you **cannot** use any ML functions to do the training or evaluation for you. Using basic statistical and linear algebra function like *mean*, *std*, *cov* etc.. is fine, but using ones like *train*, *confusion*, etc.. is not. Using any ML-related functions, may result in a **zero** for the programming component. In general, use the "spirit of the assignment" (where we're implementing things from scratch) as your guide, but if you want clarification on if can use a particular function, DM the professor on slack.

Grading

Part 1	Theory	10pts
Part 2	ANN	40pts
Part 3	MLP	40pts
Part 4	MLP w/ Greedy Pre-training	10pts

Table 1: Grading Rubric

NOTE: Depending on your design decisions throughout this assignment, you may or may not see much difference in evaluation between each part. This is fine. The purpose of this exercise is largely to be able to explore design and hyperparameter choices and to be able to construct and train multi-layer perceptrons, optionally with greedy layer pretraining.

Datasets

MNIST Database The MNIST Database is a dataset of hand-written digits from 0 to 9. The *original* dataset contains 60,000 training samples, and 10,000 testing samples, each of which is a 28×28 image.

To keep processing time reasonable, we have extracted 100 observations of each class from the training datase,t and 10 observations of each class from the validation/testing set to create a new dataset in the files *mnist_train_100.csv* and *mnist_valid_10.csv*, respectively.

The files are arranged so that each row pertains to an observation, and in each row, the first column is the *target class* $\in \{0, 9\}$. The remaining 784 columns are the *features* of that observation, in this case, the pixel values.

For more information about the original dataset, you can visit: http://yann.lecun.com/exdb/mnist/

1 Theory

1. (5pts) In class and the lecture notes, we provided the gradient of the *tanh* function without actually walking through the derivation. For this assignment's only theory question, show the work on how the partial derivative of the tanh function, $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, is $\frac{\partial g(z)}{\partial z} = (1 - g^2(z))$

   Answer: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

   $=> \frac{\partial g(z)}{\partial z} = \frac{(e^z + e^{-z})^2 - ((e^z - e^{-z})^2)}{(e^z + e^{-z})^2}$

   $=> \frac{\partial g(z)}{\partial z} = 1 - \frac{((e^z - e^{-z})^2)}{(e^z + e^{-z})^2}$

   $=> \frac{\partial g(z)}{\partial z} = g^2(z)$

2. (5pts) Generative Adversarial Network's (GANs) *generative* network often have an objective function in the form $J = -ln(\hat{y})$. What is the gradient of this objective function with regards to a single estimation $\hat{y}$. That is, what is $\frac{\partial J}{\partial \hat{y}}$?

   Answer: $J = -ln(\hat{y})$

   $=> \frac{\partial J}{\partial \hat{y}} = -\frac{1}{\hat{y}}$

2 Artificial Neural Network

In the previous assignment we used a simple architecture to do multi-class classification. Now let's add in a hidden layer, so that we have an *artificial neural network*. The core architecture is as follows (FC Layer refers to a Fully-Connected layer):

Input Layer → FC Layer → Activation Layer → FC Layer → Activation Layer → Objective

As you can see, some design choices and hyperparameter decisions are up to you. Here's some things you'll want to think about:

- How many outputs for the first hidden layer?
- What activation functions to use?
- What objective function to use?
- Other decisions related to learning rate, termination, overfitting, batches, etc..

In addition to keeping track of the training dataset's evaluation each epoch, you should now also keep tracking of the **validation dataset's** evaluation each epoch. This will allow us to observer the possibility of overfitting.
Your report should include:

- Any and all design decisions (as described above).

  For the following problem after careful observation I choose 100 epochs to be the most optimum number of epochs. Apart from this I implemented early stopping for the network to not over-fit the data. Apart from this the model did not perform very well when the dropout of 0.5 and 0.2 were applied after the internal activation layer. For the internal activations I used ReLU Layer, which would be an optimum choice for giving a linear value in the range more than zero. The other choice tanH did give quite similar results as well.

- A plot of your training dataset's objective function's evaluation as a function of the epoch, as well as the validation dataset's evaluation as a function of the epoch. Both plots should be on the same graph with a legend specifying which is which.

