Intro to Deep Learning Assignment 2

Tsinghua University
Allan Li (李盼加乐), 2025403473

The purpose of this report is to evaluate the performance of a Multi-Layer Perceptron (MLP) in classifying handwritten digits from the MNIST dataset. The MNIST dataset consists of grayscale images of digits (0–9), each standardized into a 28×28 pixel grid and represented as a 784-dimensional input vector. The task is formulated as a multiclass classification problem, where the goal is to correctly identify each image's digit label among ten possible classes.

In this experiment, different configurations of the MLP architecture are explored, including variations in activation functions (Sigmoid vs. ReLU), loss functions (Euclidean Loss vs. Softmax Cross-Entropy Loss), and network depth (one hidden layer vs. two hidden layers). The analysis focuses on comparing these design choices in terms of training success, efficiency, and the accuracy on both training and testing sets.

Summary of Results

Four different base configurations (alongside a custom configuration) were developed to attempt to classify the MNIST dataset. The following data is using base hyperparameters provided in the assignment.

```
batch_size = 100
max_epoch = 20
init_std = 0.01

learning_rate_SGD = 0.001
weight_decay = 0.1

disp_freq = 50
```

Testing accuracy results are shown below:

Activation Function \ Loss Function	Euclidean Loss	Softmax CrossEntropy-Loss
Sigmoid	0.7814	0.6955
ReLU	0.8521	0.8683

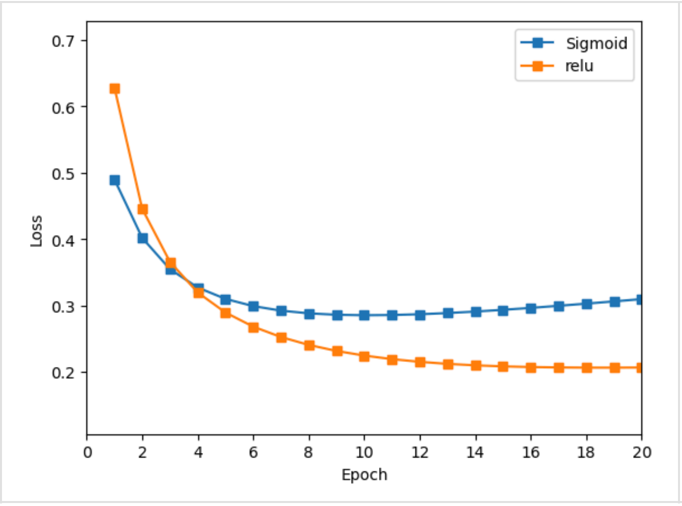
The best performing combination of activation and loss function was ReLU and Softmax CrossEntropy-Loss, with a final testing accuracy of 0.8683.

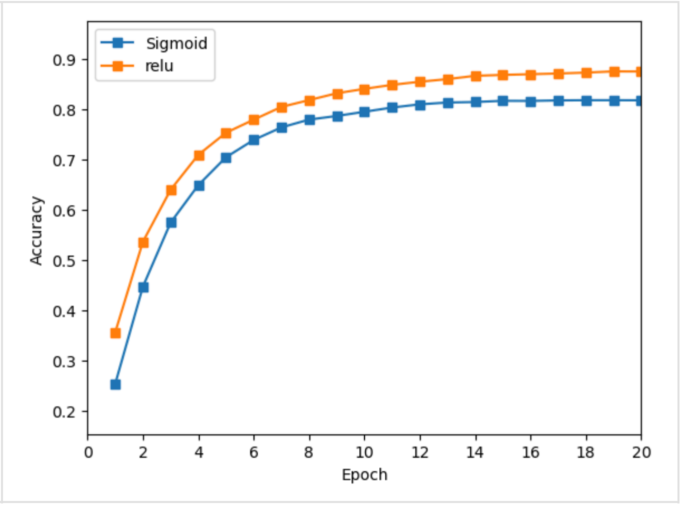
Moreover, training times for the different configurations were also provided (in seconds):

Activation Function \ Loss Function	Euclidean Loss	Softmax CrossEntropy-Loss
Sigmoid	43.8s	63.8s
ReLU	53.9s	79.4s

The best performing combination of activation and loss function w.r.t time was Sigmoid and Euclidean Loss, with a final training time of 43.8s.

