Avery Nelson --- 0 late days used

010920903

amn018@uark.edu

Artificial Intelligence Assignment 4 Report

Description of Implementation

The goal of this assignment was to implement and train neural networks. We focused on three main tasks: binary networks, digit classification, and backpropagation. I was provided with skeleton code that included data generators and high-level training/evaluation frameworks, which I had to complete by implementing the neural networks and training procedures.

The challenging part was checking validity of the responses. They all appear to be on track, but since it's training there is no single "right" answer for when it is acceptable, all I could find were ranges.

In the base code, the BinaryNetwork class was empty with only method signatures. I implemented a simple feed-forward neural network with two fully connected layers. The class takes 2-dimensional inputs (representing the two binary inputs), transforms them to a 4-dimensional hidden representation, and then outputs a single value. It is limited to outputs between 0 and 1, ensuring the values are acceptable for binary classification. For the training framework, I implemented a forward pass to compute outputs, a loss calculation, an accuracy calculation, gradient computation, and weight updates. The key additions to the training loop were the contents of this for-loop: "for i in range(1, n_iters + 1)." I chose the Binary Cross Entropy Loss (BCELoss) as it's great for binary classification tasks. The threshold of 0.5 was used to convert continuous outputs to binary predictions.

As far as the design choices go, I chose a hidden layer size of 4 because XOR requires non-linear separability and thus a hidden layer. Four is sufficient to learn logic operations and is small enough to avoid overloading the computer. A learning rate of 0.1 was given to produce stable convergence for all the operators implemented.

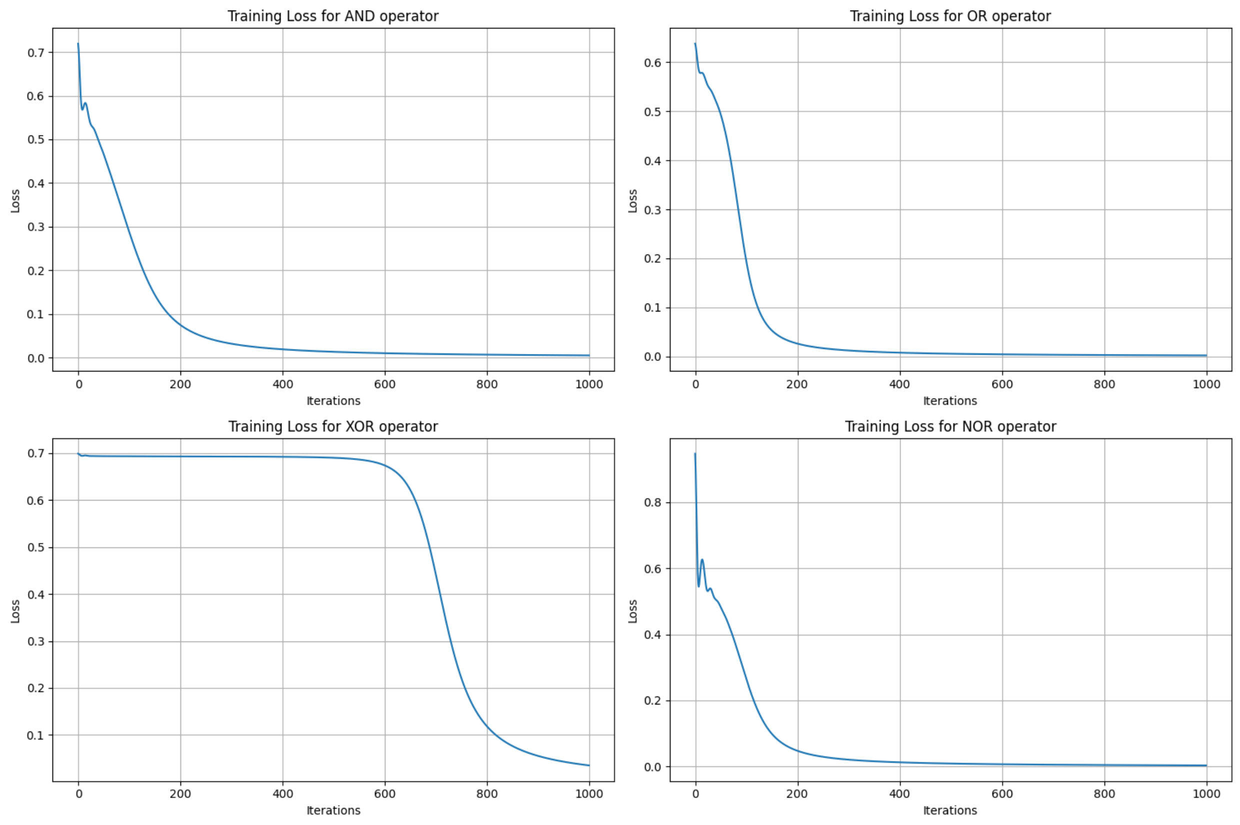
For the MNIST digit classification task, I implemented a Convolutional Neural Network in class DigitNetwork(nn.Module). It processes the images through two convolutional layers, max pooling to reduce dimensionality, flattening, and two fully connected layers with dropout for regularization. I implemented a forward pass through the network, loss calculation using CrossEntropyLoss, accuracy computation, and backpropagation+optimization. I also added timing and created a count_parameters method to analyze the model complexity and output it to the user.

