18-786

Homework 1

January 24, 2024

1 Question 1

- Filter all samples representing digits '0' or '1' from the MNIST datasets.
- Randomly split the training data into a training set (80% training samples) of a validation set (20% training samples).
- Define an MLP with 1 hidden layer and train the MLP to classify the digits '0' vs '1'. Report your MLP design and training details (which optimizer, number of epochs, learning rate, etc.)
- Keep other hyper-parameters the same, and train the model with different batch sizes: 2, 16, 128, 1024. Report the time cost, training, validation and test set accuracy of your mode

Solution:

1.1 Model for binary classificaiton

```
print(model)

SimpleMLP(
    (fc1): Linear(in_features=784, out_features=5, bias=True)
    (activation): Sigmoid()
    (fc2): Linear(in_features=5, out_features=2, bias=True)
)
```

Effect of batch performance on the model performance and train time

[redacted]	Training Time (s)	Train Acc	Val Acc	Test Acc
2	16.724894	99.911173	99.684169	99.669031
16	5.800792	99.960521	99.842084	99.952719
128	4.362510	99.891433	99.881563	99.905437
1024	3.937332	99.595341	99.526253	99.574468

Table 1: Comparing the performance of SimpleMLP model for different batch sizes 2, 16, 128, 1024

□

2 Question 2

- Implement the training loop and evaluation section. Report the hyper-parameters you choose
- Experiment with different numbers of neurons in the hidden layer and note any changes in performance.
- Write a brief analysis of the modelâs performance, including any challenges faced and how they were addressed

Solution:

2.1 Model for multi-class (10 digits) classificaiton

```
print(model)
hidden_dim = int(np.sqrt(28*28*10)) # changed in hyperopt

MulticlassMLP(
    (fc1): Linear(in_features=784, out_features=88, bias=True)
    (activation): Sigmoid()
    (fc2): Linear(in_features=88, out_features=10, bias=True)
)

# criterion = nn.CrossEntropyLoss()
```

2.2 Hyper parameter optimization for Multi-Class Optimization

I first experimented with device to see the most optimal device for training between cpu and the gpu on my Mac-M1 Pro. The results are given in the appendix for the model with sigmoid. For each of them we found that **device** = **'cpu'** was faster than **device** = **'mps'** (mac gpu in M1 Pro).

2.2.1 Results:

Two models were optimized for this task, one with sigmoid activiation and the other with ReLU activation.

1. **Sigmoid Activation Layer**

```
print(model)
hidden_dim = int(np.sqrt(28*28*10))
# hidden_dim changed in hyperopt later
MulticlassMLP(
    (fc1): Linear(in_features=784, out_features=88,
    bias=True)
    (activation): Sigmoid()
    (fc2): Linear(in_features=88, out_features=10,
    bias=True)
)
# criterion = nn.CrossEntropyLoss()
```

The hyper-parameters tested are as follows

```
batch_sizes = [64, 128, 1024]
optimizers = ['adam', 'sgd']
# learning_rates= [1e-4, 1e-3, 1e-2, 1e-1]
hidden_dims = [4, 32, 64, 128]
```

Opt#	Batch	Opt	LR	HidDim	Training Time	Train Acc	Val Acc	Test Acc
0	64	adam	0.001	4	20.822481	81.116667	80.50	81.34
1	64	adam	0.001	32	21.138930	95.725	94.583333	94.83
2	64	adam	0.001	64	22.073092	97.208333	95.566667	96.09
3	64	adam	0.001	128	22.682774	98.10	96.591667	96.87
4	64	sgd	0.01	4	19.016016	69.941667	70.083333	70.49
5	64	sgd	0.01	32	19.240644	89.883333	89.666667	90.33
6	64	sgd	0.01	64	19.526299	90.029167	89.866667	90.57
7	64	sgd	0.01	128	20.448700	90.037500	89.85	90.69
8	128	adam	0.001	4	18.587870	74.941667	73.933333	75.06
9	128	adam	0.001	32	18.807421	95.029167	94.341667	94.47
10	128	adam	0.001	64	19.121340	96.412500	95.416667	95.75
11	128	adam	0.001	128	20.281624	97.404167	96.433333	96.45
12	128	sgd	0.01	4	17.898255	63.181250	63.475	64.90
13	128	sgd	0.01	32	17.976289	87.287500	87.291667	88.10
14	128	sgd	0.01	64	18.428340	87.922917	87.775	88.92
15	128	sgd	0.01	128	18.753883	87.927083	87.90	88.73
16	1024	adam	0.001	4	17.380155	52.65	52.416667	53.09
17	1024	adam	0.001	32	17.465012	90.939583	90.625	91.21
18	1024	adam	0.001	64	17.708344	92.270833	91.866667	92.49
19	1024	adam	0.001	128	18.143819	93.297917	92.766667	93.21
20	1024	sgd	0.01	4	17.723570	20.575	22.00	23.66
21	1024	sgd	0.01	32	17.807459	62.589583	63.075	65.17
22	1024	sgd	0.01	64	18.086973	67.327083	66.508333	68.73
23	1024	sgd	0.01	128	18.751791	70.679167	70.291667	71.88