Analysis of different Activation Functions (Sigmoid vs ReLU)





```python
# sigmoid layer implementation
self.Output = 1 / (1 + np.exp(-Input))

# relu layer implementation
self.Output = np.maximum(0, Input)
```

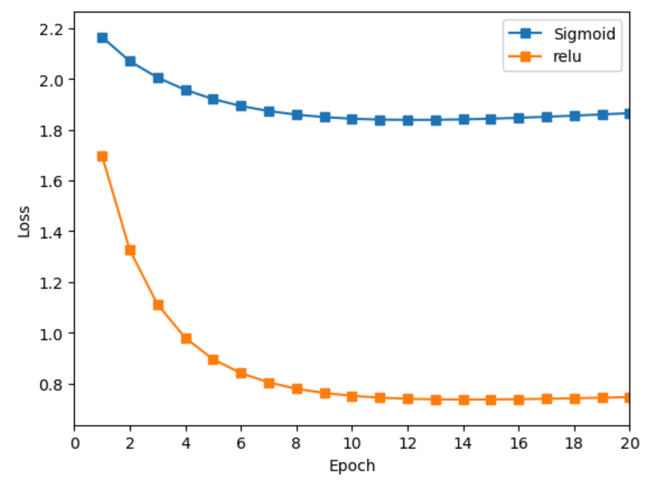
1. Training Time
   The sigmoid activation function,

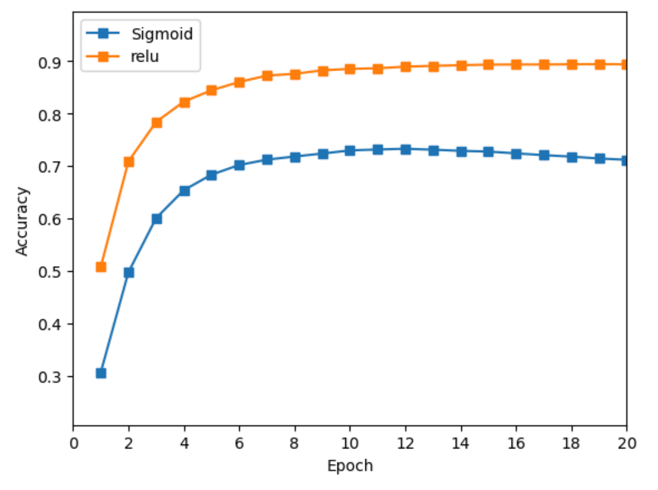
$$f(x) = \frac{1}{1 + e^{-x}}$$

   involves exponential computation, which makes it a lot more computationally expensive than ReLU. Moreover, many neurons can approach 0 or 1, causing gradients to become very small and causing slower learning. However, within our tests, ReLU performed worse than the sigmoid function in both cases of activation functions. This discrepancy should be investigated further, but might be the result of implementation differences.

2. Convergence Behaviour
   The gradient of Sigmoid shrinks dramatically for large positive or negative inputs. Thus, intuitively, the network probably gets stuck in flat regions of the loss surface, making convergence slower and less stable. In the later layers, weight updates tend to become very small. Conversely, ReLU mitigates this issue by maintaining a constant gradient for positive inputs. This causes more stable and faster convergence. The result of this is that ReLU converges faster and more stably, compared to Sigmoid.
3. Accuracy
   Due to the previous issues mentioned above, networks using Sigmoid activations underfit and achieve lower final accuracy, which is evident in the accuracy plot between Sigmoid and ReLU. It seems that ReLU is able to learn better and more complex feature representations.

Analysis of different Loss Functions (Euclidean Loss vs Softmax Cross-Entropy)





The Euclidean Loss measures the squared difference between predicted and target outputs, which treats classification as a regression problem. This leads to potentially weaker gradients that might happen if predictions approach 0 or 1, which result in slower convergence and lower accuracy overall.

In contrast, the Softmax Cross-Entropy Loss interprets the outputs as probabilities and is able to punish incorrect confident predictions better than the former loss algorithm. This produces sharper gradients and leads to faster, more stable learning.

Empirically, from the data collected during this experiment, models with Cross-Entropy achieved higher accuracy but required longer training times due to additional complicated operations like log and exponential functions.

Noticeably, across the entire experiment, Softmax + ReLU performed the best, while Softmax + sigmoid performed the worst. The latter performance is apparent. The reasoning is because sigmoid activations saturate and compress activations into a narrow range, which weaken the gradient signal passed to the Softmax layer, and this might explain the worse performance overall.

Hyperparameter Tuning

It is evident that by the results, there is a need for hyperparameter tuning, given that the highest accuracy accross the board was still within the mid 80s. The training curves suggest moderate underfitting, meaning that the model hasn't been able to capture enough complexity.