CNNs are ideal for image classification as they capture spatial patterns. I used 3×3 kernels with padding to maintain spatial dimensions. We can also enable the network to learn increasingly complex features by starting with 32 channels and increasing to 64. We also used max pooling, which reduces spatial dimensions, making the model more efficient. I set the learning rate to 0.01 to improve stability and increased the epochs to 3 for better convergence.

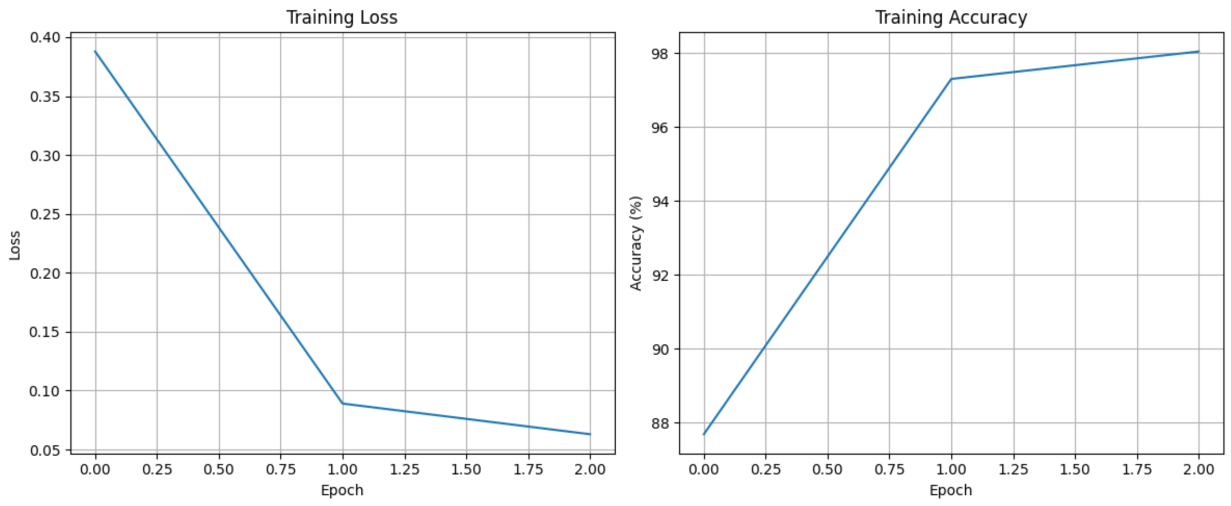
For the custom backpropagation, I implemented custom autograd functions for sigmoid activation and linear transformations. These implementations follow the calculus chain rule for computing gradients. For the sigmoid the gradient is σ(x)*(1-σ(x)) and for the linear layer the gradients correspond to the matrix calculus for $y = xW + b$, where x is the input, y is the output, $W \in \mathbb{R}^{CxD}$ is the weights, $b \in \mathbb{R}^{D}$ and is the bias (C, D are the input and output dimension, respectively). In the sigmoid implementation, I optimized by saving the output directly rather than the input, as the derivative can be expressed in terms of the output value.