Table 2: Hyperopt results for different optmizers, learning rate, batch size, and hidden dimension of the MulticlassMLP Network with sigmoid activation layer

2. **ReLU Activation Layer**

```
class MulticlassMLP(nn.Module):
    def __init__(self, in_dim, hidden_dim, out_dim):
        super(MulticlassMLP, self).__init__()
        self.fc1 = nn.Linear(in_dim, hidden_dim)
        self.activation = nn.ReLU()
        self.fc2 = nn.Linear(hidden_dim, out_dim)

    def forward(self, x):
        # Your code goes here
        x = self.fc1(x)
        x = self.activation(x)
```

```
        x = self.fc2(x)

        return x

# criterion = nn.CrossEntropyLoss()
```

Opt#	Batch	Opt	LR	HidDim	Training Time	Train Acc	Val Acc	Test Acc
0	64	adam	0.001	4	22.202067	33.362500	33.991667	34.13
1	64	adam	0.001	32	22.753937	95.647917	94.566667	95.12
2	64	adam	0.001	64	23.678958	97.191667	96.525	96.80
3	64	adam	0.001	128	24.853068	98.108333	96.425	96.53
4	64	sgd	0.01	4	21.199240	82.979167	82.091667	82.65
5	64	sgd	0.01	32	23.314560	92.931250	92.458333	92.94
6	64	sgd	0.01	64	24.677200	93.55	93.05	93.49
7	64	sgd	0.01	128	23.671364	93.885417	93.608333	94.10
8	128	adam	0.001	4	20.069331	65.714583	65.108333	66.38
9	128	adam	0.001	32	21.104044	93.787500	92.925	93.34
10	128	adam	0.001	64	22.600291	96.437500	95.45	95.97
11	128	adam	0.001	128	24.154394	97.735417	96.30	96.72
12	128	sgd	0.01	4	20.192016	81.056250	80.933333	81.43
13	128	sgd	0.01	32	19.052993	91.445833	91.433333	92.05
14	128	sgd	0.01	64	19.173184	91.566667	91.241667	91.92
15	128	sgd	0.01	128	20.031956	91.764583	91.458333	92.31
16	1024	adam	0.001	4	17.754258	40.585417	41.825	41.90
17	1024	adam	0.001	32	17.710765	91.395833	91.466667	91.97
18	1024	adam	0.001	64	18.001581	93.037500	92.708333	93.05
19	1024	adam	0.001	128	18.118424	94.625	94.158333	94.60
20	1024	sgd	0.01	4	17.477235	53.318750	53.15	54.65
21	1024	sgd	0.01	32	17.538370	85.191667	84.941667	85.86
22	1024	sgd	0.01	64	17.841422	86.214583	86.075	87.12
23	1024	sgd	0.01	128	18.008393	86.645833	86.491667	87.38

Table 3: Hyperopt results for different optmizers, learning rate, batch size, and hidden dimension of the MulticlassMLP Network with ReLU activation layer

2.2.2 Conclusion

From **??** and **??**, we see that the highlighted rows.

The **yellow** rows highlight the highest performing hyper-parameters for 'adam', highest performing hyper-parameters for 'sgd' are highligted in **orange**, and **red** highligts the worse performances across the two optimizers.

- Sigmoid Activation

  for batch size, optimizer, learning rate, hidden dimension, 64 adam 0.001 128 22.682774 98.10 96.591667 96.87

  We get a train accuracy of 97.19%, validation accuracy of 96.52%, and the test accuracy of 96.80%.

- Sigmoid Activation

  3 64 adam 0.001 128 22.682774 98.10 96.591667 96.87

  We get a train accuracy of 98.10%, validation accuracy of 96.59%, and the test accuracy of 96.87%.
- Sigmoid Activation

  7 64 sgd 0.01 128 23.671364 93.885417 93.608333 94.10

  We get a train accuracy of 93.88%, validation accuracy of 993.60%, and the test accuracy of 94.10%.

□