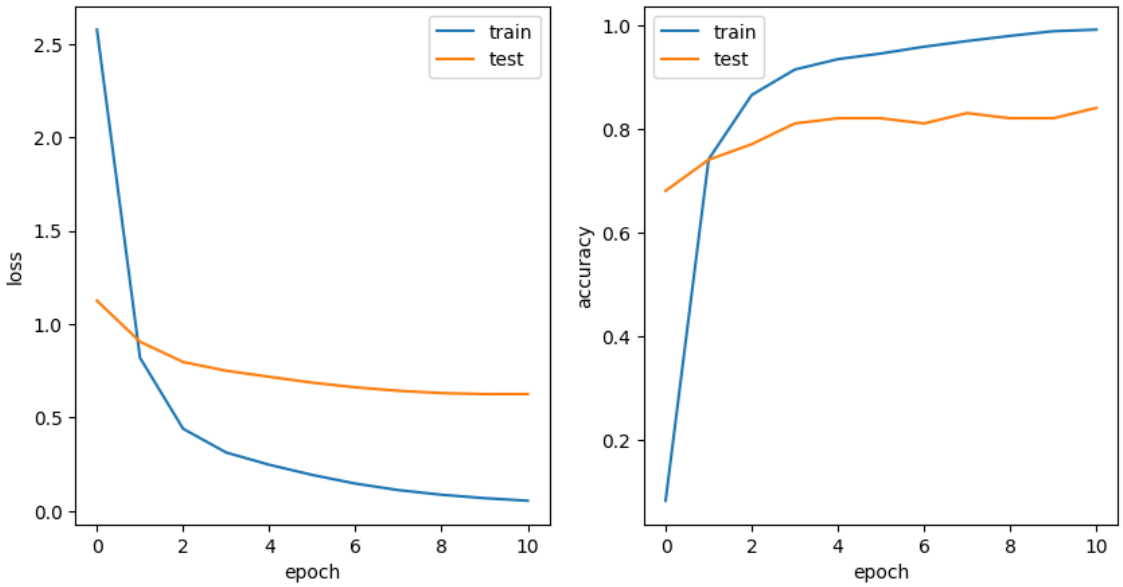

Figure 1: Training the model with early stopping

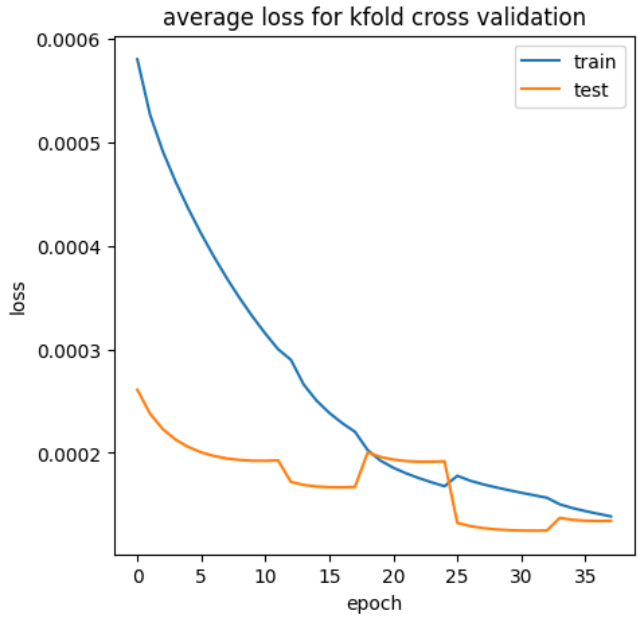

Figure 2: Training the model with early stopping and cross validation

In addition to this model the other model that I tried was implemented using K-folds cross validation. The model performed very well and got an accuracy of

Training Accuracy: 0.847 Testing accuracy: 0.791

- The final training and validation accuracies.

  The final training and testing accuracy for the model were

  Training accuracy: 0.991 Testing accuracy: 0.84

3 Multi-Layer Perceptron

Now let's allow for multiple hidden layers!

Try at least three architectures, each of which have at least two hidden layers (and at least one that has more than two).

In your report, in addition to any hyperparmameter choices, provide a *table* reporting the training and validation accuracies vs the architecture.