Results

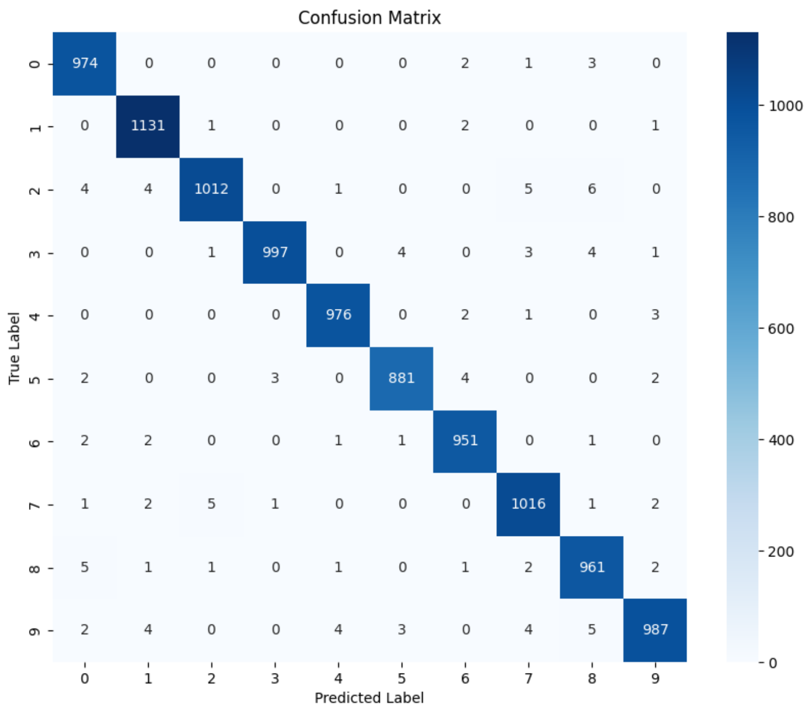
The binary network successfully achieved 100% accuracy on all four logical operations. As expected, non-linear XOR was the most challenging eventually converged. Below is the combined graph of the AND, OR, XOR, and NOR operators' training losses:

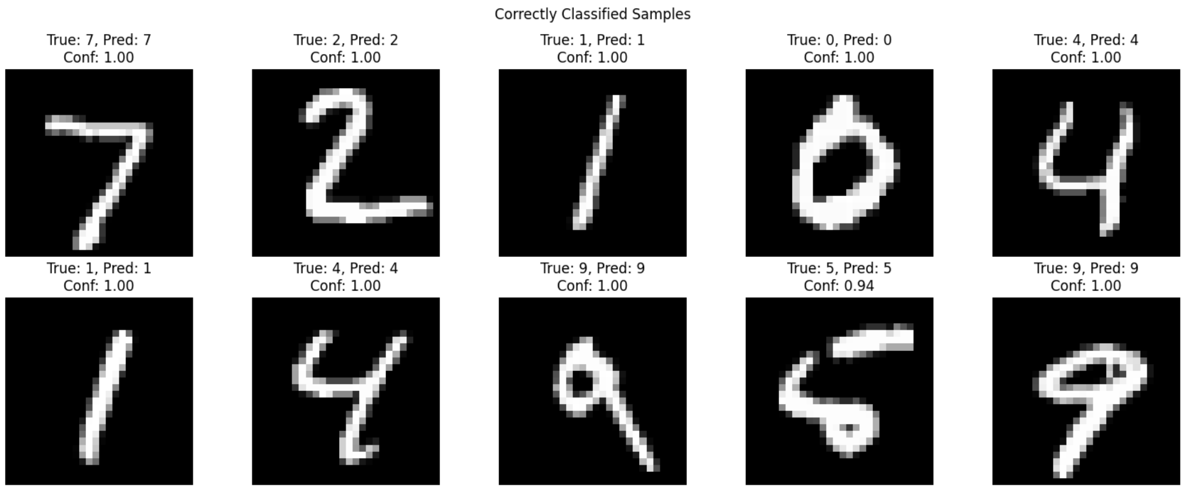


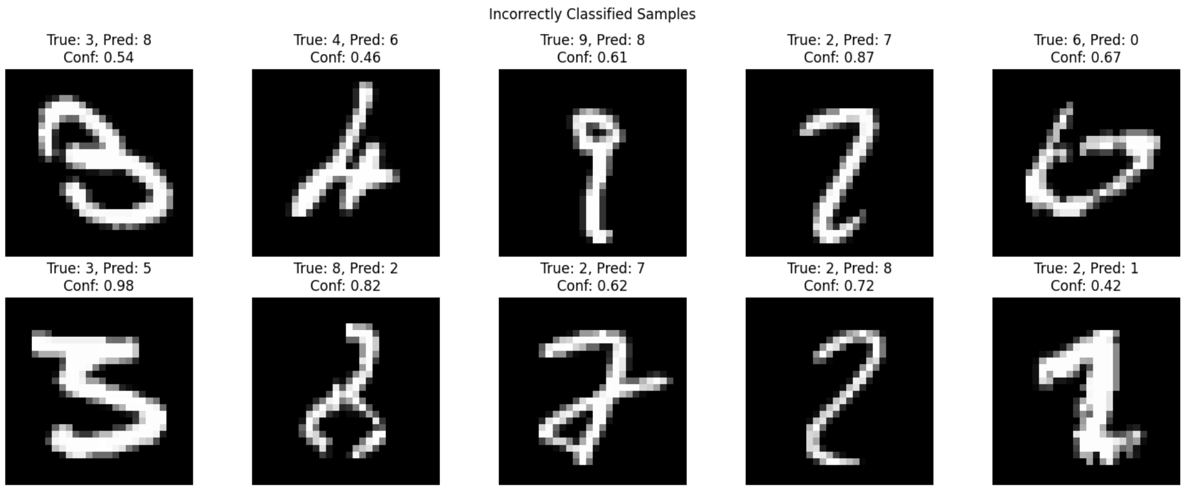
The CNN achieved good results on the MNIST dataset, reaching an accuracy of 98.04% at the end of the third epoch with an average loss of 0.0630, up significantly from the accuracy and average loss at the end of the first epoch, 87.68% and 0.3879 respectively. There were less dramatic improvements between the second and third epochs than the first and second, but they were significant enough that it made sense to use a third. We had a total training time of 97.58 seconds with an average time per batch of 0.0277 seconds. The parameters per layer had the following values: conv1→320, conv2→18496, fc1→401536, fc2→1290, giving a total of 421642. This graph shows the training loss and training accuracy of each epoch of the "Define the training framework” section of the Digit Classification task.



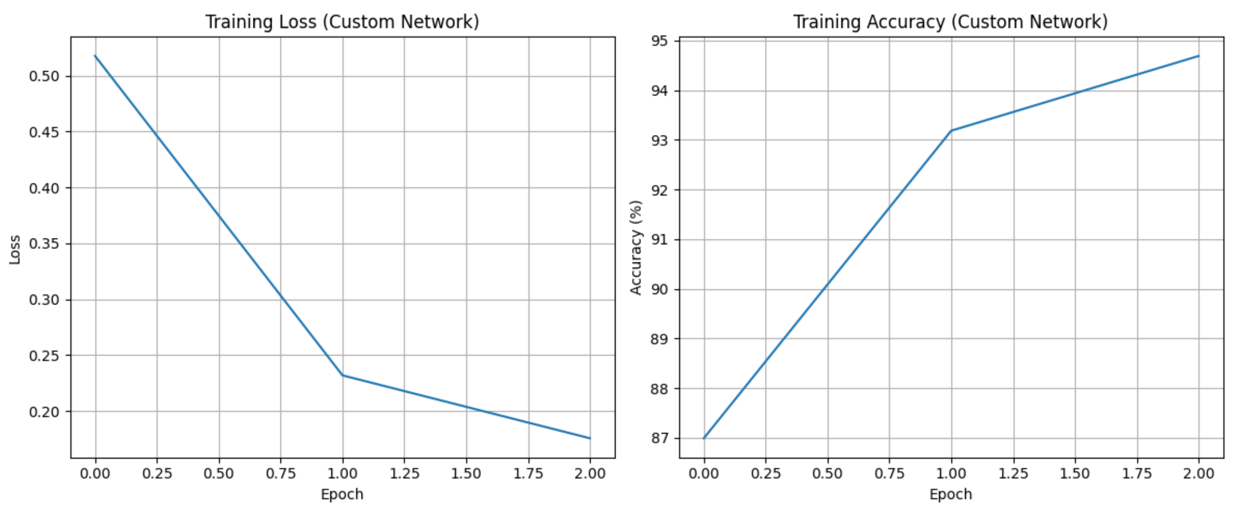
For the evaluation framework section of the digit classification task using a batch size of 1, the final accuracy was 98.86% with an average inference time per sample of 0.00048 seconds. The confusion matrix shows most errors occur between visually similar digits. Looking at some of the incorrectly classified samples, the mix-up is understandable. This confusion matrix and the charts of 10 correctly classified samples and 10 incorrectly classified samples was outputted:

