Bradley Stanley-Clamp
Git repository: neural_network_assignment

ml-iearth-mt2024: Assessed Assignment

Introduction

This report summarises the work done in implementing a neural network in python. This task was successfully achieved, resulting in model with an accuracy performance of 86% on the final test dataset.

The report is laid out as follows, firstly the initial design choices when approaching this project are described and explained. The model architecture is then described, followed by the loss function and optimisation techniques. The results of the model are then presented.

Design Choices

As well as developing my understanding of neural networks, I wanted this project to also develop my programming, version control, and testing skills. I used object-oriented programming, enforced version control with Git and used a pytest testing framework.

To develop my knowledge and skills with pytorch, I chose to use pytorch tensors as the underlying data type of the model.

The data chosen for this project was MNIST, due to its size and accessibility. As a result, the project was a multi class classification problem. Data was loaded and batched using torchvision's datasets and pytorch's data loader.

Model Architecture

Based on online examples such as [1], the model architecture was chosen to consist of five layers, three linear layers and two ReLU layers (Figure 1). No activation function was put at the output of the last linear layer as categorical cross entropy includes a softmax function. A flatten function was created to convert the data into the correct format for the linear model. Backpropagation was manually implemented for linear and activation layers via a method in the respective layers class.

Regarding weight initialisation, this was first implemented randomly, however, this resulted in unusable model output values. As a solution, He initialization [2] was implemented.

```
class Linear_model:

    def __init__(self) -> None:

        init_mode = "HeInit"

        self.fc1 = Linear_layer(784, 128, init_mode=init_mode)

        self.relu1 = ReLU()

        self.fc2 = Linear_layer(128, 64, init_mode=init_mode)

        self.relu2 = ReLU()
```

Figure 1: Linear model code structure

Bradley Stanley-Clamp
Git repository: neural_network_assignment

Loss function

As this is a multi-class classification problem, categorical cross entropy was chosen. This was implemented in two separate steps, first a softmax function to get a probabilistic output of the model, followed by a negative log likehood to calculate the loss [3]. This technique was based on the implementation of pytorch's categorical cross entropy.

Optimisation

Initially, stochastic gradient decent was used, however, to improve performance, Adam was implemented [4]. The parameters referenced in the original paper [4] were used as they resulted in good performance.

Results

The parameters chosen can be seen in Table 1. After training (Figure 2) it was clear the model optimised quickly, as a result 16 epochs were chosen to train the model.

Parameter	**Value**
Epochs	16
Batch size	64
Weight initialisation	He Initilisation
Optimiser	Adam

Table 1: Parameters and corresponding values

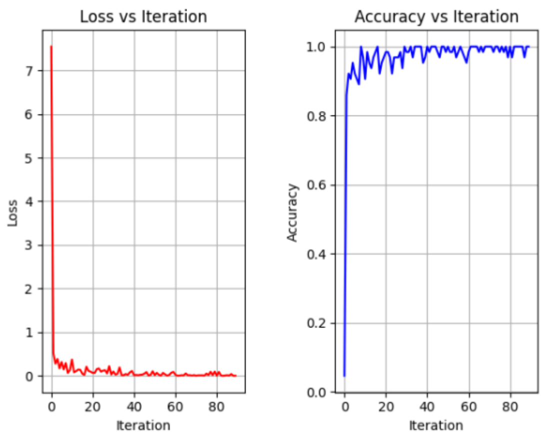

Figure 2: Loss and Accuracy against iteration during training process

Evaluating on the test dataset, the model performed well, the evaluation results are shown in Table 2.

Metric	**Value**
Accuracy	0.857
Precision	0.851
Recall	0.908
F1	0.855

Table 2: Evaluation of test dataset on trained model

Conclusion

In conclusion, I successfully built a linear neural network that can categorise unseen MNIST images with an accuracy of 86%. Future work could include, convolutional layers and parallelising training.

Bradley Stanley-Clamp
Git repository: neural_network_assignment

References

[1] R. Gopalakrishnan, ‘A Simple Neural Network Model For MNIST Using PyTorch’, Medium. Accessed: Jan. 12, 2025. [Online]. Available: https://medium.com/@ramamurthi96/a-simple-neural-network-model-for-mnist-using-pytorch-4b8b148ecbdc

[2] K. He, X. Zhang, S. Ren, and J. Sun, ‘Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification’, Feb. 06, 2015, *arXiv*: arXiv:1502.01852. doi: 10.48550/arXiv.1502.01852

[3] L. MIRANDA, ‘Understanding softmax and the negative log-likelihood’, Lj Miranda. Accessed: Jan. 12, 2025. [Online]. Available: https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

[4] D. P. Kingma and J. Ba, ‘Adam: A Method for Stochastic Optimization’, Jan. 30, 2017, arXiv: arXiv:1412.6980. doi: 10.48550/arXiv.1412.6980.