Architecture 1:

$IL -> FC(784, 196) -> ReLU -> dropout(0.5) -> FC(196, 98) -> ReLU -> FC(98, 10) -> Softmax -> CrossEntropy$

Architecture 2:

$IL -> FC(784, 392) -> TanH -> dropout(0.2) -> FC(392, 196) -> TanH -> dropout(0.2) -> FC(196, 10) -> Softmax -> CrossEntropy$

Architecture 3:

$IL -> FC(784, 392) -> ReLU -> dropout(0.5) -> FC(392, 196) -> TanH -> dropout(0.5) -> FC(196, 98) -> TanH -> FC(98, 10) -> Softmax -> CrossEntropy$

Architecture	Training Accuracy	Validation Accuracy
Architecture 1	0.968	0.81
Architecture 2	0.984	0.84
Architecture 3	0.997	0.87

Table 2: Comparison of accuracies for different MLP architectures

4 Greedy Layer Pre-training

Now let's try training a deep network using stacked auto-encoders for greedy layer pre-training!

Your final architecture should be:

$$\boxed{\text{Input} \rightarrow \text{FC} \rightarrow \text{Activation} \rightarrow \text{FC} \rightarrow \text{Activation} \rightarrow \text{FC} \rightarrow \text{Activation} \rightarrow \text{Objective}}$$

Your first three fully-connected layers should be *pretrained* using the technique of *stacked auto-encoders* discussed in class. Again, most decisions are up to you, but here's a few comments/hints particular to this problem:

- In general, the input to train each auto-encoder should generated by the previous stage. In particular:
  - The input to train the auto-encoder should be the *zscored* training data.
  - The input to the n^{th} auto-encoder should be the output of the *trained* $(n-1)^{th}$ auto-encoder.

In your report provide:

- The design decisions you made (architecture, hyperparameters, etc..)
- Plot of training and valdiation objective evaluation vs epoch.
- Final training and validation accuracies.

