Sujay Jakka

Svj0007@auburn.edu

COMP 5600

April 21, 2025

Assignment 5 Report

In this report, I will detail my algorithmic and design choices for Assignment 5. This report will consist of 6 sections, each dedicated to explaining my thought process for each question of the assignment. Furthermore, most of everything that is said in this report is also commented in my code to prevent any confusion.

Question 1

For question 1, we are asked to complete the implementation of the PerceptronModel class. Specifically, we are asked to complete the __init__ fuction, run function, get_prediction, and train functions. Furthermore, we are given a dataset where there are multiple examples that consist of a input vector x and a corresponding label y. This is a classification task where the output labels are either 1 or -1. For the __init__ function, I created a weight vector with shape: 1, dimensions where dimensions is the dimensionality of the input data. For the run function, we calculate the score assigned by the perceptron for an input data point x. Implementing this function was pretty trivial, I just return the dot product using tensor dot between my input data point x and the perceptron weights, self.w. For the prediction function for the PerceptronModel class, we need to return the predicted label given a data point x. The run method gives us some real number for a data point, now we need to assign a class to it. If the value is non-negative we should return 1, else we should return -1. To implement this, I created a simple if statement that called the run function and checked if it was greater than or equal to 0 or if it was negative. If it was nonnegative, the get_prediction function returns 1 else it returns -1. For the train function, our goal is to keep training until we complete an entire pass through the data without misclassifying a data point. We are given a data loader in this function with a batch size of 1 meaning each batch contains only one example. I first created a while loop that loops indefinitely, as we keep looping until we correctly identify all points in a single pass. Next, I create a variable called made_mistake and set it to false to indicate that the perceptron has not mislabeled any data points yet. I then iterate through each data point, and get its label and its perceptron prediction. I find its perceptron prediction using the get_prediction function. I then check if the prediction is the same as the label, if it is not I set made_mistake to true meaning the perceptron has made a mistake in this epoch. I then update my weights by adding to my weights the product of the correct label of the data point and the input vector x of the data point. Outside of the loop where I iterate through all of my data points, the algorithm checks if the perceptron has not made a mistake. If it has not, the training terminates.

Question 2

For question 2, we are asked to implement the RegressionModel class where we design a neural network to approximate the sin(x) function. For this question, we needed to implement the __init__, forward, get_loss, and train functions. The goal is to get a loss of 0.02 or less on modeling this function. In the __init__, this is where I initialized two linear layers each with a size of 128 nodes and an output layer with 1 node. I chose the Adam optimizer with a learning rate of 0.01. In the assignment instructions, they recommended to choose between 1 and 3 layers for the amount of hidden layers so I arbitrarily chose the middle of the range. Next, I chose 128 nodes for all the layers as I felt that would be enough parameters to easily model the simple non linear sin function. I made sure to have relu nonlinearity applied to the output of every layer except for the output of the output layer. For the forward function, we are given our input x which has the shape: batch size, 1. I used the input given to us in the function as input to my network. I returned the output of the neural network which are the predictions, as the output of my forward function. For the get_loss function, I called the forward function to get my predictions then I used the mse_loss function provided to us to find the mean squared error loss of my predictions. I returned this loss in my get_loss function. Lastly, for the train function I used the DataLoader class to create a dataloader with batch size 64. In the assignment instructions, we were told the recommended batch size was from 1 to 128 so I chose a batch size that was half of that. All of the hyperparameters I chosen for batch size, hidden layer size, number of hidden layers, and learning rate were my first choice picks. I happened to get lucky and pass on these initial picks as the function we are modeling is not complex. For each epoch, I iterated through all of the batches. Before I did any operations with the batches, I made sure to reset the gradients with the function optimizer.zero_grad(). I then found the loss of my current batch by calling the get_loss function with my batch input and batch labels. I then used this loss to find the gradients by doing loss.backward(). Lastly, I updated the weights of my neural network using the function optimizer.step(). I trained my model for 300 epochs. I chose this value by experimenting with 50, 100, 150, 200, and 300. I looked to see which epoch hyperparameter value gave the most consistent results and had the least loss. I landed on 300 epochs. Additionally, I was seeing very good results with this current model architecture however approximately every 1 in 15 runs would fail to reach the target loss. To fix this issue, I added one more hidden layer of 128 nodes to my model.

