CMPE442 – Machine Learning HW-2 Report

Zülal Karın 12622989076

Question 1)

The **feedforward()** function takes an input sample and traverses it forward through all layers of the network. In the first step, we add the term bias to the input. Then we move to the first hidden layer using the weights and calculate the outputs in this layer. We add the term bias to the outputs of the hidden layer. Finally, we pass to the output layer using the weights and calculate the outputs. The sigmoid activation function limits the outputs at each layer (between 0 and 1) and produces estimates of the network.

The **backprop()** function performs backpropagation of the mesh and updates the weights. In the first step, the error in the output layer is calculated. This error is the difference between the actual labels and the estimates of the network multiplied by the derived sigmoid function. Then, using this error, the error in the hidden layer is calculated. The error in the hidden layer is obtained by multiplying the transpose of the weights by the error in the output layer. Finally, derivative calculations are made to update the weights. These calculations are performed by calculating the outer product between the outputs of the hidden layer and the error in the output layer.

Question 2.a)

In this experiment , I trained four neural networks with learning rates of 0.01, 0.1, 0.5, and 0.9 and evaluated their performance on a validation set. I found that a learning rate of 0.9 achieved the highest accuracy and is the best choice for training on this dataset.

Dataset:

I used the digits dataset, which has 1,797 samples and 64 features, and divided it into training, validation, and test sets (80:10:10).

Neural Network Architecture:

Our neural network has one hidden layer with a varying number of units. The input size is the number of features in the dataset, and the output size is 10 for the digits dataset. I used the sigmoid activation function in both hidden and output layers.

Training Procedure:

I trained the network using backpropagation and a fixed number of epochs (100). I initialized the weights randomly and updated them using the backpropagation algorithm based on the calculated error. The learning rate determines the step size for weight updates.

Results and Discussion:

Here are the results of the experiment:

```
Learning Rate: 0.01, Accuracy: 0.2708333333333333
Learning Rate: 0.1, Accuracy: 0.5590277777777778
Learning Rate: 0.5, Accuracy: 0.9166666666666666
Learning Rate: 0.9, Accuracy: 0.96875
Best Learning Rate: 0.9
```

I trained four neural networks with different learning rates (0.01, 0.1, 0.5, and 0.9) and evaluated their accuracy on the validation set. The network trained with a learning rate of 0.9 achieved the highest accuracy of 96.875%, while networks with lower learning rates had lower accuracies of 27.083% and 55.903%. The network trained with a learning rate of 0.5 had competitive performance, achieving an accuracy of 91.667%.

Conclusion:

I found that a learning rate of 0.9 is the best choice for training a neural network with one hidden layer on the digits dataset. A higher learning rate allows for faster weight updates and faster convergence, while very low learning rates may lead to slower convergence and suboptimal solutions.

Question 2.b)

In this question, I analyzed the performance of a neural network with one hidden layer on the digits dataset. I investigated the impact of different numbers of hidden units on the network's performance. My findings reveal that a network with **64 hidden units yields the best performance on the validation set.**

I trained the network using backpropagation with a fixed number of epochs (iterNo = 100). The learning rate is fixed at 0.9, which is determined in a previous experiment. The weights are initialized randomly, and the backpropagation algorithm is used to update the weights based on the calculated error.

Here are the results of the experiment:

```
Hidden Units: 16, Accuracy: 0.5972222222222222
Hidden Units: 32, Accuracy: 0.8854166666666666
Hidden Units: 64, Accuracy: 0.96875
Hidden Units: 128, Accuracy: 0.9652777777777778
Best Number of Hidden Units: 64
```

I conducted experiments using 16, 32, 64, and 128 hidden units to investigate the impact of different numbers of hidden units. Each network is trained using the fixed learning rate and evaluated based on its accuracy on the validation set.

Our experiments reveal that the network trained with 64 hidden units exhibits the highest accuracy on the validation set, achieving an accuracy of 96.875%. Networks with 32 and 128 hidden units also demonstrate strong performance, achieving accuracies of 88.541% and 96.528%, respectively. The network with 16 hidden units yields a comparatively lower accuracy of 59.722%.

Choosing a moderate number of hidden units, such as 64, strikes a balance between model complexity and generalization ability, resulting in superior performance on the validation set.

I conclude that choosing an appropriate number of hidden units is essential in achieving better performance on unseen data. Further investigations could explore the use of alternative activation functions and different network architectures to improve performance on more complex datasets.

Question 2.c)

In the third question, I wanted to see how well the neural network did on the test set using the best settings from earlier tests. I found the best learning rate to be 0.9 and the best number of hidden units to be 64. After training the model with the train and validation sets, I tested it on the separate test set. **The model had an accuracy of 94.44%**, meaning it got almost all the answers right.

The output I got is as follows:

```
Test Set Accuracy: 0.9444444444444444
Confusion Matrix:
[[33  0  0  0  0  0  0  0  0  0]
 [ 0 26  1  0  0  0  0  0  0  1]
 [ 0  0 33  0  0  0  0  0  0  0]
 [ 0  0  0 33  0  1  0  0  0  0]
 [ 0  0  0  0 46  0  0  0  0  0]
 [ 0  0  0  0  0 45  1  0  0  1]
 [ 1  0  0  0  0  0 34  0  0  0]
 [ 0  0  0  0  0  1  0 33  0  0]
 [ 0  5  0  0  0  1  0  0 24  0]
 [ 0  1  0  1  1  1  0  3  0 33]]
```

I also made a confusion matrix to see how well the model did for each class. It mostly did well, but there were a few mistakes, especially between classes 1 and 9. Overall, the model did a good job and could be used for things like reading text and recognizing digits.

Even though the model did well, there are still more things we could do to make it even better. For example, we could use more techniques to make the training data more diverse and try different settings for the model. By doing more experiments, we could make the model even more accurate and useful for real-world tasks.