Some suggestions made were as follows:

```
# new set of hyperparameters
batch_size = 128
max_epoch = 50
init_std = 0.01

learning_rate_SGD = 0.005
weight_decay = 1e-4

disp_freq = 100
```

Weight decay was decreased because it was too strong, and this was to prevent over-regularization. Moreover, increasing the learning rate to `0.005` would allow the model to learn faster early on while still stabilizing later. Increasing the training epochs from 20 to 50 would also give the network more opportunity to refine parameters once the training progress starts to plateau.

Additionally, increasing the batch size from 100 to 128 makes gradient updates smoother.

Thus, the following results were now reported given the new hyperparameters:

Activation Function \ Loss Function	Euclidean Loss	Softmax CrossEntropy-Loss
Sigmoid	0.8732	0.9057
ReLU	0.9268	0.9457

This represents a significant increase to all variations of the models.

Moreover, training time increased significantly, which makes sense given the increased amount of epochs.

Activation Function \ Loss Function	Euclidean Loss	Softmax CrossEntropy-Loss
Sigmoid	103.7s	106.2s
ReLU	106.2s	120.02s

Designing a Custom Model

Since the best performing model across the previous 4 was the ReLU + Softmax combination, it was decided that intuitively, the most appropriate two-layer model would consist of two ReLU layers, using the Softmax Loss Function.

The implementation was as follows:

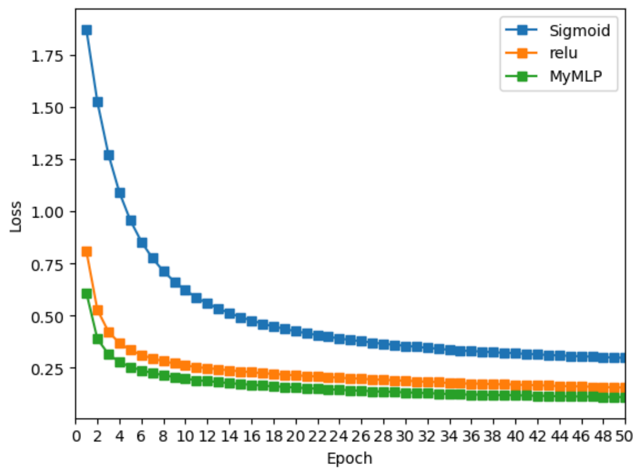
```
myMLP = Network()

myMLP.add(FCLayer(784, 256))
myMLP.add(ReLULayer())
myMLP.add(FCLayer(256, 128))
myMLP.add(ReLULayer())
myMLP.add(FCLayer(128, 10))

criterion = SoftmaxCrossEntropyLossLayer()
```

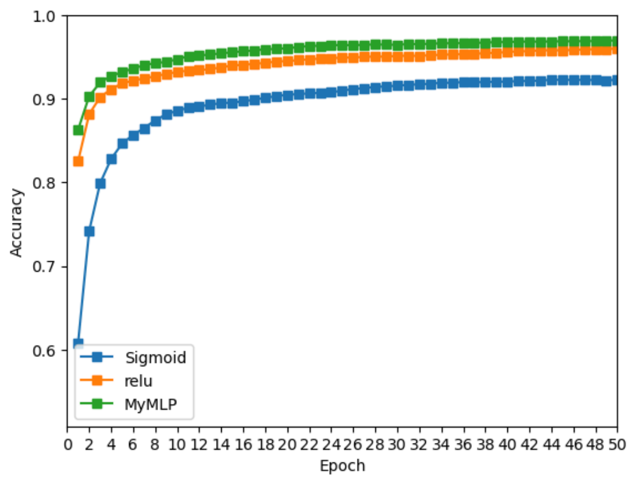
Moreover, the node counts for each layer was somewhat arbitrarily determined, but also inspired by other literature of simple MLPs.

The performance of this model (compared to the one layer ReLU + Softmax model) is as follows (all using the hyperparameters specified in the previous section):

Model	Testing Accuracy	Testing Time
ReLU + Softmax	0.9457	120.02s
MyMLP	0.9625	248.1s

As well, the Loss and Accuracy graphs were as follows:





As seen both graphically and numerically, the improvement from adding a second hidden layer is relatively insignificant. Our custom model achieved a testing accuracy of **0.9625**, a ~1% increase compared to the baseline model (ReLU + Softmax). Despite the fact that this represents a measurable improvement, it suggests that a single layer networks is already capable of capturing most of the complexity within the MNIST dataset.

This observation is intuitive: handwritten digits are comparatively simpler than other classification tasks involving objects like animals or foodstuffs. Moreover, the two layer model roughly doubled the training time, which is a significant tradeoff between model complexity and computational efficiency.

In the author's opinion, the baseline model offers a better balance between accuracy, training time, and simplicity (compared to the custom two-layer model).