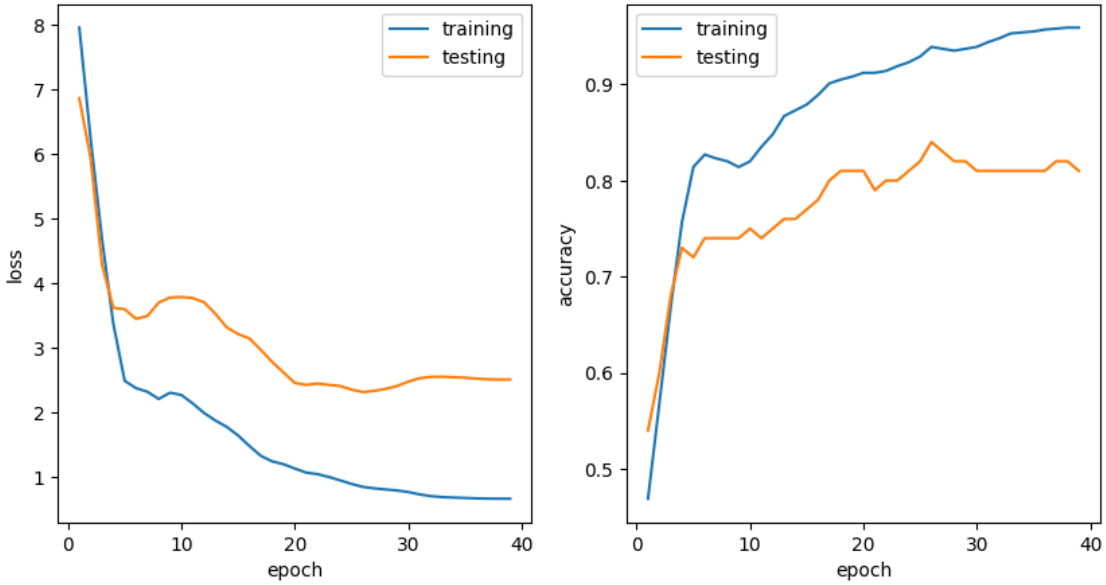

Figure 3: Training the model without early stopping

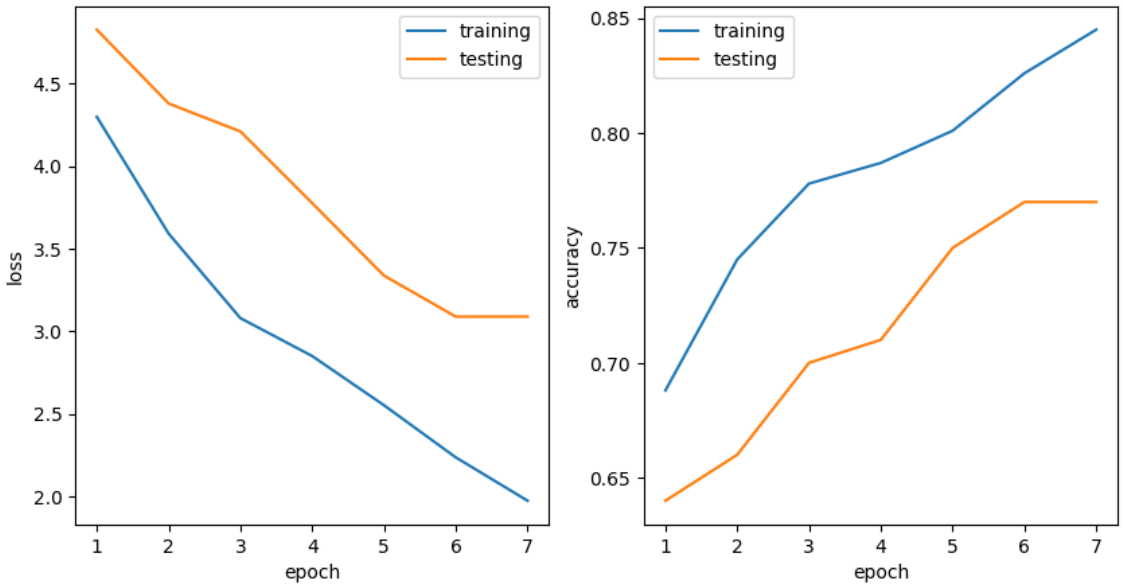

Figure 4: Training the model with early stopping

For the following question after the pre-training of all individual fully connected layer, the architecture followed is

$inputLayer(X) - > fullyConnected(784, 256) - > ReLU - > fullyConnected(256, 128) - > ReLU - > fullyConnected(128, 10) - > Softmax - > CrossEntropyLoss$

for training the following architecture the hyper-parameter choices were, epochs = 40, learning rate = 10e-2, weights and biases initialized using Xavier initialization, and using ADAM optimizer to train the data. For the following problem, the architecture follows two hidden layers which are primarily Fully Connected Layer -¿ ReLU Layer activation. For the output layer, 10 outputs from fully connected layer are provided to the soft-max function and the output is processed by cross entropy loss to calculate the loss. learning rate of 0.01 was used as it gives a faster convergence but also it does not result in exploding gradient. Apart from this I used ADAM Optimizer to deal with any possibility of exploding or vanishing gradient. Moreover 40epochs give optimum accuracy for both the train and test set also avoiding over-fitting. While dealing with over-fitting, I also used early stopping.

without early stopping the training accuracy was: 0.931 and the testing accuracy was: 0.81

Although with early stopping training accuracy is: 0.836 and accuracy for test data was: 0.77

Submission

For your submission, upload to Blackboard a single zip file containing:

1. PDF Writeup
2. Source Code
3. readme.txt file

The readme.txt file should contain information on how to run your code to reproduce results for each part of the assignment.

The PDF document should contain the following:

1. Part 1:
   (a) Answer to the theory questions.
2. Part 2:
   (a) Any hyperparameter choices.
   (b) Your plot of the objective function for the training and validation sets as a function of the training epoch.
   (c) Your final training and validation accuracies.
3. Part 3:
   (a) A list/table of your design choices and resulting training and testing accuracies.
4. Part 4:
   (a) Design and hyperparameter choices.
   (b) Your plot of the objective function for the training and validation sets as a function of the training epoch.
   (c) Your final training and validation accuracies.