Question 3

For question 3, we are tasked to complete the implementation of the DigitClassificationModel class which will design a neural network to classify handwritten digits of the MNIST dataset. We need to specifically implement the __init__, run, get_loss, and train functions similar to the previous questions. I first tried a similar network to the previous question where I had 2 hidden layers of 128 nodes and an output layer of 10 nodes. I made sure like the previous network in the previous question to have relu applied after every hidden layer. I was not able to hit the 97% level accuracy so I increased the amount of nodes in the two hidden layers to 256. This model performed really well, however at times it would fall just shy of the target accuracy. This is when I decided to just add an extra hidden layer of 128 nodes. After I did this, and experimented with the following learning rates: 0.001, 0.005, and 0.01. I ended up choosing the 0.001 learning rate and I seen very consistent accurate results. The model would yield an accuracy of around 97.5% every time I tested the model after training. For the __init__ function, I just implemented the discussed architecture above including the optimizer using optim.Adam with a learning rate of 0.001. The run function was similar to question 2 where I just fed the input of the function to the neural network. In this case, the input was a tensor with shape batch size, 784 where each example in the dataset had a 784 value vector representing its features. The output of the neural network is what we returned as the output to the run function. This output was a tensor of shape batch size, 10 where each example had scores for each digit 0 through 9 where a higher score meant a higher likelihood that the image was of that digit. In the get_loss function, we are given the input and its corresponding labels. I found the predictions from the neural network by calling the run function. Then I used the imported cross_entropy loss function to find the cross entropy loss between my model's predictions and the correct labels. I then returned this loss. For the train function, this was almost exactly implemented like the train function in question 2 except for a few differences. I only trained for 5 epochs as in the assignment instructions it said that the staff implementation consistently achieved accuracy of 98% after training for 5 epochs and here our target is only 97%. I also set a threshold where after an epoch if the validation accuracy is 98% or over it will stop training. Now lastly, I experimented with batch sizes of 32, 64, and 128 and I saw the best results with the batch size of 128. After doing all of this, my model was able to reach accuracy of on average 97.5% on the testing data after being trained.

Question 4

For question 4, the task is to create a recurrent neural network to identify what language a word is in whether that is English, Spanish, Finnish, etc. We have to implement the LanguageIDModel class specifically the __init__, run, get_loss, and train functions. For the __init__ function I defined two linear layers. One linear layer is to transform the input character data for a timestep with the shape batch size, num_chars to batch_size, hidden_size. The other linear layer is to transform the hidden state but it will retain its dimensions of batch_size, hidden_size. In the __init__ function, I set my hidden_size to 256 as I wanted to make sure there are enough nodes to truly capture the patterns in the data, and I also set the leaning rate for my Adam optimizer to 0.005 after trying 0.01 and getting worse results. The run function was not very complicated. I first found the batch size which can be found from the shape on my input tensor. Then I initialized a tensor of shape batch_size, hidden_size of all zeros which represents my initial hidden state. Next, I iterate through the length of xs, the length of xs is the amount of characters in a word in the batch, and I calculate the new hidden_state by transforming the input character at the timestep with the linear layer W_x and adding it to the transformed hidden_state of linear layer W_h with the current hidden_state. I make sure to apply relu to this hidden_state for nonlinearity similar to the previous questions. After I finish updating the hidden_state after each character, I input the hidden_state in my output layer that results in the tensor that has scores for each language across all examples in the batch. For the get_loss function, similar to question 2 and 3 we just find the predictions using the run() function and then use the cross entropy function with our predictions and labels to find the loss. Lastly, the train function is similar to the train function in question 3 except for a few differences. I set a limit for 20 epochs of training, as in the assignment instructions they describe how it took them between 10-20 epochs to get over 89% accuracy on validation which in this case we only need 81%. Next, our x input in the batches are tensors with the shape of batch_size, length_of_word, num_chars however our get_loss and run functions expect them to be of shape length_of_word, batch_size, num_chars. Before I call my get_loss function, I use the function movedim to change the shape of my input tensor. Next, I stop training once I reach a validation of 84% or over after an epoch. With a batch_size of 64(after trying batch sizes 32, 64, and 128), I am able to get accuracy consistently of 84.1% on the testing set after training.

Question 5

For question 5, we are asked to implement the convolve function and the DigitConvolutionalModel class to create a convolutional neural network to identify handwritten digits in the MNIST dataset. For the convolve function, I first found my input height and input width using input's shape. Next I found my weight height and weight width using the weight's shape. Then I initialized an output tensor of all zeros which we will update throughout the convolution process. The output dimensions are input height – weight height + 1 and input width – weight width + 1 for the output height and output width respectively. Next, I had two nested for loops. The first for loop iterates from 0 to input height – weight height + 1 and the second for loop from 0 to input width – weight width + 1. We do not iterate from 0 to input height or input width as we need to make sure our input patch is big enough for our weight tensor. Inside the second for loop, we find our input patch by offsetting the current row(height) and column(width) value by the weight height and weight width. We then do a dot product with our input patch and weight and update our output tensor. Lastly, we return our output tensor. Now for __init__, forward, get_loss, and train functions for the DigitConvolutionModel most of this implementation can done by reusing our implementation for question 3. For my architecture, I had 2 hidden layers each with 256 nodes and an output layer of 10 nodes. Furthermore, the learning rate and batch_size I selected through trial and error was 0.005 and 128 respectively. In my train function, I made sure to end training if my validation accuracy reached 82% or over after an epoch. The target accuracy for this question was 80%, and I reach an accuracy on average of about 90%.

Question 6

For question 6, we are asked to implement the Attention class specifically the forward function. This was pretty simple as the equation was provided to us in the assignment document. I first transform my input matrix into 3 matrices K, Q, and V using the K linear layer, Q linear layer, and V linear layer provided to us. Then according to the equation provided to us in the assignment details, we have to do a matrix multiplication with K and the transpose of Q. I then transposed the last two dimensions of Q using the movedim function. Next, I calculated my attention scores by doing a matrix multiplication with K and Q and then scaling each of those attention scores with the square root of the layer size. Next, I mask out some of the attention scores using the code provided to us in the comments of the forward function as we do not want our model to look ahead. Furthermore, I then apply a softmax across each row of attention scores tensor. Lastly, I return the matrix multiplication between my attention scores and my V tensore to complete the function.