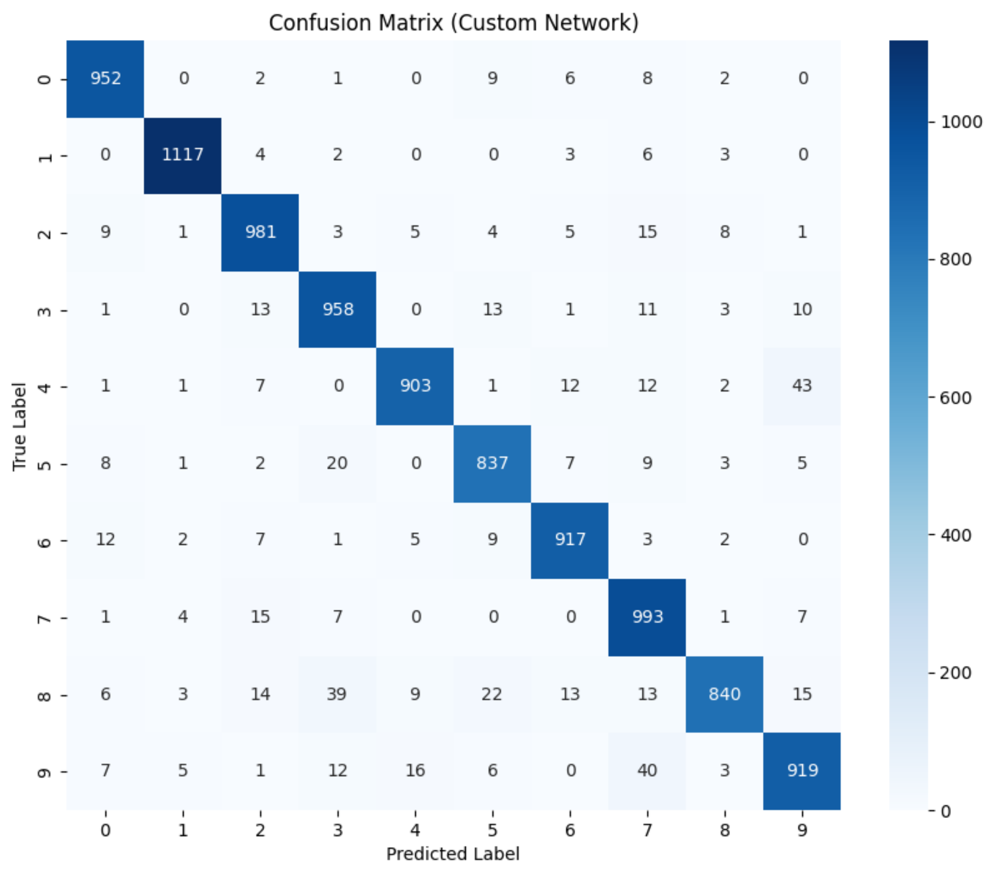






The network with custom backpropagation also performed well, giving an accuracy of 94.69% with an average loss of 0.1756 at the end of the third epoch. This section also shows the significant improvements offered by using multiple epochs, increasing the accuracy from 86.99% to 93.18% then 94.69%. The average loss decreases from 0.5175 to 0.2319 to 0.1756. It has a total training time of 28.14 seconds with an average time per batch of 0.0013 seconds. The final accuracy is 94.17% with an average inference time of 0.00020 seconds. While less accurate with three epochs than the problem above, it does have a smaller average inference time due to the simpler architecture and the accuracy could be improved further. The custom backpropagation shows similar patterns in the confusion matrix, meaning that it is likely also mixing up primarily digits that look similar just as the previous task did. The graph of training loss and accuracy by epoch and the confusion matrix from the "Testing your implementation" section of the Backpropagation task are shown below:





	0	1	2	3	4	5	6	7	8	9
0	952	0	2	1	0	9	6	8	2	0
1	0	1117	4	2	0	0	3	6	3	0
2	9	1	981	3	5	4	5	15	8	1
3	1	0	13	958	0	13	1	11	3	10
4	1	1	7	0	903	1	12	12	2	43
5	8	1	2	20	0	837	7	9	3	5
6	12	2	7	1	5	9	917	3	2	0
7	1	4	15	7	0	0	0	993	1	7
8	6	3	14	39	9	22	13	13	840	15
9	7	5	1	12	16	6	0	40	3	919

I believe my implementation offers successful results from all tasks and thus appears to complete the assignment as instructed.